Laboratory Manual
2023-24

Sub: Compiler Construction

T.Y. B.Tech. (CSE)

Prepared By,

Prof. Kamble D.R.

G.K. Gujar Memorial Charitable Trust's

DR.ASHOK GUJAR TECHNICAL INSTITUTE'S

DR. DAULATRAO AHER COLLEGE OF ENGINEERING,KARAD

Vidyanagar Extn. Banawadi, Dist. Satara

Prepared by :Ms. Kamble D.R.	Approved by: Head of Department

G.K. Gujar Memorial Charitable Trust's

DR.ASHOK GUJAR TECHNICAL INSTITUTE'S

DR. DAULATRAO AHER COLLEGE OF ENGINEERING, KARAD

Lab Manual

Subject : Compiler Construction

TITLE : Index (List of Experiment)

Sr. No	Name of Experiment
01	Design of preprocessor for C program.
02	Design of lexical analyzer for C language.
03	Program to create a symbol table generator.
04	Program to implement recursive descent parsing method for simple expression.
05	Program to create a syntax tree for simple expression techniques.
06	Program to implement intermediate code generator for the Boolean expression in three address code format.
07	Program to implement FIRST and FOLLOW for given grammar.
08	Program to implement code optimization for given statements.
09	Program to implement code generator for the statements in three address code format.
10	Program to implement LEX and YACC program using FLEX and YACC.

Prepared by :Ms. D.R.Kamble	Approved by: Head of Department

G.K. Gujar Memorial Charitable Trust's **DR.ASHOK GUJAR TECHNICAL INSTITUTE'S** **DR. DAULATRAO AHER COLLEGE OF ENGINEERING, KARAD**	
Lab Manual	
Subject : Compiler Construction	
Expt. No: 01	**TITLE:** Design of preprocessor for C program.

Aim: Design of preprocessor for C program.

Theory:

The C preprocessor is a macro preprocessor (allows you to define macros) that transforms your program before it is compiled. These transformations can be inclusion of header file, macro expansions etc.

All preprocessing directives begin with a # symbol. For example,

#define PI 3.14

Including Header Files

The #include preprocessor is used to include header files to a C program. For example,

#include<stdio.h>

Here, "stdio.h" is a header file. The #include preprocessor directive replaces the above line with the contents of stdio.h header file which contains function and macro definitions.

That's the reason why you need to use #include <stdio.h> before you can use functions like scanf() and printf().

You can also create your own header file containing function declaration and include it in your program using this preprocessor directive.

#include "my_header.h"

Macros using #define

You can define a macro in C using #define preprocessor directive.

A macro is a fragment of code that is given a name. You can use that fragment of code in your program by using the name.

For example,

#define c 299792458 // speed of lightUsing

#define preprocessor

```
#include <stdio.h>
#define PI 3.1415
```

```
int main()
{
   float radius, area;
   printf("Enter the radius: ");
   scanf("%d", &radius);    // Notice, the use of PI
   area = PI*radius*radius;
   printf("Area=%.2f",area);
   return 0;
}
```

Using #define preprocessor

```
#include <stdio.h>
#define PI 3.1415
#define circleArea(r) (PI*r*r)
int main()
{
   int radius;
   float area;
   printf("Calculate area of circle:");
   printf("Enter  radius: ");
   scanf("%d", &radius);
   area = circleArea(radius);
   printf("area = %.2f", area);
return 0;
}
```

Output:

```
    ile   dit   earch   un   ompile   ebug
[■]═══════════════════════ Output ═══
Calculate area of circle:
Enter radius:10

area =  314.00_
```

Conclusion:

Thus, we have successfully implemented preprocessor # define to calculate area of circle.

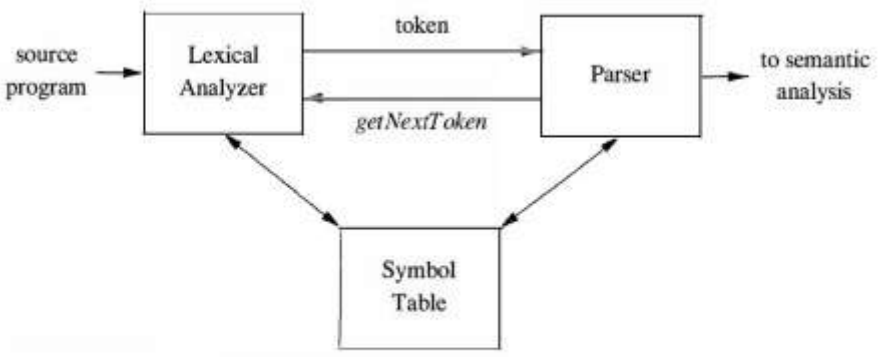
G.K. Gujar Memorial Charitable Trust's **DR.ASHOK GUJAR TECHNICAL INSTITUTE'S** **DR. DAULATRAO AHER COLLEGE OF ENGINEERING, KARAD**	
Lab Manual	
Subject : Compiler Construction	
Expt. No: 02	**TITLE:** Design of lexical analyzer for C language.

Aim: Design lexical analyzer for C language.

Theory:

The main task of the lexical analyzer is to read the input characters of the source program, group them into lexemes, and produce as output a sequence of tokens for each lexeme in the source program. The stream of tokens is sent to the parser for syntax analysis. It is common for the lexical analyzer to interact with the symbol table as well. When the lexical analyzer discovers a lexeme constituting an identifier, it needs to enter that lexeme into the symbol table. In some cases, information regarding the kind of identifier may be read from the symbol table by the lexical analyzer to assist it in determining the proper token it must pass to the parser.

Commonly, the interaction is implemented by having the parser call the lexical analyzer. The call, suggested by the *getNextToken* command, causes the lexical analyzer to read characters from its input until it can identify the next lexeme and produce for it the next token, which it returns to the parser.



Interactions between the lexical analyzer and the parser

Since the lexical analyzer is the part of the compiler that reads the source text, it may perform certain other tasks besides identification of lexemes. One such task is stripping out comments and whitespace (blank, newline, tab, and perhaps other characters that are used to separate tokens in the input). Another task is correlating error messages generated by the compiler with the source program. For instance, the lexical analyzer may keep track of the

number of newline characters seen, so it can associate a line number with each error message. In some compilers, the lexical analyzer makes a copy of the source program with the error messages inserted at the appropriate positions. If the source program uses a macro-preprocessor, the expansion of macros may also be performed by the lexical analyzer.

TOKEN	INFORMAL DESCRIPTION	SAMPLE LEXEMES
if	characters `i`, `f`	`if`
else	characters `e`, `l`, `s`, `e`	`else`
comparison	< or > or <= or >= or == or !=	`<=`, `!=`
id	letter followed by letters and digits	`pi`, `score`, `D2`
number	any numeric constant	`3.14159`, `0`, `6.02e23`
literal	anything but ", surrounded by "'s	`"core dumped"`

Examples of tokens

Program for Lexical Analyzer in C

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>

int isKeyword(char buffer[]){
char keywords[32][10] = {"auto","break","case","char","const","continue","default","do","double","else","enum","extern",
"float","for","goto","if","int","long","register","return",
"sort","signed","sizeof","static","struct","switch","typedef","union","unsigned","void","volatile",
"while"};
int i, flag = 0;
for(i = 0; i < 32; ++i){
if(strcmp(keywords[i], buffer) == 0){
flag = 1;
break;
}
}
return flag;
}

int main(){
char ch, buffer[15], operators[] = "+-*/%=";
FILE *fp;
```

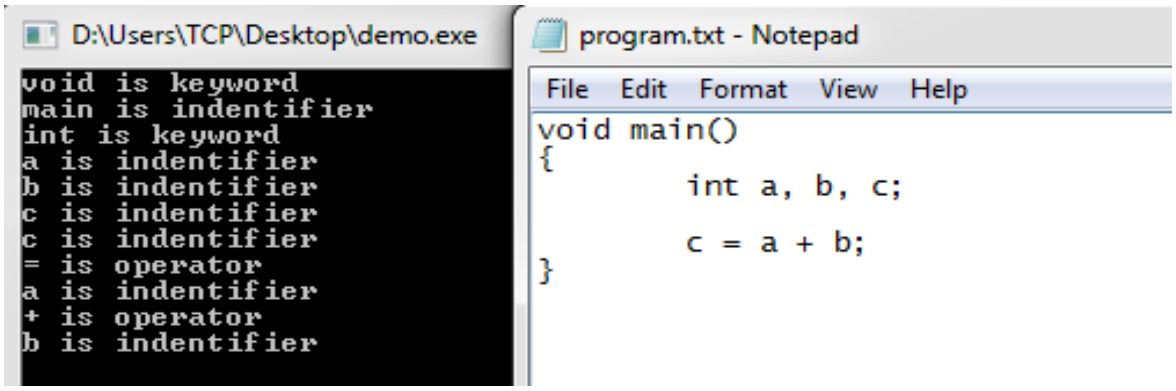
```
int i,j=0;
fp = fopen("program.txt","r");
if(fp == NULL){
printf("error while opening the file\n");
exit(0);
}
while((ch = fgetc(fp)) != EOF){
   for(i = 0; i < 6; ++i){
   if(ch == operators[i])
   printf("%c is operator\n", ch);
   }

   if(isalnum(ch)){
   buffer[j++] = ch;
   }
   else if((ch == ' ' || ch == '\n') && (j != 0)){
   buffer[j] = '\0';
   j = 0;

   if(isKeyword(buffer) == 1)
   printf("%s is keyword\n", buffer);
   else
   printf("%s is indentifier\n", buffer);
   }

}
fclose(fp);
return 0;
}
```

Output:



Conclusion:

Thus, we have successfully implemented lexical analyzer to produce tokens.

G.K. Gujar Memorial Charitable Trust's **DR.ASHOK GUJAR TECHNICAL INSTITUTE'S** **DR. DAULATRAO AHER COLLEGE OF ENGINEERING, KARAD**	
Lab Manual	
Subject : Compiler Construction	
Expt. No. 3	Program to create a symbol table generator.

Aim: Program to create a symbol table generator.

Theory:

Definition
The symbol table is defined as the set of Name and Value pairs.

Symbol Table is an important data structure created and maintained by the compiler in order to keep track of semantics of variables i.e. it stores information about the scope and binding information about names, information about instances of various entities such as variable and function names, classes, objects, etc.

- It is built-in lexical and syntax analysis phases.
- The information is collected by the analysis phases of the compiler and is used by the synthesis phases of the compiler to generate code.
- It is used by the compiler to achieve compile-time efficiency.
- It is used by various phases of the compiler as follows:-
  1. **Lexical Analysis:** Creates new table entries in the table, for example like entries about tokens.
  2. **Syntax Analysis:** Adds information regarding attribute type, scope, dimension, line of reference, use, etc in the table.
  3. **Semantic Analysis:** Uses available information in the table to check for semantics i.e. to verify that expressions and assignments are semantically correct(type checking) and update it accordingly.
  4. **Intermediate Code generation:** Refers symbol table for knowing how much and what type of run-time is allocated and table helps in adding temporary variable information.
  5. **Code Optimization:** Uses information present in the symbol table for machine-dependent optimization.
  6. **Target Code generation:** Generates code by using address information of identifier present in the table.

Symbol Table entries – Each entry in the symbol table is associated with attributes that support the compiler in different phases.

Use of Symbol Table-

The symbol tables are typically used in compilers. Basically compiler is a program which scans the application program (for instance: your C program) and produces

machine code.

During this scan compiler stores the identifiers of that application program in the symbol table. These identifiers are stored in the form of name, value address, type.

Here the name represents the name of identifier, value represents the value stored in an identifier, the address represents memory location of that identifier and type represents the data type of identifier.

Thus compiler can keep track of all the identifiers with all the necessary information.

Items stored in Symbol table:

- Variable names and constants
- Procedure and function names
- Literal constants and strings
- Compiler generated temporaries
- Labels in source languages

Information used by the compiler from Symbol table:

- Data type and name
- Declaring procedures
- Offset in storage
- If structure or record then, a pointer to structure table.
- For parameters, whether parameter passing by value or by reference
- Number and type of arguments passed to function
- Base Address

Operations of Symbol table – The basic operations defined on a symbol table include:

Operation	Function
allocate	to allocate a new empty symbol table
free	to remove all entries and free storage of symbol table
lookup	to search for a name and return pointer to its entry
insert	to insert a name in a symbol table and return a pointer to its entry
set_attribute	to associate an attribute with a given entry
get_attribute	to get an attribute associated with a given entry

Operations on Symbol Table :

Following operations can be performed on symbol table-

1. Insertion of an item in the symbol table.

2. Deletion of any item from the symbol table.

3. Searching of desired item from symbol table.

Implementation of Symbol table –

Following are commonly used data structures for implementing symbol table:-

1. **List –**

we use a single array or equivalently several arrays, to store names and their associated information ,New names are added to the list in the order in which they are encountered . The position of the end of the array is marked by the pointer available, pointing to where the next symbol-table entry will go. The search for a name proceeds backwards from the end of the array to the beginning. when the name is located the associated information can be found in the words following next.

id1	info1	id2	info2	……..	id_n	info_n

- In this method, an array is used to store names and associated information.
- A pointer **"available"** is maintained at end of all stored records and new names are added in the order as they arrive
- To search for a name we start from the beginning of the list till available pointer and if not found we get an error **"use of the undeclared name"**
- While inserting a new name we must ensure that it is not already present otherwise an error occurs i.e. **"Multiple defined names"**
- Insertion is fast O(1), but lookup is slow for large tables – O(n) on average
- The advantage is that it takes a minimum amount of space.

1. **Linked List –**
   - This implementation is using a linked list. A link field is added to each record.
   - Searching of names is done in order pointed by the link of the link field.
   - A pointer **"First"** is maintained to point to the first record of the symbol table.
   - Insertion is fast O(1), but lookup is slow for large tables – O(n) on average
2. **Hash Table –**
   - In hashing scheme, two tables are maintained – a hash table and symbol table and are the most commonly used method to implement symbol tables.
   - A hash table is an array with an index range: 0 to table size – 1. These entries are pointers pointing to the names of the symbol table.
   - To search for a name we use a hash function that will result in an integer between 0 to table size – 1.
   - Insertion and lookup can be made very fast – O(1).
   - The advantage is quick to search is possible and the disadvantage is that hashing is complicated to implement.
3. **Binary Search Tree –**
   - Another approach to implementing a symbol table is to use a binary search tree i.e. we add two link fields i.e. left and right child.
   - All names are created as child of the root node that always follows the property of the binary search tree.
   - Insertion and lookup are $O(\log_2 n)$ on average.

Advantages of Symbol Table

1. The efficiency of a program can be increased by using symbol tables, which give

quick and simple access to crucial data such as variable and function names, data kinds, and memory locations.

2. better coding structure Symbol tables can be used to organize and simplify code, making it simpler to comprehend, discover, and correct problems.
3. Faster code execution: By offering quick access to information like memory addresses, symbol tables can be utilized to optimize code execution by lowering the number of memory accesses required during execution.
4. Symbol tables can be used to increase the portability of code by offering a standardized method of storing and retrieving data, which can make it simpler to migrate code between other systems or programming languages.
5. Improved code reuse: By offering a standardized method of storing and accessing information, symbol tables can be utilized to increase the reuse of code across multiple projects.
6. Symbol tables can be used to facilitate easy access to and examination of a program's state during execution, enhancing debugging by making it simpler to identify and correct mistakes.

Disadvantages of Symbol Table

1. **Increased memory consumption:** Systems with low memory resources may suffer from symbol tables' high memory requirements.
2. **Increased processing time:** The creation and processing of symbol tables can take a long time, which can be problematic in systems with constrained processing power.
3. **Complexity:** Developers who are not familiar with compiler design may find symbol tables difficult to construct and maintain.
4. **Limited scalability:** Symbol tables may not be appropriate for large-scale projects or applications that require o the management of enormous amounts of data due to their limited scalability.
5. **Upkeep:** Maintaining and updating symbol tables on a regular basis can be time- and resource-consuming.
6. **Limited functionality:** It's possible that symbol tables don't offer all the features a developer needs, and therefore more tools or libraries will be needed to round out their capabilities.

Applications of Symbol Table

1. **Resolution of variable and function names:** Symbol tables are used to identify the data types and memory locations of variables and functions as well as to resolve their names.
2. **Resolution of scope issues:** To resolve naming conflicts and ascertain the range of variables and functions, symbol tables are utilized.
3. Symbol tables, which offer quick access to information such as memory locations, are used to optimize code execution.
4. **Code generation:** By giving details like memory locations and data kinds, symbol tables are utilized to create machine code from source code.
5. **Error checking and code debugging:** By supplying details about the status of a program during execution, symbol tables are used to check for faults and debug code.
6. **Code organization and documentation:** By supplying details about a program's

structure, symbol tables can be used to organize code and make it simpler to understand.

Program code:

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<string.h>
#include<stdlib.h>
#define NULL 0
int size=0;
void Insert();
void Display();
void Delete();
int Search(char lab[]);
void Modify();
struct SymbTab
{
 char label[10],symbol[10];
 int addr;
struct SymbTab *next;};
struct SymbTab *first,*last;
void main()
{
 int op,y;
 char la[10];
 clrscr();
 do
 {
 printf("\n\tSYMBOL TABLE IMPLEMENTATION\n");
 printf("\n\t1.INSERT\n\t2.DISPLAY\n\t3.DELETE\n\t4.SEARCH\n\t5.MODIFY\n\t6.END\n");
 printf("\n\tEnter your option : ");
 scanf("%d",&op);
 switch(op)
 {
```

```
                    case 1:
                        Insert();
                        break;
                    case 2:
                        Display();
                        break;
                    case 3:
                        Delete();
                        break;
                    case 4:
        printf("\n\tEnter the label to be searched : ");
                        scanf("%s",la);
                        y=Search(la);
                printf("\n\tSearch Result:");
                        if(y==1)
        printf("\n\tThe label is present in the symbol table\n");
                        else
        printf("\n\tThe label is not present in the symbol table\n");
                        break;
                    case 5:
                        Modify();
                        break;
                    case 6:
                        exit(0);
                        }
 }while(op<6);
 getch();
}
void Insert()
{
                    int n;
                   char l[10];
            printf("\n\tEnter the label : ");
                        scanf("%s",l);
                        n=Search(l);
```

```
                                                    if(n==1)
    printf("\n\tThe label exists already in the symbol table\n\tDuplicate can't be inserted");
                                                    else
                                                    {
                        struct                  SymbTab                 *p;
                            p=malloc(sizeof(struct                      SymbTab));
                                                    strcpy(p->label,l);
                printf("\n\tEnter       the       symbol      :       ");
                                                    scanf("%s",p->symbol);
                printf("\n\tEnter       the       address     :       ");
                                                    scanf("%d",&p->addr);
                                                    p->next=NULL;
                                                    if(size==0)
                                                    {
                                                    first=p;
                                                    last=p;
                                                    }
                                                    else
                                                    {
                                                    last->next=p;
                                                    last=p;
                                                    }
                                                    size++;
                                                    }
                            printf("\n\tLabel                       inserted\n");
}
void                                                Display()
{
                                int                 i;
                        struct                  SymbTab                 *p;
                                                    p=first;
                                printf("\n\tLABEL\t\tSYMBOL\t\tADDRESS\n");
                                                    for(i=0;i<size;i++)
                                                    {
                            printf("\t%s\t\t%s\t\t%d\n",p->label,p->symbol,p->addr);
```

```
                                                    p=p->next;
                                                    }
}
int                         Search(char                         lab[])
{
 int                                                    i,flag=0;
 struct                     SymbTab                          *p;
 p=first;
                                                    for(i=0;i<size;i++)
                                                    {
                                                if(strcmp(p->label,lab)==0)
                                                    flag=1;
                                                    p=p->next;
                                                    }
 return                                                 flag;
}
void                                                   Modify()
{
                            char                          l[10],nl[10];
                            int                          add,choice,i,s;
                    struct                SymbTab                *p;
                                                    p=first;
          printf("\n\tWhat      do      you      want      to      modify?\n");
   printf("\n\t1.Only the label\n\t2.Only the address\n\t3.Both the label and address\n");
                printf("\tEnter      your      choice      :      ");
                                                scanf("%d",&choice);
                                                switch(choice)
                                                {
                                    case                          1:
                printf("\n\tEnter      the      old      label      :      ");
                                                scanf("%s",l);
                                                s=Search(l);
                                                if(s==0)
                        printf("\n\tLabel          not          found\n");
                                                else
```

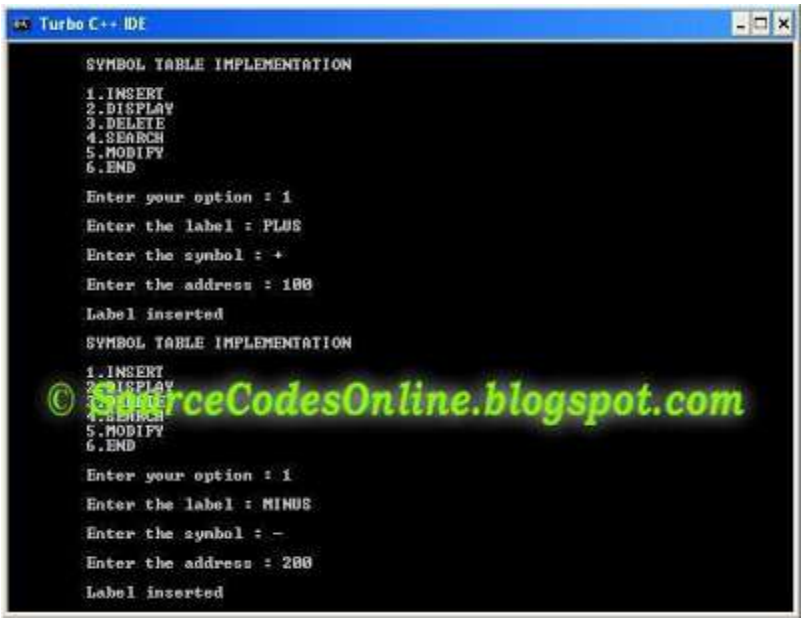
```
{
printf("\n\tEnter the new label : ");
scanf("%s",nl);
for(i=0;i<size;i++)
{
if(strcmp(p->label,l)==0)
strcpy(p->label,nl);
p=p->next;
}
printf("\n\tAfter Modification:\n");
Display();
}
break;
case 2:
printf("\n\tEnter the label where the address is to be modified : ");
scanf("%s",l);
s=Search(l);
if(s==0)
printf("\n\tLabel not found\n");
else
{
printf("\n\tEnter the new address : ");
scanf("%d",&add);
for(i=0;i<size;i++)
{
if(strcmp(p->label,l)==0)
p->addr=add;
p=p->next;
}
printf("\n\tAfter Modification:\n");
Display();
}
break;
case 3:
printf("\n\tEnter the old label : ");
```

```
                                        scanf("%s",l);
                                        s=Search(l);
                                        if(s==0)
                        printf("\n\tLabel not found\n");
                                        else
                                        {
        printf("\n\tEnter the new label : ");
                                        scanf("%s",nl);
        printf("\n\tEnter the new address : ");
                                        scanf("%d",&add);
                                        for(i=0;i<size;i++)
                                        {
                                        if(strcmp(p->label,l)==0)
                                        {
                                        strcpy(p->label,nl);
                                        p->addr=add;
                                        }
                                        p=p->next;
                                        }
                        printf("\n\tAfter Modification:\n");
                                        Display();
                                        }
                                        break;
                                        }
}
void Delete()
{
                                int a;
                                char l[10];
                        struct SymbTab *p,*q;
                                        p=first;
        printf("\n\tEnter the label to be deleted : ");
                                        scanf("%s",l);
                                        a=Search(l);
                                        if(a==0)
```

```
                printf("\n\tLabel not found\n");
                else
                {
                if(strcmp(first->label,l)==0)
                first=first->next;
            else if(strcmp(last->label,l)==0)
                {
                q=p->next;
                while(strcmp(q->label,l)!=0)
                {
                p=p->next;
                q=q->next;
                }
                p->next=NULL;
                last=p;
                }
                else
                {
                q=p->next;
                while(strcmp(q->label,l)!=0)
                {
                p=p->next;
                q=q->next;
                }
                p->next=q->next;
                }
                size--;
            printf("\n\tAfter Deletion:\n");
                Display();
                }
}
```

OUTPUT:

INSERTION:

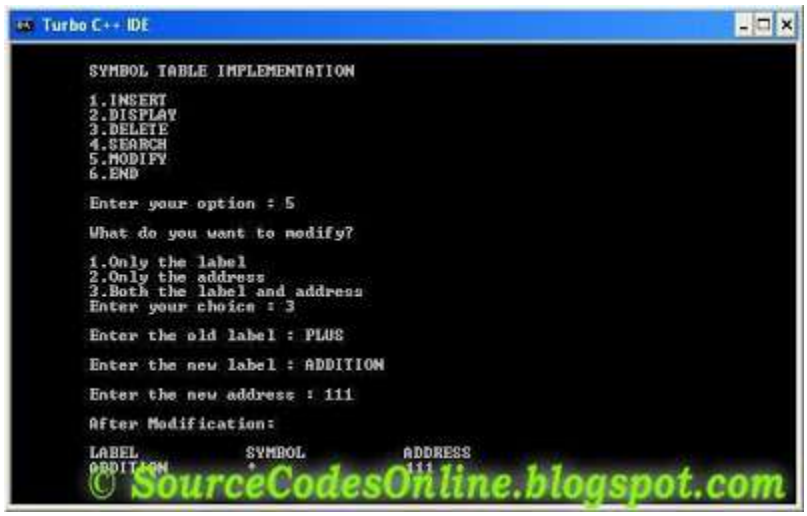


DISPLAY:

DELETION:

SEARCH:

MODIFICATION:



Conclusion:

Thus, we have successfully implemented to generate symbol table.

G.K. Gujar Memorial Charitable Trust's **DR.ASHOK GUJAR TECHNICAL INSTITUTE'S** **DR. DAULATRAO AHER COLLEGE OF ENGINEERING, KARAD**	
Lab Manual	
Subject : Compiler Construction	
Expt. No. 4	Program to implement recursive descent parsing method for simple expression.

Aim: Program to implement recursive descent parsing method for simple expression.

Theory:

RECURSIVE DESCENT PARSING

- Recursive descent parsing is one of the top-down parsing techniques that uses a set of recursive procedures to scan its input**.**
- This parsing method may involve **backtracking**, that is, making repeated scans of the input.

Example for backtracking :

Consider the grammar G : S → cAd

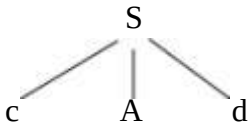
A → ab | a

and the input string w=cad.

The parse tree can be constructed using the following top-down approach :

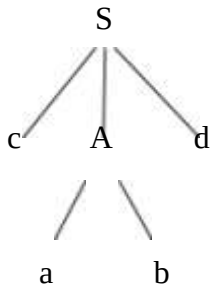
Step1:

Initially create a tree with single node labeled S. An input pointer points to 'c', the first symbol of w. Expand the tree with the production of S.



Step2:

The leftmost leaf ‘c’ matches the first symbol of w, so advance the input pointer to the second symbol of w ‘a’ and consider the next leaf ‘A’. Expand A using the first alternative.



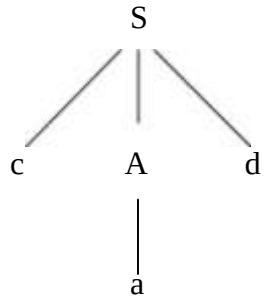
Step3:

The second symbol ‘a’ of w also matches with second leaf of tree. So advance the input pointer to third symbol of w ‘d’**.** But the third leaf of tree is b which does not match with the input symbol **d.**

Hence discard the chosen production and reset the pointer to second position. This is called **backtracking.**

Step4:

Now try the second alternative for A.



Now we can halt and announce the successful completion of parsing

Grammer with Left Recursion

E ->E+i | i

After elimination of Left Recursion

E ->iE'
E' -> +iE' | epsilon

Input i+i*i;

To implement Recursive Descent Parser > C Program

Program:

```
#include"stdio.h"
#include"conio.h"
#include"string.h"
#include"stdlib.h"
#include"ctype.h"

char ip_sym[15],ip_ptr=0,op[50],tmp[50];
void e_prime();
void e();
void t_prime();
void t();
void f();
void advance();
int n=0;
void e()
{
 strcpy(op,"TE'");
 printf("E=%-25s",op);
 printf("E->TE'\n");
 t();
 e_prime();
}

void e_prime()
{
int i,n=0,l;
for(i=0;i<=strlen(op);i++)
    if(op[i]!='e')
 tmp[n++]=op[i];
strcpy(op,tmp);
l=strlen(op);
for(n=0;n < l && op[n]!='E';n++);
if(ip_sym[ip_ptr]=='+')
 {
     i=n+2;
 do
 {
 op[i+2]=op[i];
 i++;
 }while(i<=l);
  op[n++]='+';
  op[n++]='T';
```

```
 op[n++]='E';
 op[n++]=39;
 printf("E=%-25s",op);
 printf("E'->+TE'\n");
 advance();
 t();
 e_prime();
 }
 else
 {
    op[n]='e';
 for(i=n+1;i<=strlen(op);i++)
 op[i]=op[i+1];
 printf("E=%-25s",op);
 printf("E'->e");
 }
}
void t()
{
 int i,n=0,l;
 for(i=0;i<=strlen(op);i++)
  if(op[i]!='e')
   tmp[n++]=op[i];
 strcpy(op,tmp);
 l=strlen(op);
for(n=0;n < l && op[n]!='T';n++);

 i=n+1;
 do
 {
  op[i+2]=op[i];
  i++;
 }while(i < l);
 op[n++]='F';
 op[n++]='T';
 op[n++]=39;
 printf("E=%-25s",op);
 printf("T->FT'\n");
 f();
 t_prime();
}

void t_prime()
{
int i,n=0,l;
for(i=0;i<=strlen(op);i++)
```

```
   if(op[i]!='e')
 tmp[n++]=op[i];
strcpy(op,tmp);
l=strlen(op);
for(n=0;n < l && op[n]!='T';n++);
if(ip_sym[ip_ptr]=='*')
 {
    i=n+2;
 do
 {
 op[i+2]=op[i];
 i++;
 }while(i < l);
  op[n++]='*';
  op[n++]='F';
  op[n++]='T';
  op[n++]=39;
  printf("E=%-25s",op);
  printf("T'->*FT'\n");
  advance();
  f();
  t_prime();
 }
 else
 {
   op[n]='e';
  for(i=n+1;i<=strlen(op);i++)
 op[i]=op[i+1];
 printf("E=%-25s",op);
 printf("T'->e\n");
 }
}

void f()
{
int i,n=0,l;
for(i=0;i<=strlen(op);i++)
   if(op[i]!='e')
 tmp[n++]=op[i];
 strcpy(op,tmp);
l=strlen(op);
for(n=0;n < l && op[n]!='F';n++);
 if((ip_sym[ip_ptr]=='i')||(ip_sym[ip_ptr]=='I'))
 {
 op[n]='i';
 printf("E=%-25s",op);
```

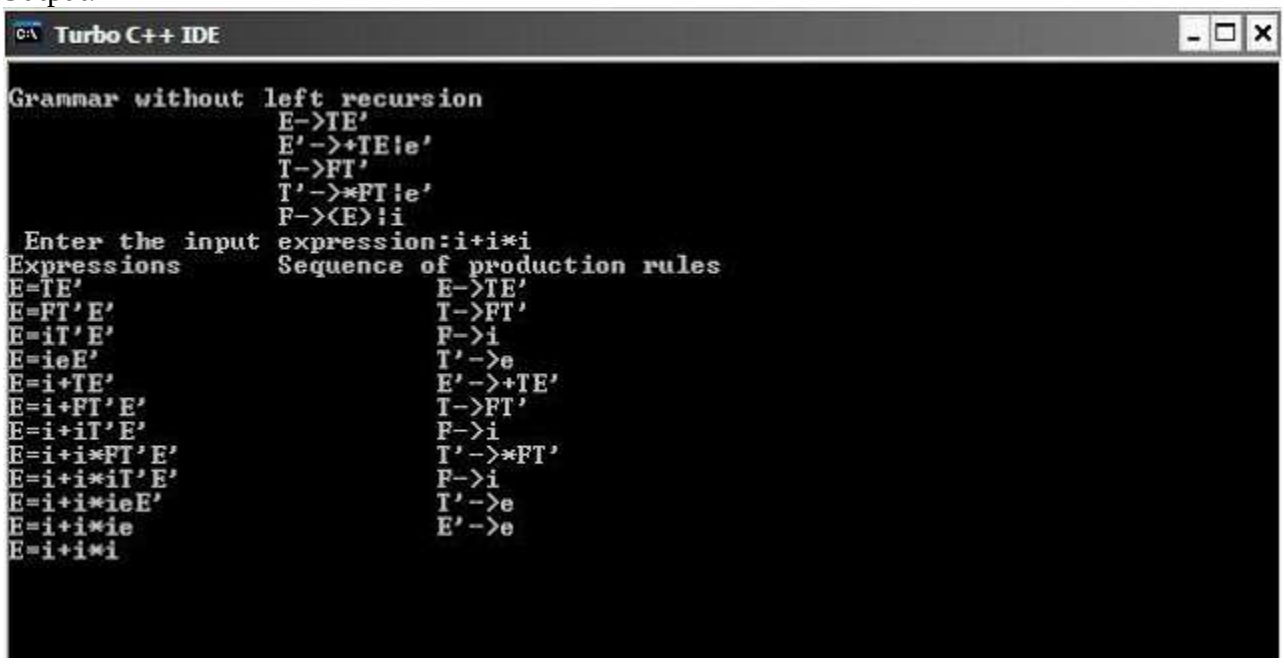
```
 printf("F->i\n");
 advance();
 }
 else
 {
  if(ip_sym[ip_ptr]=='(')
  {
   advance();
   e();
   if(ip_sym[ip_ptr]==')')
   {
    advance();
     i=n+2;
 do
 {
 op[i+2]=op[i];
 i++;
 }while(i<=l);
  op[n++]='(';
  op[n++]='E';
  op[n++]=')';
  printf("E=%-25s",op);
  printf("F->(E)\n");
   }
  }
  else
  {
   printf("\n\t syntax error");
   getch();
   exit(1);
  }
 }
}

void advance()
{
 ip_ptr++;
}

void main()
{
 int i;
 clrscr();
 printf("\nGrammar without left recursion");
 printf("\n\t\t E->TE' \n\t\t E'->+TE'|e \n\t\t T->FT' ");
 printf("\n\t\t T'->*FT'|e \n\t\t F->(E)|i");
```

```
 printf("\n Enter the input expression:");
 gets(ip_sym);
 printf("Expressions");
 printf("\t Sequence of production rules\n");
  e();
  for(i=0;i < strlen(ip_sym);i++)
 {
  if(ip_sym[i]!='+'&&ip_sym[i]!='*'&&ip_sym[i]!='('&&
     ip_sym[i]!=')'&&ip_sym[i]!='i'&&ip_sym[i]!='I')
  {
   printf("\nSyntax error");
   break;
  }
  for(i=0;i<=strlen(op);i++)
    if(op[i]!='e')
 tmp[n++]=op[i];
    strcpy(op,tmp);
    printf("\nE=%-25s",op);
 }
 getch();
}
```

Output:



Conclusion:

Thus we have successfully parsed the input i+i*i by using recursive descent method.

G.K. Gujar Memorial Charitable Trust's **DR.ASHOK GUJAR TECHNICAL INSTITUTE'S** **DR. DAULATRAO AHER COLLEGE OF ENGINEERING, KARAD**	
Lab Manual	
Subject : Compiler Construction	
Expt. No. 5	Program to create a syntax tree for simple expression techniques.

Aim: Program to create a syntax tree for simple expression techniques.

Theory:

A syntax tree depicts the natural hierarchical structure of a source program. A **dag (Directed Acyclic Graph)** gives the same information but in a more compact way because common sub expressions are identified. A syntax tree and dag for the assignment statement **a : = b * - c + b * - c** are as follows:

assign

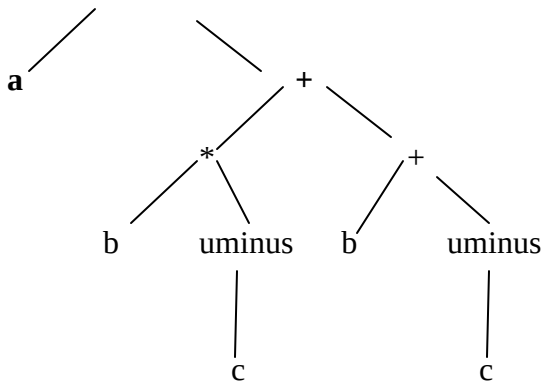


Figure: Syntax tree

```
#include<stdio.h>
#include<stdlib.h>
#define Blank ' '
#define Tab '\t'
#define MAX 100

long int pop ();
char expression[MAX], syntaxtree[MAX];
long int stack[MAX];
int top;

void main()
{
long int value;
char choice='y';
```

```
clrscr();
while(choice == 'y')
{
top = 0;
printf("Enter Expression : ");
fflush(stdin);
gets(expression);
expression_to_syntaxtree();
printf("Expression Tree Format : %s\n",syntaxtree);
printf("Want to continue(y/n) : ");
scanf("%c",&choice);
}
}

expression_to_syntaxtree()
{
int i,p=0,type,precedence,len;
char next ;
stack[top]='#';
len=strlen(expression);
expression[len]='#';
for(i=0; expression[i]!='#';i++)
{
if( !white_space(expression[i]))
{
switch(expression[i])
{
case '(':
push(expression[i]);
break;

case ')':
while((next = pop()) != '(')
syntaxtree[p++] = next;
break;
case '+':
case '-':
case '*':
case '/':
case '%':
case '^':
precedence = prec(expression[i]);
while(stack[top]!='#' && precedence<= prec(stack[top]))
syntaxtree[p++] = pop();
push(expression[i]);
break;
1.  default: /*if an operand comes */
syntaxtree[p++] = expression[i];
}/*End of switch */
}/*End of if */
}/*End of for */
while(stack[top]!='#')
syntaxtree[p++] = pop();
syntaxtree[p] = '\0' ;  /*End syntaxtree with'\0' to make it a string*/
}
```

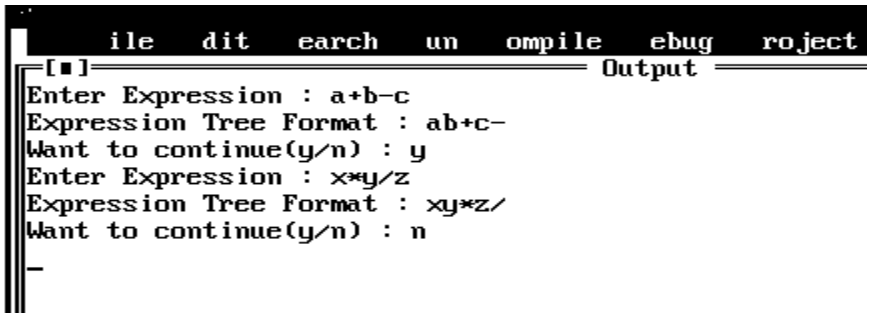
```
prec(char symbol )
{
switch(symbol)
{
case '(':
return 0;
case '+':
case '-':
return 1;
case '*':
case '/':
case '%':
return 2;
case '^':
return 3;
}
}

push(long int symbol)
{
if(top > MAX)
{
printf("Stack overflow\n");
exit(1);
}
else
{
top=top+1;
stack[top] = symbol;
}
}

long int pop()
{
if (top == -1 )
{
printf("Stack underflow \n");
exit(2);
}
else
return (stack[top--]);
}

white_space(char symbol)
{
if( symbol == Blank || symbol == Tab || symbol == '\0')
return 1;
else
return 0;
```

Output:



Conclusion:

Thus we have successfully generated syntax tree.

	G.K. Gujar Memorial Charitable Trust's **DR.ASHOK GUJAR TECHNICAL INSTITUTE'S** **DR. DAULATRAO AHER COLLEGE OF ENGINEERING, KARAD**
	Lab Manual
Subject : Compiler Construction	
Expt. No. 6	Implement intermediate code generator for the Boolean expression in three address code format.

Aim: Implement intermediate code generator for the Boolean expression in three address code format.

Theory:

Three-Address Code:

Three-address code is a sequence of statements of the general form x : = y *op* z where x, y and z are names, constants, or compiler-generated temporaries; *op* stands for any operator, such as a fixed- or floating-point arithmetic operator, or a logical operator on Boolean valued data. Thus a source language expression like x+ y*z might be translated into a sequence

t1 : = y * z
t2 : = x + t1

where t1 and t2 are compiler-generated temporary names.

A three-address statement is an abstract form of intermediate code. In a compiler, these statements can be implemented as records with fields for the operator and the operands. Three such representations are:

Quadruples:

- ✓ A quadruple is a record structure with four fields, which are, ***op, arg1, arg2*** and ***result.***
- ✓ The *op* field contains an internal code for the operator. The three-address statement **x : = y op z** is represented by placing *y* in *arg1*, *z* in *arg2* and *x* in *result.*
- ✓ The contents of field's arg1, arg2 and result are normally pointers to the symbol-table entries for the names represented by these fields. If so, temporary names must be entered into the symbol table as they are created.

Triples:

- ✓ To avoid entering temporary names into the symbol table, we might refer to a temporary value by the position of the statement that computes it.
- ✓ If we do so, three-address statements can be represented by records with only three fields: *op, arg1* and *arg2.*
- ✓ The fields *arg1* and *arg2*, for the arguments of *op*, are either pointers to the symbol table or pointers into the triple structure (for temporary values).
- ✓ Since three fields are used, this intermediate code format is known as *triples.*

Indirect Triples:

- ✓ Another implementation of three-address code is that of listing pointers to triples, rather than listing the triples themselves. This implementation is called indirect triples.

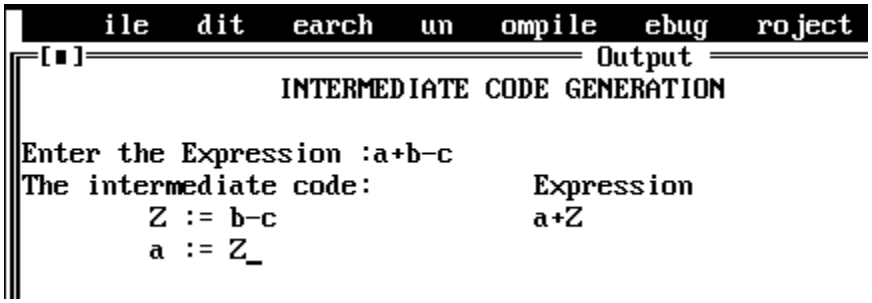
```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>

int i=1,j=0,no=0,tmpch=90;
char str[100],left[15],right[15];
void findopr();
void explore();
void fleft(int);
void fright(int);
struct exp
{
        int pos;
        char op;
}k[15];
void main()
{
        clrscr();
        printf("\t\tINTERMEDIATE CODE GENERATION\n\n");
        printf("Enter the Expression :");
        scanf("%s",str);
        printf("The intermediate code:\t\tExpression\n");
        findopr();
        explore();
        getch();
}
void findopr()
{
        for(i=0;str[i]!='\0';i++)
                if(str[i]==':')
                {
                k[j].pos=i;
                k[j++].op=':';
                }
        for(i=0;str[i]!='\0';i++)
                if(str[i]=='/')
                {
                k[j].pos=i;
                k[j++].op='/';
```

```
		}
	for(i=0;str[i]!='\0';i++)
		if(str[i]=='*')
		{
		k[j].pos=i;
		k[j++].op='*';
		}
	for(i=0;str[i]!='\0';i++)
		if(str[i]=='+')
		{
		k[j].pos=i;
		k[j++].op='+';
		}
	for(i=0;str[i]!='\0';i++)
		if(str[i]=='-')
		{
		k[j].pos=i;
		k[j++].op='-';
		}
}
void explore()
{
	i=1;
	while(k[i].op!='\0')
	{
		fleft(k[i].pos);
		fright(k[i].pos);
		str[k[i].pos]=tmpch--;
		printf("\t%c := %s%c%s\t\t",str[k[i].pos],left,k[i].op,right);
		for(j=0;j<strlen(str);j++)
			if(str[j]!='$')
				printf("%c",str[j]);
		printf("\n");
		i++;
	}
	fright(-1);
	if(no==0)
	{
		fleft(strlen(str));
		printf("\t%s := %s",right,left);
```

```
                getch();
                exit(0);
        }
        printf("\t%s :=  %c",right,str[k[--i].pos]);
        getch();
}
void fleft(int x)
{
        int w=0,flag=0;
        x--;
        while(x!= -1 &&str[x]!= '+' &&str[x]!='*'&&str[x]!='='&&str[x]!='\0'&&str[x]!='-
'&&str[x]!='/'&&str[x]!=':')
        {
                if(str[x]!='$'&& flag==0)
                {
                left[w++]=str[x];
                left[w]='\0';
                str[x]='$';
                flag=1;
                }
                x--;
        }
}
void fright(int x)
{
        int w=0,flag=0;
        x++;
        while(x!=                -1                &&                str[x]!=
'+'&&str[x]!='*'&&str[x]!='\0'&&str[x]!='='&&str[x]!=':'&&str[x]!='-'&&str[x]!='/')
        {
                if(str[x]!='$'&& flag==0)
                {
                right[w++]=str[x];
                right[w]='\0';
                str[x]='$';
                flag=1;
                }
                x++;
        }
}
```

Output:



Conclusion:

Thus we have successfully generated intermediate code for expression a+b-c.

G.K. Gujar Memorial Charitable Trust's **DR.ASHOK GUJAR TECHNICAL INSTITUTE'S** **DR. DAULATRAO AHER COLLEGE OF ENGINEERING, KARAD**	
Lab Manual	
Subject : Compiler Construction	
Expt. No. 7	Program to implement FIRST and FOLLOW for given grammar.

Aim: Program to implement FIRST and FOLLOW for given grammar.

Theory:

FIRST and FOLLOW are two functions associated with grammar that help us fill in the entries of an M-table.

FIRST ()− It is a function that gives the set of terminals that begin the strings derived from the production rule.

A symbol c is in FIRST (α) if and only if $\alpha \Rightarrow c\beta$ for some sequence β of grammar symbols.

A terminal symbol a is in FOLLOW (N) if and only if there is a derivation from the start symbol S of the grammar such that $S \Rightarrow \alpha N a \beta$, where α and β are a (possible empty) sequence of grammar symbols. In other words, a terminal c is in FOLLOW (N) if c can follow N at some point in a derivation.

Benefit of FIRST ( ) and FOLLOW ( )

- It can be used to prove the LL (K) characteristic of grammar.
- It can be used to promote in the construction of predictive parsing tables.
- It provides selection information for recursive descent parsers.

Computation of FIRST

FIRST (α) is defined as the collection of terminal symbols which are the first letters of strings derived from α.

FIRST $(\alpha) = \{\alpha \mid \alpha \rightarrow^* \alpha\beta$ for some string $\beta\}$

If X is Grammar Symbol, then First (X) will be −

- If X is a terminal symbol, then FIRST(X) = {X}
- If $X \rightarrow \varepsilon$, then FIRST(X) = {ε}

- If X is non-terminal & $X \rightarrow a\ \alpha$, then FIRST (X) = {a}
- If $X \rightarrow Y_1, Y_2, Y_3$, then FIRST (X) will be

(a) If Y is terminal, then

FIRST (X) = FIRST (Y_1, Y_2, Y_3) = {Y_1}

(b) If Y_1 is Non-terminal and

If Y_1 does not derive to an empty string i.e., If FIRST (Y_1) does not contain ε then, FIRST (X) = FIRST (Y_1, Y_2, Y_3) = FIRST(Y_1)

(c) If FIRST (Y_1) contains ε, then.

FIRST (X) = FIRST (Y_1, Y_2, Y_3) = FIRST(Y_1) − {ε} ∪ FIRST(Y_2, Y_3)

Similarly, FIRST (Y_2, Y_3) = {Y_2}, If Y_2 is terminal otherwise if Y_2 is Non-terminal then

- FIRST (Y_2, Y_3) = FIRST (Y_2), if FIRST (Y_2) does not contain ε.
- If FIRST (Y_2) contain ε, then
- FIRST (Y_2, Y_3) = FIRST (Y_2) − {ε} ∪ FIRST (Y_3)

Similarly, this method will be repeated for further Grammar symbols, i.e., for $Y_4, Y_5, Y_6 \ldots . Y_K$.

Computation of FOLLOW

Follow (A) is defined as the collection of terminal symbols that occur directly to the right of A.

FOLLOW(A) = {a|S ⇒* αAaβ where α, β can be any strings}

Rules to find FOLLOW

- If S is the start symbol, FOLLOW (S) ={$}
- If production is of form $A \rightarrow \alpha\ B\ \beta, \beta \neq \varepsilon$.

(a) If FIRST (β) does not contain ε then, FOLLOW (B) = {FIRST (β)}

Or

(b) If FIRST (β) contains ε (i. e. , β ⇒* ε), then

FOLLOW (B) = FIRST (β) − {ε} ∪ FOLLOW (A)

∵ when β derives ε, then terminal after A will follow B.

- If production is of form A → αB, then Follow (B) ={FOLLOW (A)}.

Example:

Consider the following grammar:

E → E+T | T
T → T*F | F
F → (E) | id

After eliminating left-recursion the grammar is

E → TE'
E' → +TE' | ε
T → FT'
T' → *FT' | ε
F → (E) | id

First( ) :

FIRST(E) = { (, id}
FIRST(E') ={+ , ε }
FIRST(T) = { (, id}
FIRST(T') = {*, ε }
FIRST(F) = { (, id }

Follow( ):

FOLLOW(E) = { $,) }
FOLLOW(E') = { $,) }
FOLLOW(T) = { +, $,) }
FOLLOW(T') = { +, $,) }
FOLLOW(F) = {+, * , $,) }

Program:

```
import java.io.*;
class fafp2
{
public static void main(String args[])throws IOException
{
int i,j,k,tempi=0;
FileReader f=new FileReader("faf.txt");
BufferedReader in=new BufferedReader(f);
String nonter[]=new String[5];
```

```
String ter[]=new String[6];
String ip[][]=new String[5][3];
String firsts[]=new String[6];
String follows[]=new String[6];
String lineget,str,str2,str3="",temp="",tocal="",sec="",foltemp="$";
nonter[0]="E";
nonter[1]="N";
nonter[2]="T";
nonter[3]="M";
nonter[4]="F";
ter[0]="e";
ter[1]="+";
ter[2]="(";
ter[3]=")";
ter[4]="*";
ter[5]="d";
System.out.println("ASSUMING E'=N,T'=M,id=d,epsilon=e for coding
purposes.");
System.out.println();
System.out.println();
//first
System.out.println("GIVEN EXPRESSION");
for(i=0;i<5;i++)
{
lineget=in.readLine();
System.out.println(lineget);
str=lineget.substring(0,1);
str2=lineget.substring(2);
ip[i][0]=str;
ip[i][1]=str2;
if(lineget.contains("/"))
{
for(j=0;j<lineget.length();j++)
{
if(lineget.charAt(j)=='/')
{
str3=lineget.substring(j+1);
ip[i][2]=str3;
}
}
}
}
for(j=0;j<5;j++)
{
for(k=0;k<6;k++)
{
```

```
if(ip[j][1].startsWith(ter[k]))
{
temp=ip[j][1].substring(0,1);
firsts[j]=temp+ip[j][2];
}
}
}
for(i=0;i<5;i++)
{
for(j=0;j<5;j++)
{
if(firsts[j]==null)
{
tocal=ip[j][1].substring(0,1);
for(k=0;k<5;k++)
{
temp=nonter[k];
if(tocal.equals(ip[k][0]))
{
tempi=k;
break;
}
} if(firsts[tempi]!=null)
{
firsts[j]=firsts[tempi];
}
}
}
} //follow
for(i=0;i<5;i++)
{
tocal=nonter[i];
for(j=0;j<5;j++)
{
sec=ip[j][1].substring(1,2);
if(sec.equals(tocal))
{
if(ip[j][1].length()>2)
{
temp=ip[j][1].substring(2,3);
for(k=0;k<5;k++)
{
if(temp.equals(nonter[k]))
{
foltemp=firsts[k];
if(foltemp.contains("e"))
```

```
{
foltemp=foltemp+follows[j];
} else
{
foltemp=foltemp+firsts[k];
}
}
}
for(k=0;k<6;k++)
{
if(temp.equals(ter[k]))
{
foltemp=foltemp+temp;
}
}
} if(ip[j][1].length()==2)
{
foltemp=foltemp+follows[j];
}
}
}
follows[i]=foltemp;
foltemp="";
}
for(i=0;i<5;i++)
{
if(follows[i].contains("e"))
{
follows[i]=follows[i].replace("e","");
}
}
System.out.println();
System.out.println();
System.out.println("\t"+"FIRST"+"\t"+"FOLLOW");
for(i=0;i<5;i++)
{
System.out.println(nonter[i]+"="+"\t"+firsts[i]+"\t"+follows[i]);
}
}
}
```

Output:

```
/*
C:Desktop\ccpro>java fafp2
ASSUMING E'=N,T'=M,id=d,epsilon=e for coding purposes.
GIVEN EXPRESSION
E=TN
```

N=+TN/e
T=FM
M=*FM/e
F=(E)/d
FIRST FOLLOW
E= (d $)
N= +e $)
T= (d +$)
M= *e +$)
F= (d *+$)
*/

Conclusion:

Thus we have successfully implemented the FIRST and FOLLOW for given grammar.

G.K. Gujar Memorial Charitable Trust's **DR.ASHOK GUJAR TECHNICAL INSTITUTE'S** **DR. DAULATRAO AHER COLLEGE OF ENGINEERING, KARAD**	
Lab Manual	
Subject : Compiler Construction	
Expt. No. 8	Program to implement code optimization for given statements.

Aim: Program to implement code optimization for given statements.

Theory:

The code optimization in the synthesis phase is a program transformation technique, which tries to improve the intermediate code by making it consume fewer resources (i.e. CPU, Memory) so that faster-running machine code will result. Compiler optimizing process should meet the following objectives :

- The optimization must be correct, it must not, in any way, change the meaning of the program.
- Optimization should increase the speed and performance of the program.
- The compilation time must be kept reasonable.
- The optimization process should not delay the overall compiling process.

When to Optimize?

Optimization of the code is often performed at the end of the development stage since it reduces readability and adds code that is used to increase the performance.

Why Optimize?

Optimizing an algorithm is beyond the scope of the code optimization phase. So the program is optimized. And it may involve reducing the size of the code. So optimization helps to:

- Reduce the space consumed and increases the speed of compilation.
- Manually analyzing datasets involves a lot of time. Hence we make use of software like Tableau for data analysis. Similarly manually performing the optimization is also tedious and is better done using a code optimizer.
- An optimized code often promotes re-usability.

Types of Code Optimization: The optimization process can be broadly classified into two types :

1. **Machine Independent Optimization:** This code optimization phase attempts to improve the **intermediate code** to get a better target code as the output. The part of the intermediate code which is transformed here does not involve any CPU registers or absolute memory locations.
2. **Machine Dependent Optimization:** Machine-dependent optimization is done after the **target code** has been generated and when the code is transformed according to the target machine architecture. It involves CPU registers and may have absolute memory references rather than relative references. Machine-dependent optimizers put efforts to take maximum **advantage** of the memory hierarchy.

Where to apply Optimization?

Now that we learned the need for optimization and its two types,now let's see where to apply these optimization.

- **Source program:** Optimizing the source program involves making changes to the algorithm or changing the loop structures. The user is the actor here.
- **Intermediate Code:** Optimizing the intermediate code involves changing the address calculations and transforming the procedure calls involved. Here compiler is the actor.
- **Target Code:** Optimizing the target code is done by the compiler. Usage of registers, and select and move instructions are part of the optimization involved in the target code.
- **Local Optimization:** Transformations are applied to small basic blocks of statements. Techniques followed are Local Value Numbering and Tree Height Balancing.
- **Regional Optimization:** Transformations are applied to Extended Basic Blocks. Techniques followed are Super Local Value Numbering and Loop Unrolling.
- **Global Optimization:** Transformations are applied to large program segments that include functions, procedures, and loops. Techniques followed are Live Variable Analysis and Global Code Replacement.
- **Interprocedural Optimization:** As the name indicates, the optimizations are applied inter procedurally. Techniques followed are Inline Substitution and Procedure Placement.

Advantages of Code Optimization:

Improved performance: Code optimization can result in code that executes faster and uses fewer resources, leading to improved performance.
Reduction in code size: Code optimization can help reduce the size of the generated code, making it easier to distribute and deploy.
Increased portability: Code optimization can result in code that is more portable across different platforms, making it easier to target a wider range of hardware and software.
Reduced power consumption: Code optimization can lead to code that consumes less power, making it more energy-efficient.
Improved maintainability: Code optimization can result in code that is easier to understand and maintain, reducing the cost of software maintenance.

Disadvantages of Code Optimization:

Increased compilation time: Code optimization can significantly increase the compilation time, which can be a significant drawback when developing large software systems.
Increased complexity: Code optimization can result in more complex code, making it harder to understand and debug.
Potential for introducing bugs: Code optimization can introduce bugs into the code if not done carefully, leading to unexpected behavior and errors.
Difficulty in assessing the effectiveness: It can be difficult to determine the effectiveness of code optimization, making it hard to justify the time and resources spent on the process.

Common Sub expressions elimination:

An occurrence of an expression E is called a common sub-expression if E was previously computed, and the values of variables in E have not changed since the previous computation. We can avoid computing again and again the expression if we can use the previously computed value.
For example
t1: = 4*i
t2: = a [t1]
t3: = 4*j
t4: = 4*i
t5: = n
t6: = b [t4] +t5
The above code can be optimized using the common sub-expression elimination as
t1: = 4*i
t2: = a [t1]
t3: = 4*j
t5: = n
t6: = b [t1] +t5
The common sub expression t4: =4*i is eliminated as its computation is already in t1. And value of i is not been changed from definition to use.

Program:

```
import java.io.*;
class opt
{
public static void main(String args[])throws IOException
{
int i,j=0,k=1,count=0;
FileReader f=new FileReader("jenny.txt");
BufferedReader in=new BufferedReader(f);
String input[] =new String[5];
int arr[]=new int[5];
char temp[]=new char[5];
```

```
String lineget,temp1;
while(k!=0)
{
lineget=in.readLine();
if(lineget==null)
{
k=0;
break;
}
for(i=0;i<lineget.length();i++)
{
if(lineget.charAt(i)=='=')
{
input[j]=lineget.substring(i+1);
j++;
}
}
}
for(i=0;i<3;i++)
{
System.out.println("t"+i+"="+input[i]);
}
for(i=0;i<3;i++)
{
temp1=input[i];
temp=temp1.toCharArray();
for(j=0;j<3;j++)
{
arr[i]=arr[i]+temp[j];
}
}
for(i=0;i<3;i++)
{
for(k=1;k<3;k++)
{
if(arr[i]==arr[k])
{
input[k+1]="";
}}
}
System.out.println("Output is:");
for(i=0;i<3;i++)
{
if(input[i]=="")
{
//do nothing
```

```
}
else
System.out.println("t"+i+"="+input[i]);
}
}
} /* OUTPUT
C:Desktop\ccpro $ javac opt.java
C:Desktop\ccpro $ java opt
t0=a+b
t1=b*c
t2=b+a
Output is:
t0=a+b
t1=b*c
C:Desktop\ccpro $
*/
```

Conclusion:
Thus we have successfully implemented code optimization for given statements.

	G.K. Gujar Memorial Charitable Trust's **DR.ASHOK GUJAR TECHNICAL INSTITUTE'S** **DR. DAULATRAO AHER COLLEGE OF ENGINEERING, KARAD**
	Lab Manual
Subject : Compiler Construction	
Expt. No. 9	Write a program to implement code generator for the statements in three address code format.

Aim: Write a program to implement code generator for the statements in three address code format.

Theory:

A code generator generates target code for a sequence of three- address statements and effectively uses registers to store operands of the statements.

For example: consider the three-address statement **a := b+c** It can have the following sequence of codes:

ADD Rj, Ri Cost = 1 // if Ri contains b and Rj contains c (or)
ADD c, Ri Cost = 2 // if c is in a memory location (or)
MOV c, Rj Cost = 3 // move c from memory to Rj and add ADD Rj, Ri

Register and Address Descriptors:

- ✓ A register descriptor is used to keep track of what is currently in each registers. The register descriptors show that initially all the registers are empty.
- ✓ An address descriptor stores the location where the current value of the name can be found at run time.

A code-generation algorithm:

The algorithm takes as input a sequence of three-address statements constituting a basic block.
For each three-address statement of the form x : = y op z**,** perform the following actions:

1. Invoke a function *getreg* to determine the location L where the result of the computation y op z should be stored.
2. Consult the address descriptor for y to determine y', the current location of y. Prefer the register for y' if the value of y is currently both in memory and a register. If the value of y is not already in L, generate the instruction **MOV y' , L** to place a copy of y in L.
3. Generate the instruction **OP z' , L** where z' is a current location of z. Prefer a register to a memory location if z is in both. Update the address descriptor of x to indicate that x is in location L. If x is in L, update its descriptor and remove x from all other descriptors.
4. If the current values of y or z have no next uses, are not live on exit from the block, and are in registers, alter the register descriptor to indicate that, after execution of x : = y op z , those registers will no longer contain y or z.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
char op[2],arg1[5],arg2[5],result[5];
void main()
{
  FILE *fp1,*fp2;
  fp1=fopen("c:\tc\bin\input.txt","r");
  fp2=fopen("c:\tc\bin\output.txt","w");
  while(!feof(fp1))
  {

          fscanf(fp1,"%s%s%s%s",op,arg1,arg2,result);
          if(strcmp(op,"+")==0)
          {
            fprintf(fp2,"\nMOV R0,%s",arg1);
            fprintf(fp2,"\nADD R0,%s",arg2);
            fprintf(fp2,"\nMOV %s,R0",result);
          }
           if(strcmp(op,"*")==0)
          {
            fprintf(fp2,"\nMOV R0,%s",arg1);
            fprintf(fp2,"\nMUL R0,%s",arg2);
            fprintf(fp2,"\nMOV %s,R0",result);
          }
          if(strcmp(op,"-")==0)
          {
            fprintf(fp2,"\nMOV R0,%s",arg1);
            fprintf(fp2,"\nSUB R0,%s",arg2);
            fprintf(fp2,"\nMOV %s,R0",result);
          }
             if(strcmp(op,"/")==0)
          {
            fprintf(fp2,"\nMOV R0,%s",arg1);
            fprintf(fp2,"\nDIV R0,%s",arg2);
            fprintf(fp2,"\nMOV %s,R0",result);
          }
if(strcmp(op,"=")==0)
          {
            fprintf(fp2,"\nMOV R0,%s",arg1);
```

```
            fprintf(fp2,"\nMOV %s,R0",result);
          }
          }//while end
          fclose(fp1);
          fclose(fp2);
          getch();
}
```

Input:

```
+ a b t1
* c d t2
- t1 t2 t
= t ? x
```

Output:

```
MOV R0,a
ADD R0,b
MOV t1,R0
MOV R0,c
MUL R0,d
MOV t2,R0
MOV R0,t1
SUB R0,t2
MOV t,R0
MOV R0,t
MOV x,R0
```

Conclusion:

Thus we have successfully generated code for expressions in three address codes.

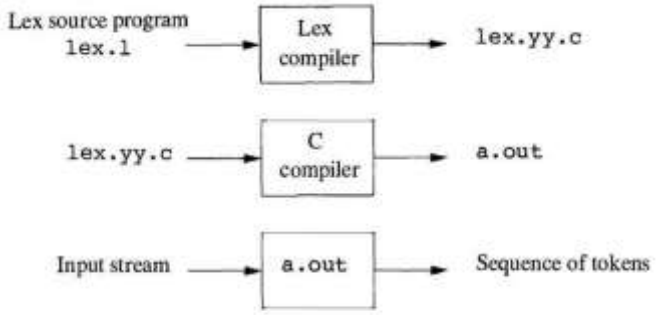
	G.K. Gujar Memorial Charitable Trust's **DR.ASHOK GUJAR TECHNICAL INSTITUTE'S** **DR. DAULATRAO AHER COLLEGE OF ENGINEERING, KARAD**
	Lab Manual
Subject : Compiler Construction	
Expt. No. 10	To implement LEX and YACC program using FLEX and YACC.

Aim: To implement LEX and YACC program using FLEX and YACC.

Theory:

LEX:

We introduce a tool called Lex, or in a more recent implementation Flex, that allows one to specify a lexical analyzer by specifying regular expressions to describe patterns for tokens. The input notation for the Lex tool is referred to as the Lex language and the tool itself is the Lex compiler. The Lex compiler transforms the input patterns into a transition diagram and generates code, in a file called *lex.yy.c* that simulates this transition diagram.



Creating a lexical analyzer with Lex

Structure of Lex Programs:

A Lex program has the following form:

declarations

%%

translation rules

%%

auxiliary functions

The declarations section includes declarations of variables, manifest constants (identifiers declared to stand for a constant, e.g., the name of a token), and regular definitions. The translation rules each have the form

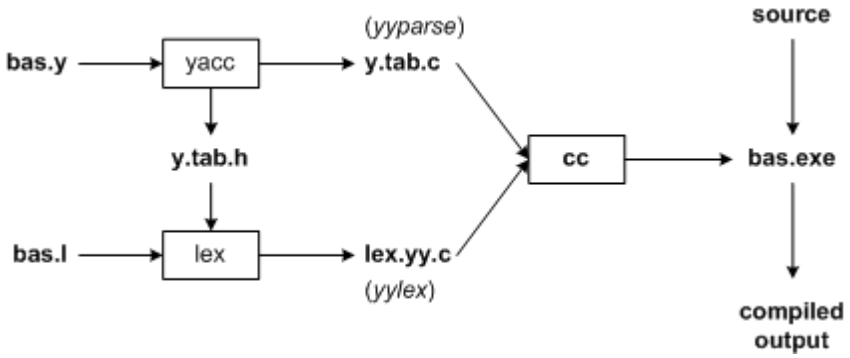
Pattern {Action}

Each pattern is a regular expression, which may use the regular definitions of the declaration section. The actions are fragments of code, typically written in C, although many variants of Lex using other languages have been created.

The third section holds whatever additional functions are used in the actions. Alternatively, these functions can be compiled separately and loaded with the lexical analyzer.

YACC

It is a computer program for the Unix operating system. The name is an acronym for "Yet Another Compiler-Compiler". It is a LALR parser generator, generating a parser, the part of a compiler that tries to make syntactic sense of the source code, specifically a LALR parser, based on an analytic grammar written in a notation similar to BNF. It was developed in 1970 by Stephen C. Johnson at AT&T Corporation and originally written in the B programming language. Yacc and similar programs (largely reimplementation) have been very popular.



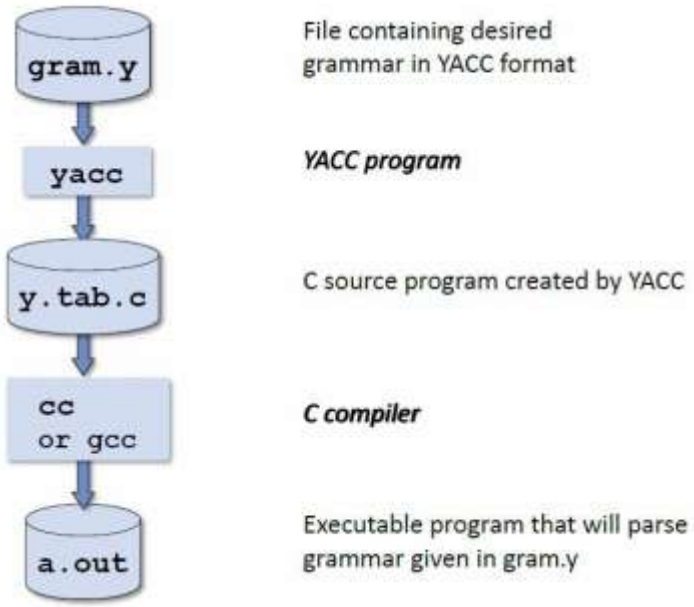
Working of YACC:

YACC is designed for use with C code and generates a parser written in C. The parser is configured for use in conjunction with a lex-genrated scanner and relies on standard shared features and calls the function yylex as a scanner co routine. We provide a grammar specification file, which is traditionally named using a.y extension. We invoke yacc on the file .y file and it creates y.tab.h and y.tab.c files containing a thousand or so lines of intense c code that implements an efficient LALR(1) parser out of grammar, including the code for the actions we specified. The file provides an extern function yyparse.y that will attempt to successfully parse a valid sentence. We compile that c file normally ,link with the rest of our code, and we have a parser .By default ,the parser reads from stdin and writes to stdout, just like a lex generated scanner does,

%yacc myFile.y creates y.tab.c of C code for parser

%gcc -c y.tab.c compiles parser code

%gcc -o parser y.tab.o lex.yy.o –ll-ly link parser, scanner, lib.



File format:
Definitions
%%
Rules
%%
Supplementary codes

Definition Section: All code between % and % is copied to the C file. The definitions section is where we configure various parser features such as defining token codes, establishing operator precedence and associativity and setting up variables used to communicate between the scanner and the parser.

Rules Section: The required productions section is where we specify the grammar rule.

Supplementary Code Section: It is used for ordinary C code that we want copied verbatim to the generated C file, declarations are copied to the top of the file, user subroutines to the bottom.

Source Program 1:

```
%%
[^+-/*=0-9A-Za-z\n] { printf("Error"); }
[+-/*=0-9] { printf("<%s>", yytext); }
[A-Za-z] { printf("<id>"); }
%%
```

Output

```
6+8
<6><+><8>
```

a=b*c
<id><=><id><*><id>
a=b+c&
<id><=><id><+><id>Error

calc.lex

```
%{
#include <stdlib.h>
void yyerror(char *);
#include "y.tab.h"
%}
%%
[0-9]+ {
yylval = atoi(yytext);
return INTEGER;
} [-+\n] return *
yytext;
[ \t] ; /* skip whitespace */
. yyerror("invalid character");
%%
int yywrap(void) {
return 1;
}
```

calc.yac

```
%{
#include <stdio.h>
int yylex(void);
void yyerror(char *);
%}
%token INTEGER
%%
program:
program expr '\n' { printf("%d\n", $2); }
|
;
expr:
INTEGER { $$ = $1; }
| expr '+' expr { $$ = $1 + $3; }
| expr '-' expr { $$ = $1 - $3; }
| expr '*' expr { $$ = $1 * $3; }
| expr '/' expr { $$ = $1 / $3; }
;
%%
void yyerror(char *s) {
fprintf(stderr, "%s\n", s);
}
```

```
int main(void) {
yyparse();
return 0;
}
```

Conclusion:

Thus we have successfully implemented LEX program using the FELX and calculator program of YACC using YACC.

Prepared by :Ms. D. R. Kamble	Approved by: Head of Department

Laboratory Manual

2023-24

Sub:- **Database Engineering (DBE)**

T.Y.B.Tech (CSE)

Prepared By
Prof. Shinde T.R.

G.K. Gujar Memorial Charitable Trust's,
Dr.Ashok Gujar Technical Institute's
Dr. Daulatrao Aher College of Engineering, Karad.
Vidyanagar Ext. Banawadi, Tal. Karad 415124, Dist. Satara, Maharashtra INDIA

Prepared By:- Prof. Shinde T.R.	Approved By: Head of Department

Student Responsibilities

1. In the very beginning of the laboratory work, the student has to do programming individually. For this reason, regular attendance is strictly required.
2. Every laboratory session is divided into two parts. In the first part, the instructor will be lecturing on the test objective, procedure and data collection. In the second part, the students, organized in groups (individual student), are required to do the programming. In order to perform the experiment within the assigned period, and to gain the maximum benefit from the experiment, the students must familiarize themselves with the purpose, objective, and procedure of the experiment before coming to the laboratory. Relevant lecture notes and laboratory manual should be studied carefully and thoroughly.
3. At the end of the experiment, every student should submit the written algorithm & programs & testes result for approval by the instructor.
4. It should be understood that laboratory facilities and equipment's (Hardware & Software) are provided to enhance the learning process.
5. The equipment's must be properly cared after every laboratory session. Also, students should always take precautions to avoid any possible hazards. Students must follow laboratory regulations provided at the end of this section.

G.K. Gujar Memorial Charitable Trust's,
Dr.Ashok Gujar Technical Institute's
Dr. Daulatrao Aher College of Engineering, Karad.
Vidyanagar Ext. Banawadi, Tal. Karad 415124, Dist. Satara, Maharashtra INDIA

Department of Computer Science & Engineering

EXPERIMENT LIST

Department: Computer Science and Engineering
Class: Third Year (T.Y. B.Tech)
Subject: Database Engineering

Academic Year: 2023-24
Semester: VI
Semester Duration: 6 months

Exp No.	Experiment Name
1	Introduction to Database Management System
2	Study and Draw E-R Model of DBMS
3	Installation & Demonstration of DBMS like MySql, Oracle etc. Reduce E-R model into tables
4	Implementation of program to modify data from database/data dictionary using JAVA and Oracle/SQL.
5	Study of Basic SQL Commands: DDL & DML queries.
6	Implementation of SQL Query constraints with referential integrity constraints.
7	Implementation of SQL Query Processing: program to show use of SQL Clauses-Order by, group by and between clauses.
8	Implementation of SQL Query Aggregate functions & set operations
9	Implementation of SQL query Join operations.
10	Implementation of String operations in SQL.
11	Implementation of Views for any created table.
12	Implementation of Static Hashing Technique using java.
13	Study of NoSql.

Subject In charge
Prof.Shinde T.R.

HOD
Prof.Kakade S.P.

Experiment No.1

Title: Introduction to Database Management System

Aim: To Study the Database Management System (DBMS)

Prerequisites: Set Theory, Operating System, Data Structures

Theory:

- What is DBMS? Explain its Architecture with diagram
- Properties of DBMS
- Purpose of DBMS: Drawbacks of conventional file system
- Application of DBMS.

Conclusion: We have studied the Database management system, architecture and its user roles.

Questions:

1. What is Database Management System?
2. What are the Database System applications?
3. What is difference between Database System and File System?
4. Explain data abstraction, schema & data independence
5. Difference between Data Vs Information Vs Metadata
6. List and explain the various database users

Experiment No. 2

Title: Study and Draw E-R Model of DBMS
Aim: To learn and draw the E-R model of DBMS

Theory:

- Explain Entity Relationship model in detail
- Components of an E-R Model

An ERD typically consists of four different graphical components:

1. **Entity.** A data entity is anything real or abstract about which we want to store data. Entity types fall into five classes: roles, events, locations, tangible things or concepts. E.g. employee, payment, campus, book. Specific examples of an entity are called instances. E.g. the employee John Jones, Mary Smith's payment, etc.

2. **Relationship.** A data relationship is a natural association that exists between one or more entities. E.g. Employees process payments.

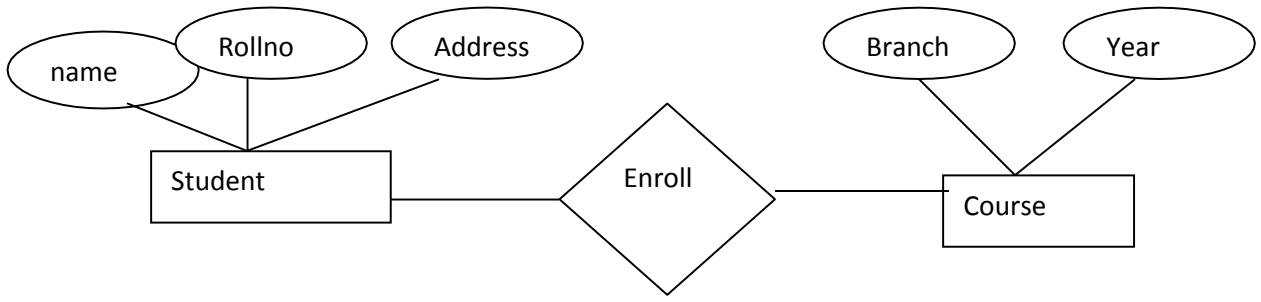
3. **Cardinality.** Defines the number of occurrences of one entity for a single occurrence of the related entity. E.g. an employee may process many payments but might not process any payments depending on the nature of her job.

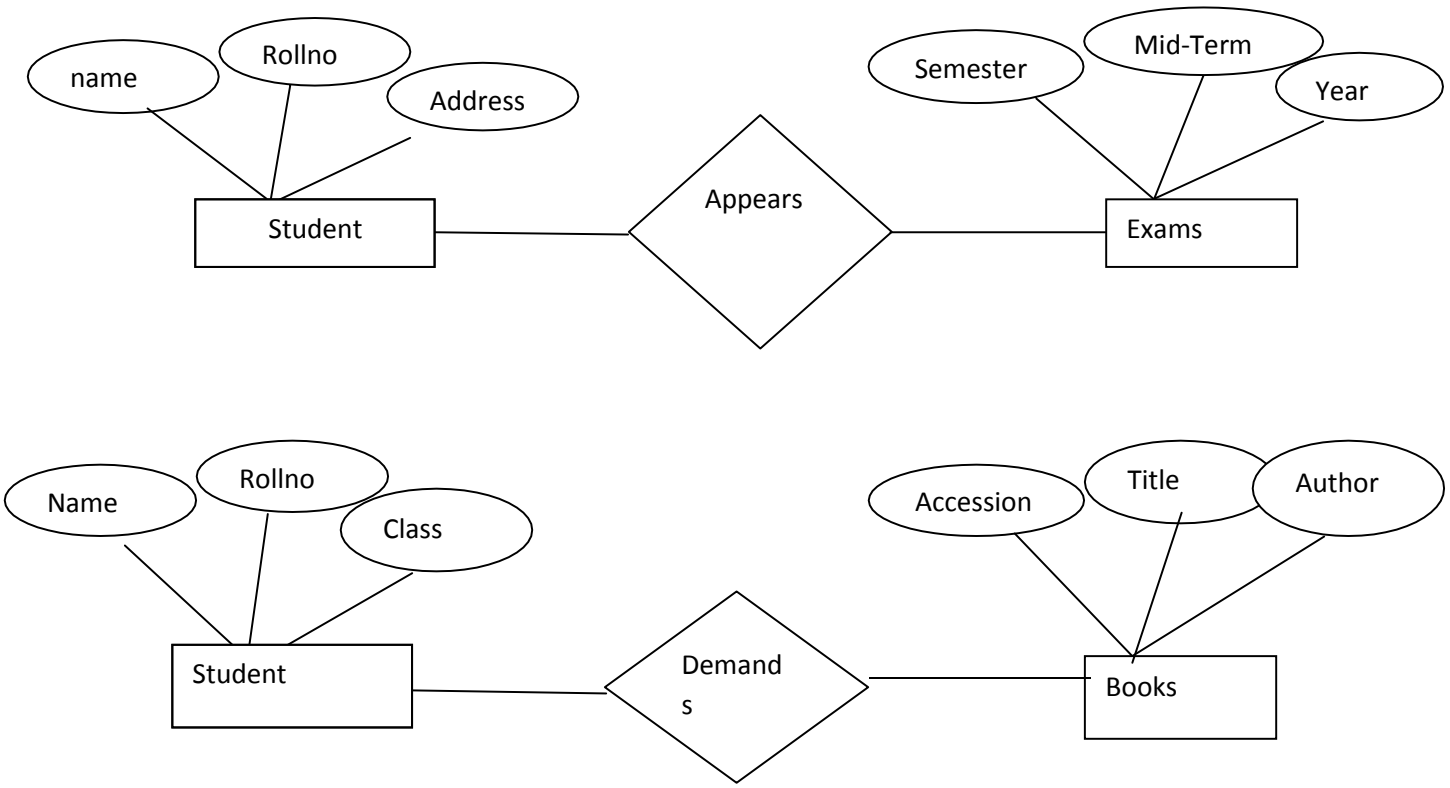
4. **Attribute**. A data attribute is a characteristic common to all or most instances of a particular entity. Synonyms include property, data element, field. E.g. Name, address, Employee Number, pay rate are all attributes of the entity employee. An attribute or combination of attributes that uniquely identifies one and only one instance of an entity is called a primary key or identifier. E.g. Employee Number is a primary key for Employee.

Components of E-R model: (Draw all sign conventions)

1. Rectangle: Represents Entity Set
2. Diamond: Represents Relationship
3. Ellipse :Represents attributes
4. Dashed Ellipse: Represents derived attributes
5. Double Ellipse: Represents multi valued attributes
6. Lines: Link to entity set and relationship
7. Double Lines: Total Participation

E-R Diagram for student activities at college:





Conclusion: We have studied the concepts of E-R Model and how to draw it.

Questions:

1) What is E-R Model? Draw it with an example
2) What is E-R diagram and its components?
3) Draw an ER diagram for the University database, Car Insurance Company and Bank Management System.
4) Draw E-R diagram and create schema in SQL for relation: Customer buying a car from car manufacturing company. If car has an accident then Insurance compony is liable to pay for damage recovery of the car
5) Draw E-R diagram and create schema in SQL for following relations in college management system

   Instructor (i_id, name, dept_name, salary)

   Teaches (id, course_id, section_id, sem, year)

   Student (sid, name, dept_name, tot_credits)

   Department (dept_name, building, budget)

   Course (course_id, title, dept_name, credits)

   Advisor(sid, i_id)

Experiment No. 3

Title: Installation & Demonstration of DBMS like MySql, Oracle etc. Reduce E-R model into tables

Aim: To Install & Demonstrate of DBMS like MySql, Oracle etc. and Reduce E-R model into tables (Practice session: Students should be allowed to choose appropriate DBMS software, install it, configure it and start working on it. Create sample tables, execute some queries, use SQLPLUS features, use PL/SQL features like cursors on sample database. Students should be permitted to practice appropriate User interface creation tool and Report generation tool)

Steps:

Welcome to Oracle Database 10g Express Edition (Oracle Database XE)! This tutorial gets you quickly up and running using Oracle Database XE by creating a simple application. This guide covers the following topics:

- Logging in as the Database Administrator
- Unlocking the Sample User Account
- Logging in as the Sample User Account
- Creating a Simple Application
- Running Your New Application
- Using the Oracle Database XE Menus

1. Logging in as the Database Administrator:

The first thing you need to do is to log in as the Oracle Database XE Administrator. Follow these steps:

1. Open the Database Home Page login window:

On Windows, from the Start menu, select **Programs**(or All Programs),then Oracle Database 10g Express Edition, and then **Go To Database Home Page**.

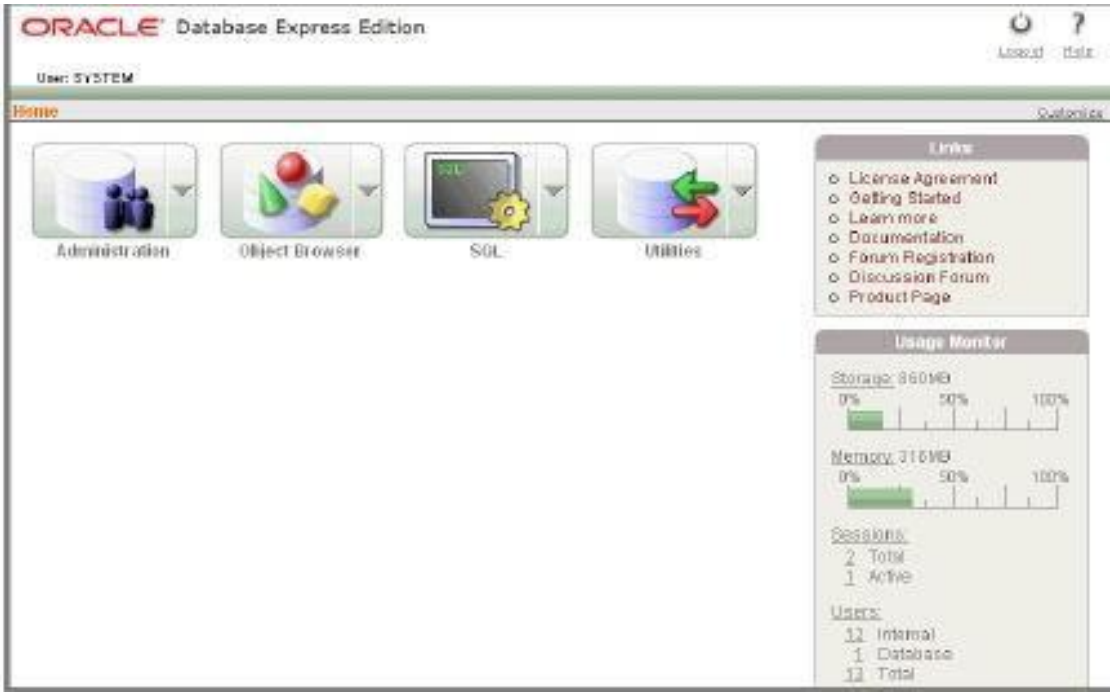
- On Linux, click the Application menu (on Gnome) or the K menu (on KDE), then point to Oracle Database 10g Express Edition, and then Go To Database Home Page.

2. At the Database Home Page login window, enter the following information:

- Username: Enter system for the user name.
- Password: Enter the password that was specified when Oracle Database XE was installed.

3.Click Login

The Oracle Database XE home page appears.



2. Unlocking the Sample User Account

To create your application, you need to log in as a database user. Oracle Database XE comes with a sample database user called HR. This user owns a number of database tables in a sample schema that can be used to create applications for a fictional Human Resources department. However, for security reasons, this user's account is locked. You need to unlock this account before you can build a sample application.

To unlock the sample user account:

1. Make sure you are still logged on as the database administrator, as described in the previous section.
2. Click the Administration icon, and then click Database Users.
3. Click the HR schema icon to display the user information for HR.

- Under Manage Database User, enter the following settings:
- Password and Confirm Password: Enter hr for the password.
- Account Status: Select Unlocked.
- Roles: Ensure that both CONNECT and RESOURCE are enabled.

4. Click Alter User.

Now you are ready to create your first application.

3. Logging in as the Sample User Account

To log in as the sample user account:

1. Log out from the database administrator account by clicking Logout in the upper right corner of the Database Home Page.
2. In the window, click Login.
3. In the Login window, enter hr for both the user name and password.
4. Click Login.

The Database Home Page appears.

4. Creating a Simple Application

Creating an application is an easy way to view and edit your database data. You create this application based on the EMPLOYEES table, which is part of the HR schema.

To create an application based on the EMPLOYEES table:

1. On the Database Home Page, click the Application Builder icon.
2. Click the Create button.
3. Under Create Application, select Create Application and click Next.
4. Under Create Application:

a. Name: Enter MyApp.

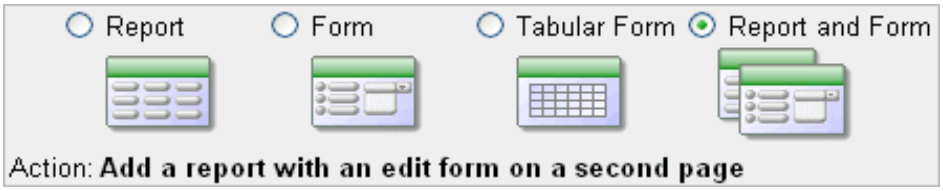
b. Accept the remaining defaults.

c. Click Next.

Next, add pages to your application.

5. Under Add Page:

a. For Select Page Type, select Report and Form.

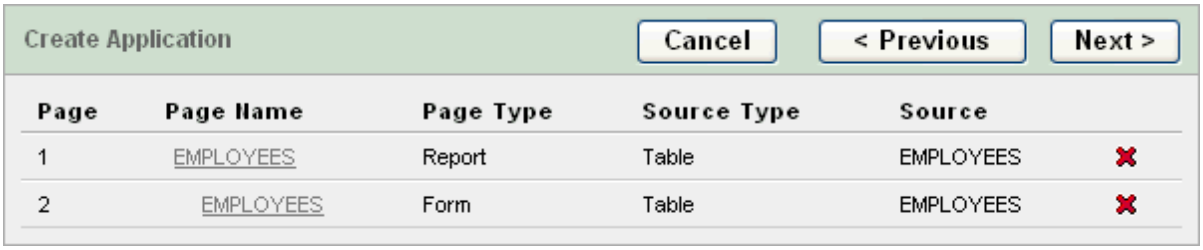


Notice that Action describes the type of page you are adding.

b. Next to the Table Name field, click the up arrow, and then select EMPLOYEES from the Search Dialog window.

c. Click Add Page.

Two new pages display at the top of the page, under Create Application.



d. Click Next

6. On the Tabs panel, accept the default (One Level of Tabs) and click Next.

7. On the Shared Components panel, accept the default (No) and click Next.

This option enables you to import shared components from another application. Shared components are common elements that can display or be applied on any page within an application.

8. For Authentication Scheme, Language, and User Language Preference Derived From, accept the defaults and click Next.

9. For the theme, select Theme 2 click Next. Themes are collections of templates that you can use to define the layout and style of an entire application.

10. Confirm your selections. To return to a previous wizard page, click Previous.

To accept your selections, click Create.

After you click Create, the following message displays at the top of the page:

Application created successfully.

5.Running Your New Application:

To run your application:

1. Click the Run Application icon.

In the log in page, enter hr for both the User Name and Password. Your application appears, showing the EMPLOYEES table.

3. Explore your application.

You can query the EMPLOYEES table, if you want. To manage the application, use the Developer toolbar at the bottom on the page.

The Developer toolbar offers a quick way to edit the current page, create a new page, control, or component, view session state, or toggle debugging or edit links on and off.

4. To exit your application and return to Application Builder, click Edit Page 1 on the Developer toolbar.

5. To return to the Database Home Page, select the Home breadcrumb at the top of the page.

Congratulations! You have just created your first application using Oracle Database XE.

6. Using the Oracle Database XE Menus:

You can use the Oracle Database XE menus to perform basic functions with Oracle Database XE. To see the menus, do the following:

■ On Windows, from the Start menu, select Programs (or All Programs) and then Oracle Database 10g Express Edition.

■ On Linux, click the Application menu (on Gnome) or the K menu (on KDE) and then point to Oracle Database 10g Express Edition. The following menu items are available:

■ Get Help: Displays the following selections:

- Go To Online Forum: Displays the online forum for discussions about Oracle Database XE.
- Read Documentation: Displays the Oracle Database XE documentation library on the Internet.
- Read Online Help: Displays the Oracle Database XE online help. This help is only available if the database is started.

■Register For Online Forum: Allows you to register for the Oracle Database XE online forum.

■Backup Database: In NOARCHIVELOG mode (the default), shuts down the database, backs it up, and then restarts it. In ARCHIVELOG mode, performs an online backup of the database. For more information on backups, refer to Oracle Database Express Edition 2 Day DBA.

■ Get Started: Link to this tutorial.

■ Go To Database Home Page: Displays the Oracle Database XE Home Page in your default browser. "Logging in as the Database Administrator" on page 1 explains how to log into this home page as a database administrator.

■ Restore Database: Shuts down and then restores the database to the most recent backup. For more information on restoring a database, refer to Oracle Database Express Edition 2 Day DBA.

■ Run SQL Command Line: Starts the SQL Command Line utility for Oracle Database XE. To connect to the database, issue the following command at the **SQL prompt that appears**:

Connect username/password

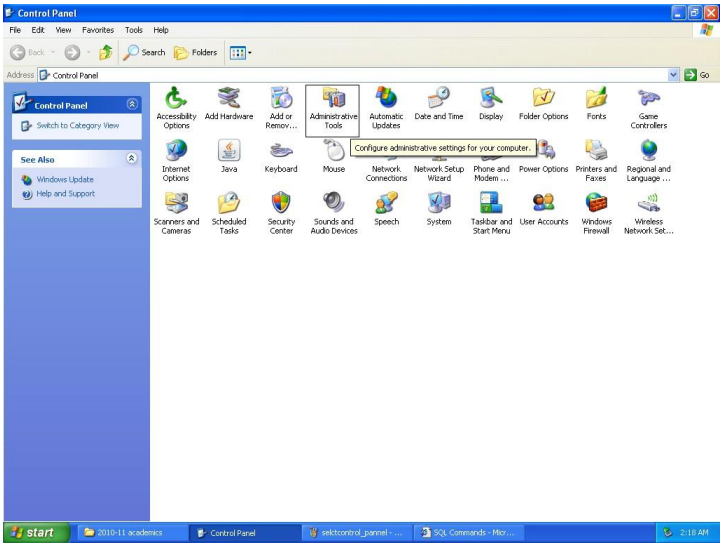
where username is the user name, such as sys, system, or another account name, and password is the password that was assigned when Oracle Database XE was installed. The get help, you can enter the command help at the SQL prompt, once you have connected to the database.

■ Start Database: Starts Oracle Database XE. By default, the database is started for you after installation and every time your computer is restarted. However, if you think the database is not running you can use this menu item to start it.

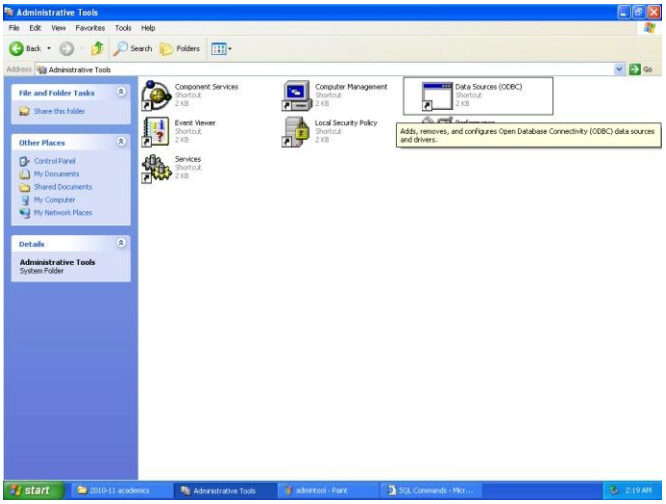
Stop Database: Stops Oracle Database XE.

How to create ODBC connectivity

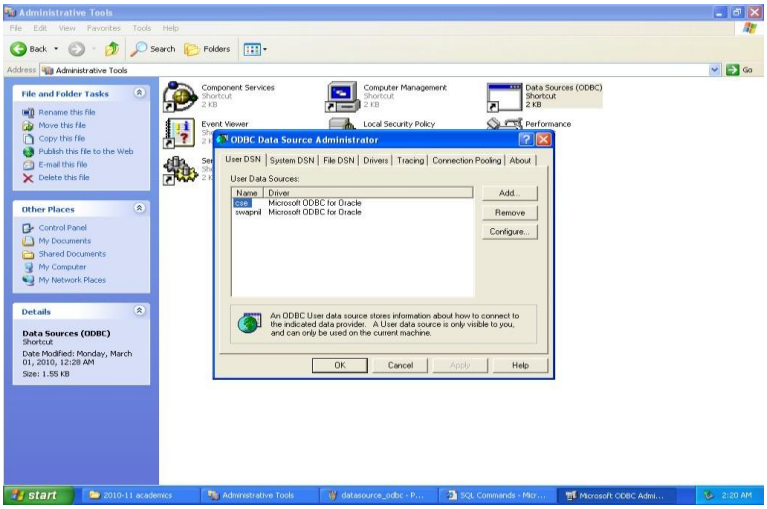
STEP 1: go to control panel and open Administrative tools



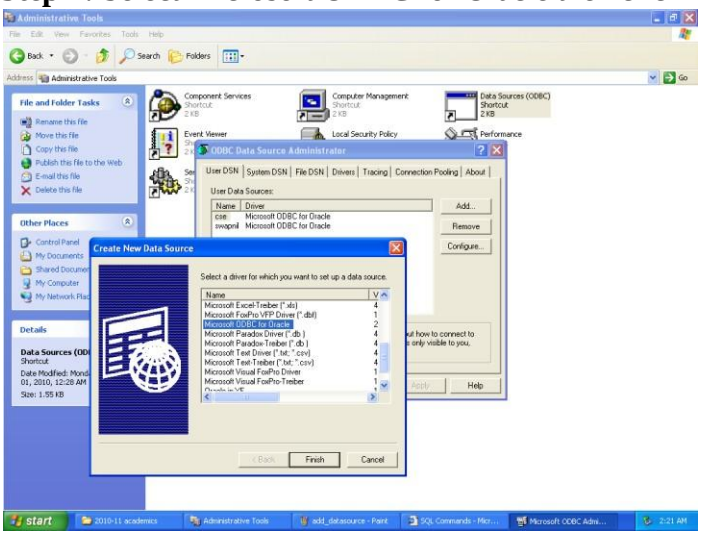
Step 2: select data source (ODBC) and open



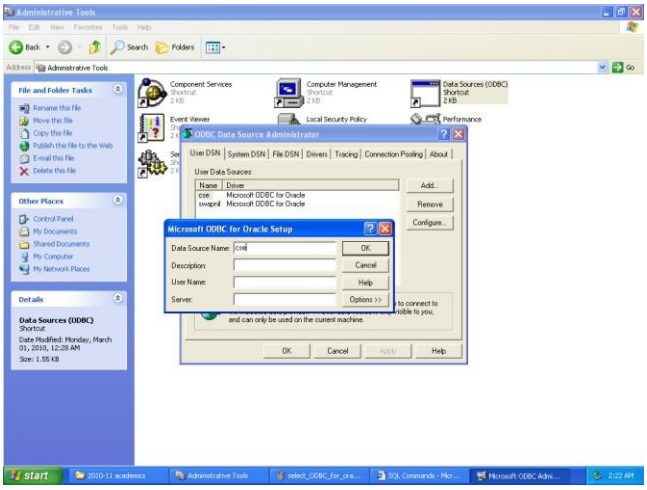
Step 3: Click Add…



Step 4: Select Microsoft ODBC for Oracle then click Finish.



Step 5: Enter Data Source Name(in prevision example it is "cse") and click Ok

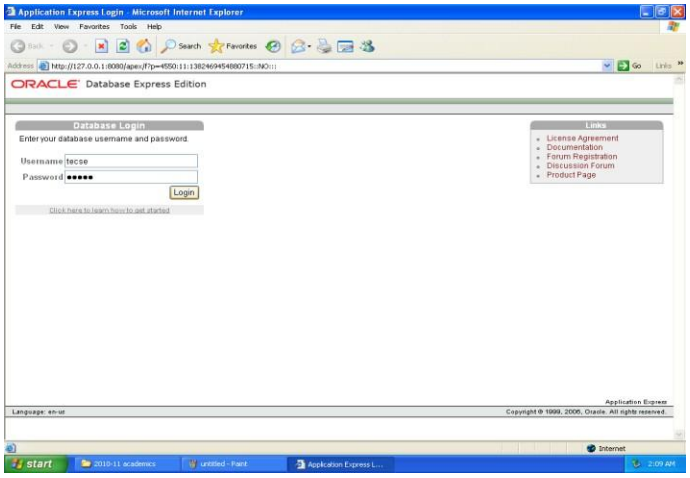


After compilation it will create DSN for oracle. This DSN is used during java program for connection con = DriverManager.getConnection("jdbc:odbc:**cse**","tecse","tecse");

How to run SQL queries in oracle 10 g.

Step1 : Select Oracle Database 10g Express Edition - Go To Database Home Page from Start menu

Step 2: login to oracle 10 g using username and password

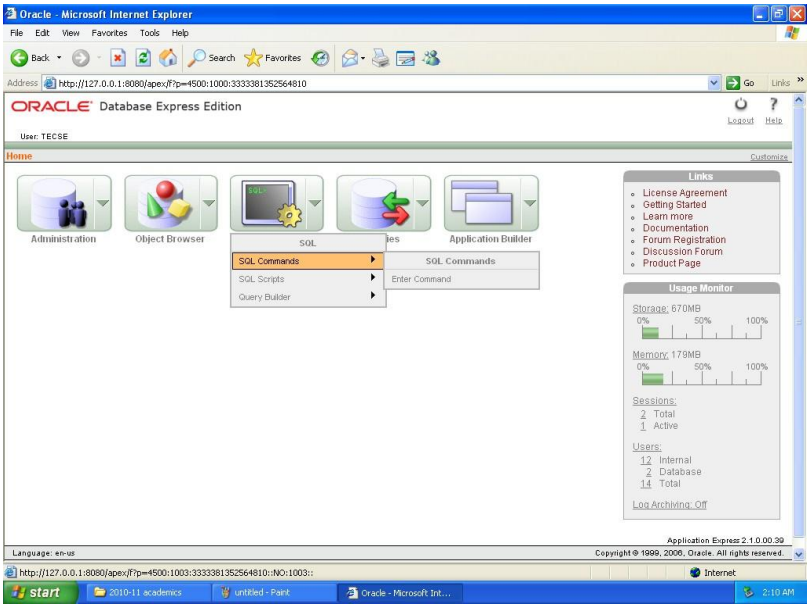


In our example we used username: tecse and password; tecse

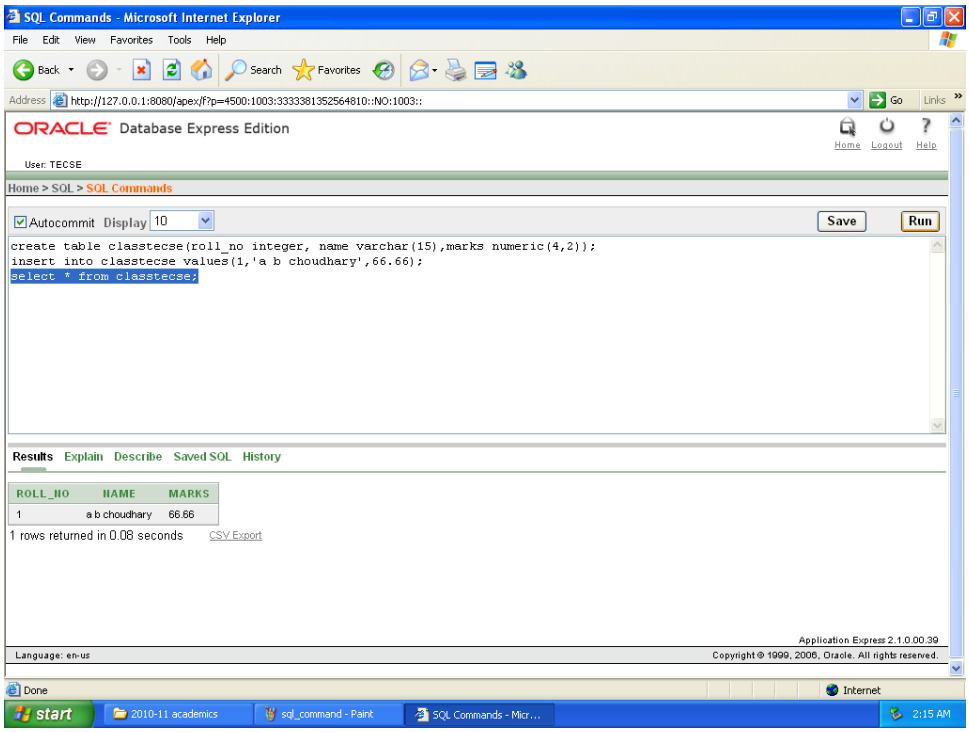
i.e. con = DriverManager.getConnection("jdbc:odbc:cse","tecse","tecse");

Passsword given at the time of installation is for user system which can used for create new users.

Step 3 : select SQL – SQL Commands – Enter Command



Step 4: type query then select and click Run button. You will get result in result window



e.g. select * from classtecse

Also we create different users and assign privileges through create user.

Role Name	Description
CONNECT	Enables a user to connect to the database. Grant this role to any user or application that needs database access.

Role Name	Description
RESOURCE	Enables a user to create certain types of schema objects in his own schema. Grant this role only to developers and to other users that must create schema objects. This role grants a subset of the *create object* system privileges. For example, it grants the CREATE TABLE system privilege, but does not grant the CREATE VIEW system privilege. It grants only the following privileges: CREATE CLUSTER, CREATE INDEXTYPE, CREATE OPERATOR, CREATE PROCEDURE, CREATE SEQUENCE, CREATE TABLE, CREATE TRIGGER, CREATE TYPE
DBA	Enables a user to perform most administrative functions, including creating users and granting privileges; creating and granting roles; creating and dropping schema objects in other users' schemas; and more. It grants all system privileges, but does not include the privileges to start up or shut down the database. It is by default granted to user SYSTEM.

Conclusion: We have install Oracle Database 10g Express Edition (Oracle Database XE) successfully.

Experiment No.4

Title: Implementation of program to modify data from database/data dictionary using JAVA and Oracle/SQL.

Aim: To understand the connectivity to a database table using java. In java database is accessed through java database connectivity (JDBC).

Requirements: Mysql database software, Java software.

Theory/ Design:

The four main components required for implementation of JDBC are application, driver manager, data source specific drivers and corresponding data sources. The application program establishes/terminates connection with data. Driver manager will load JDBC drivers and pass JDBC function calls from the application to the driver. The data source processes commands from the driver and return the results.

1. Drivers for the database are loaded by using Class.for Name.

2. The getConnection() function of DriverManager class of JDBC is used to create the connection object. The URL specifies the machine name (db.onlinebook.edu)

```
public static void Sample(String DB_id, String U-id, String Pword)
{
StringURL =”jdbc:oracle:oci8:@db.onlinebook.edu:100:onbook_db”;
Class.forName(“oracle.jdbc.driver.OracleDriver”);
Connection cn=DriverManager.getConnection(URL,U_id,Pword);
Statement st = Cn.createStatement();
try {
st.executeUpdate(“INSERT INTO AUTHOR VALUES('A010','SMITH',JONES','TEXAS')”);
 }
catch(SQLException se)
{
System.out.println(“Tuple cannot be inserted.”+se);
}
ResultSet rs = st.execute Query(“SELECT Aname,State from AUTHOR WHERE City = 'Seatle'”);
while(rs.next())
{
System.out.println(rs.getString(1) + “ “+rs.getString(2));
}
 st.close();
Cn.close();
}
```

Database Servers:-

A database server is a computer program that provides database services to other computer programs or computers, as defined by the client-server model. The term may also refer to a computer dedicated to running such a program. Database management systems frequently provide database server functionality, and some DBMS's (e.g., MySQL) rely exclusively on the client-server model for database access. In a master-slave model, database master servers are central and primary locations of data while database slave servers are synchronized backups of the master acting as proxies.

My SQL:-

The Structured Query Language (SQL) is a very popular database language, and its standardization makes it quite easy to store, update and access data. One of the most powerful SQL servers out there is called MySQL and surprisingly enough, its free.

Some of the features of MySQL Include: Handles large databases, in the area of 50,000,000+ records. No memory leaks. Tested with a commercial memory leakage detector (purify). A privilege and password system which is very flexible and secure, and which allows host-based verification. Passwords are secure since all password traffic when connecting to a server is encrypted.

PL/SQL:-

PL/SQL is a database-oriented programming language that extends Oracle SQL with procedural capabilities. We will review in this lab the fundamental features of the language and learn how to integrate it with SQL to help solve database problems.SQL statements are defined in term of constraints we wish to fix on the result of a query. Such a language is commonly referred to as*declarative*. This contrasts with the so called procedural languages where a program specifies a list of operations to be performed sequentially to achieve the desired result. PL/SQL adds *selective* (i.e. if...then...else...) and *iterative*constructs(i.e.loops) to SQL. PL/SQL is most useful to write *triggers* and *stored procedures*. Stored procedures are units of procedural code stored in a compiled form within the database.

JDBC:-JDBC stands for "Java DataBase Connectivity". It is an API (Application Programming Interface) which consists of a set of Java classes, interfaces and exceptions and a specification to which both JDBC driver vendors and JDBC developers (like you) adhere when developing applications.

JDBC is a very popular data access standard. RDBMS (Relational Database Management Systems) or third-party vendors develop drivers which adhere to the JDBC specification. Other developers use these drivers to develop applications which access those databases e.g. you'll use ConnectorJ JDBC driver to access MySQL database. Since the drivers adhered to JDBC specification, the JDBC application developers can replace one driver for their application with another better one without having to rewrite their application. If they had used some proprietary API provided by some RDBMS vendor, they will not have been able to change the driver and/or database without having to rewrite the complete application.

Who develops JDBC Specification?

SUN prepares and maintains the JDBC specification. Since JDBC is just a specification (suggestions for writing and using JDBC drivers), third-party vendors develop JDBC drivers adhering to this specification. JDBC developers then use these drivers to access data sources.

Why use JDBC?
JDBC is there only to help you (a Java developer) develop data access applications without having to learn and use proprietary APIs provided by different RDBMS vendors. You just have to learn JDBC and then you can be sure that you'll be able to develop data access applications which can access different RDBMS using different JDBC drivers.

JDBC Architecture
We'll divide it into 2 parts:

JDBC API (java.sql & javax.sql packages)

- JDBC Driver Types

JDBC API
The JDBC API is available in the java.sql and javax.sql packages. Following are important JDBC classes, interfaces and exceptions in the java.sql package:

- DriverManager - Loads JDBC drivers in memory. Can also be used to open connections to a data source.
- Connection - Represents a connection with a data source. Is also used for creatingStatement, PreparedStatement and CallableStatement objects.
- Statement - Represents a static SQL statement. Can be used to retrieve ResultSetobject/s.
- PreparedStatement - Higher performance alternative to Statement object, represents a precompiled SQL statement.
- CallableStatement - Represents a stored procedure. Can be used to execute stored procedures in a RDBMS which supports them.
- ResultSet - Represents a database result set generated by using a SELECT SQL statement.
- SQLException - An exception class which encapsulates database base access errors.

JDBC Driver Types
There are 4 types of JDBC drivers. Commonest and most efficient of which are type 4 drivers. Here is the description of each of them:

- JDBC Type 1 Driver - They are JDBC-ODBC Bridge drivers. They delegate the work of data access to ODBC API. They are the slowest of all. SUN provides a JDBC/ODBC driver implementation.
- JDBC Type 2 Driver - They mainly use native API for data access and provide Java wrapper classes to be able to be invoked using JDBC drivers.
- JDBC Type 3 Driver - They are written in 100% Java and use vendor independent Net-protocol to access a vendor independent remote listener. This listener in turn maps the vendor independent calls to vender dependent ones. This extra step adds complexity and decreases the data access efficiency.
- JDBC Type 4 Driver - They are also written in 100% Java and are the most efficient among all driver types.

Introduction to ODBC

The Open Database Connectivity (ODBC) interface is a call-based application programming interface defined by Microsoft Corporation as a standard interface to database management systems on Windows operating systems. In addition, ODBC is now widely used on many non-Windows platforms, such as UNIX and Macintosh.

Software requirements

To write ODBC applications for Adaptive Server Enterprise, you need:

- Adaptive Server Enterprise, versions 12.0 and above.
- A C compiler capable of creating programs for your environment.
- ODBC Software Development Kit.
- On non-Windows platforms, there are open source projects like unixODBC and iOBDC that release the required headers and libraries to build ODBC applications.

Problem Statement:

1. Implementation of program to modify data from database using JAVA with backend Oracle/SQL.
2. Implementation of program to create, insert and delete in database using JAVA with backend Oracle/SQL.

Conclusion: We have implemented the JAVA program to connect the SQL database by using JDBC & ODBC connection

Experiment No. 5

Title: Study of Basic SQL Commands: DDL & DML queries.

Aim: To implement DDL & DML queries of SQL

Theory / Design:

- Introduction to SQL & basic queries.

Basic SQL:

Each record has a unique identifier or primary key. SQL, which stands for Structured Query Language, is used to communicate with a database. Through SQL one can create and delete tables. Here are some commands:

- CREATE TABLE - creates a new database table
- ALTER TABLE - alters a database table
- DROP TABLE - deletes a database table
- CREATE INDEX - creates an index (search key)
- DROP INDEX - deletes an index

SQL also has syntax to update, insert, and delete records.

- SELECT - get data from a database table
- UPDATE - change data in a database table
- DELETE - remove data from a database table
- INSERT INTO - insert new data in a database table

1. SELECT

The SELECT is used to query the database and retrieve selected data that match the specific criteria that you specify:

SELECT *column1* [, *column2*, ...]
FROM *tablename*
WHERE *condition*

The conditional clause can include these operators

- = Equal
- > Greater than
- < Less than
- >= Greater than or equal
- <= Less than or equal

- <> Not equal to
- LIKE pattern matching operator

SELECT * FROM *tablename*

returns all the data from the table.

Use single quotes around text values (most database systems will also accept double quotes). Numerical values should not be enclosed in quotes.

LIKE matches a pattern. The wildcard % is used to denote 0 or more characters.

- 'A%' : matches all strings that start with *A*
- '%a' : matches all strings that end with *a*
- '%a%' : matches all strings that contain an *a*

2. CREATE TABLE

The CREATE TABLE statement is used to create a new table. The format is:

CREATE TABLE *tablename*
(*column1 data type*,
column2 data type,
column3 data type);

- char(size): Fixed length character string.
- varchar(size): Variable-length character string. Max size is specified in parenthesis.
- number(size): Number value with a max number of columns specified in parenthesis
- date: Date value
- number(size,d): A number with a maximum number of digits of "size" and a maximum number of "d" digits to the right of the decimal

3. INSERT VALUES

Once a table has been created data can be inserted using INSERT INTO command.

INSERT INTO *tablename*
(col1, ... , coln)
VALUES (val1, ... , valn)

4. UPDATE
To change the data values in a pre existing table, the UPDATE command can be used.

UPDATE *tablename*

SET *colX = valX* [, *colY = valY*, ...]
WHERE *condition*

5. DELETE
The DELETE command can be used to remove a record(s) from a table.

DELETE FROM *tablename*
WHERE *condition*

To delete all the records from a table without deleting the table do

DELETE * FROM *tablename*

6. DROP
To remove an entire table from the database use the DROP command.

DROP TABLE *tablename*

7. ORDER BY
ORDER BY clause can order column name in either ascending (ASC) or descending (DESC) order.

ORDER BY *col_name* ASC

8. AND / OR
AND and OR can join two or more conditions in a WHERE clause. AND will return data when all the conditions are true. OR will return data when any one of the conditions is true.

9. IN
IN operator is used when you know the exact value you want to return for at least one of the columns

SELECT * FROM *table_name* WHERE *col_name* IN (*val1*, *val2*, ...)

10. BETWEEN / AND
The BETWEEN ... AND operator selects a range of data between two values. These values can be numbers, text, or dates.

SELECT * FROM *table_name* WHERE *col_name* BETWEEN *val1* AND *val2*

Implementation:
//PROGRAM FOR BASIC QUERY

CREATE TABLE QUERY: (DDL)

create table teit(rollno int,name char(30),percentage int)

output:

Table created.

INSERT QUERY: (DML)

insert into teit values(7,'POOJA',65)

OUTPUT:

1 row(s) inserted.

SELECT QUERY: (DML)

select*from teit

OUTPUT:

ROLLNO	NAME	PERCENTAGE
7	POOJA	65

ALTER QUERY: for adding column

alter table teit add (age int)

OUTPUT:

Table altered.

ROLLNO	NAME	PERCENTAGE	AGE
7	POOJA	65	-
10	ANUJA	62	-
24	RUTUJA	63	-

UPDATE QUERY: (DML)

update teit set age=23 where rollno=7

1 row(s) updated.

ROLLNO	NAME	PERCENTAGE	AGE
7	POOJA	65	23
10	ANUJA	62	21
24	RUTUJA	63	21

RENAME QUERY: (DDL)

Rename column name:-

select name as stname from teit

OUTPUT:

STNAME
POOJA
ANUJA
RUTUJA

Rename table name:-

rename teit to studentdb

OUTPUT:

Statement processed.

ALTER QUERY: for deleting column

alter table studentdb drop column age

OUTPUT:

Table dropped.

ROLLNO	NAME	PERCENTAGE
7	POOJA	65
10	ANUJA	62
24	RUTUJA	63

TRUNCATE QUERY:

truncate table studentdb

OUTPUT:

Table truncated.

DROP QUERY:

drop table studentdb

OUTPUT:

Table dropped.

Problem: Perform DDL & DML queries for Bank management system

Conclusion:
We have studied the implementation of DDL & DML commands for SQL.

Experiment No. 6

Title : Implementation of SQL Query constraints with referential integrity constraints.
Aim : To implement SQL Query constraints with referential integrity constraints

Theory :

- **PRIMARY KEY**: A PRIMARY KEY constraint for a table enforces the table to accept unique data for a specific column and this constraint create a unique index for accessing the table faster
- **UNIQUE**: The UNIQUE constraint in Mysql does not allow to insert a duplicate value in a column.
- **NOT NULL**: In Mysql NOT NULL constraint allows to specify that a column can not contain any NULL value.
- **FOREIGN KEY**: A FOREIGN KEY in mysql creates a link between two tables by one specific column of both table. The specified column in one table must be a PRIMARY KEY and referred by the column of another table known as FOREIGN KEY.
- **CHECK**: The CHECK constraint determines whether the value is valid or not from a logical expression.
- **DEFAULT**: While inserting data into a table, if no value is supplied to a column, then the column gets the value set as DEFAULT

Implementation:

❖ Primary Key Constraint

- **Column Level**

```
create table cust(c_name char(20),c_id int primary key,c_add char(25))

insert into cust values('Mrunali',3,'Vaduj')

select *from cust
```

C_NAME	C_ID	C_ADD
Mrunali	3	Vaduj

Table Level:

```
create table cust1(c_name char(20),c_id int,c_add char(25),primary key(c_id,c_name))

insert into cust1 values('Sheetal',3,'Vaduj')

select *from cust1
```

C_NAME	C_ID	C_ADD
Sheetal	3	Vaduj

❖ Foreign Key

- **Column Level**

```
create table emp(c_name char(20),c_id int references cust,e_add char(25))
```

insert into emp values('Sheetal',3,'RAVIVAR PETH')

select *from emp

C_NAME	C_ID	E_ADD
Sheetal	3	RAVIVAR PETH

- **Table Level**

create table emp1(c_name char(20),c_id int,e_add char(25),foreign key(c_name,c_id)references cust1(c_name,c_id))

insert into emp1 values('Sheetal',3,'RAVIVAR PETH')

insert into emp1 values('Sheetal',3,'karve Road')

select *from emp1

C_NAME	C_ID	E_ADD
Sheetal	3	RAVIVAR PETH
Sheetal	3	karve Road

❖ **Unique Key**

- **Column Level**

create table acc1(acc_no int unique,branch_name char(10),bal int)

insert into acc1 values(null,'karad',2000)

select *from acc1

ACC_NO	BRANCH_NAME	BAL
-	karad	2000

- **Table Level**

create table acc2(acc_no int,branch_name char(10),bal int,unique(acc_no,branch_name)

insert into acc2 values(null,'karad',2000)

insert into acc2 values(3332,'karad',2200)

select *from acc2

ACC_NO	BRANCH_NAME	BAL
-	karad	2000
3332	karad	2200

❖ Null Value Concept

create table Branch(B_no varchar(10),B_name varchar(25))

insert into Branch(B_no,B_name)values('B1',null)

insert into Branch(B_no,B_name)values('B2','')

select *from Branch where B_name=''

select *from Branch where B_name is null

B_NO	B_NAME
B1	-
B1	-
B2	-

❖ Not Null Value Concept

create table account(acc_no int not null,branch_name char(10),bal int)

insert into account values(3332,'karad',2000)

insert into account values(4567,'satara',3000)

select *from account

ACC_NO	BRANCH_NAME	BAL
3332	karad	2000
4567	satara	3000

Check Constraint

- **Column Level-**

create table act(acc_no int,branch_name char(10)check(branch_name=upper(branch_name)),bal int)

insert into act values(3332,'KARAD',2000)

insert into act values(4567,'SATARA',5000)

select *from act

ACC_NO	BRANCH_NAME	BAL
3332	KARAD	2000
4567	SATARA	5000

- **Table Level-**

create table abc(acc_no int,branch_name char(10),bal int,c_name char(30),check(branch_name=upper(branch_name)),check(c_name=upper(c_name)))

insert into abc values(2222,'KARAD',1000,'ABC')

insert into abc values(3232,'SATARA',5000,'MABC')

select *from abc

ACC_NO	BRANCH_NAME	BAL	C_NAME
2222	KARAD	1000	ABC
3232	SATARA	5000	MABC

- **ON DELETE CASCADE**

create t able customer(cust_id int primary key,cust_name char(15),cust_addr char(20))

Table created.
insert into customer values(1,'sona','karad')

1 row(s) inserted.

insert into customer values(2,'mona','karad')

1 row(s) inserted.
insert into customer values(3,'anu','satara')

1 row(s) inserted.
insert into customer values(4,'ravi','pune')

1 row(s) inserted.
insert into customer values(5,'mahi','sangli’)

1 row(s) inserted.
select *from customer

CUST_ID	CUST_NAME	CUST_ADDR
1	sona	karad
2	mona	karad
3	anu	satara
4	ravi	pune
5	mahi	sangli

create table cust(c_id int references customer(cust_id)ON DELETE CASCADE,c_name char(20),c_add char(30)

insert into cust values(1,'sona','karad')

insert into cust values(2,'sham','nagpur')

insert into cust values(3,'sam','solapur')

insert into cust values(4,'naina','kolhapur')

select *from cust

C_ID	C_NAME	C_ADD
1	sona	karad
2	sham	nagpur
3	sam	solapur
4	naina	kolhapur

delete from customer where cust_id=3

select *from customer

CUST_ID	CUST_NAME	CUST_ADDR
1	sona	karad
2	mona	karad
4	ravi	pune
5	mahi	sangli

select *from cust

C_ID	C_NAME	C_ADD
1	sona	karad
2	sham	nagpur
4	naina	kolhapur

- **ON DELETE SET NULL**

create table cust2(c_id int ,c_name char(20),c_add char(30),foreign key(c_id)references customer(cust_id)on delete set null)

Table created.
insert into cust2 values(2,'sham','nagpur')

1 row(s) inserted.
insert into cust2 values(4,'naina','kolhapur')
1 row(s) inserted.
select *from cust2

C_ID	C_NAME	C_ADD
2	sham	nagpur
4	naina	kolhapur

delete from cust2 where c_id=2
1 row(s) deleted.
select *from cust2

C_ID	C_NAME	C_ADD
4	naina	kolhapur

Problem statement: Implement all constraint to create database for college management System.

Conclusion:
Thus we have studied and implemented all the SQL Query constraints with referential integrity constraints

Experiment No. 7

Title: Implementation of SQL Query Processing: program to show use of SQL Clauses-Order by, group by and between clauses.

Aim: To implement SQL Query Processing: program to show use of SQL Clauses-Order by, group by and between clauses.

Theory:

- Write the syntax in SQL for following clauses: Order by, group by, having and between clauses

Implementation:

select * from dari

STUD_ROLLNO	STUD_NAME	ADDRESS
1	Daryappa	jaysingpur
2	pravin	pandarpur
3	Ganesh	phalthan

select * from sdev

ROLLNO	NAME	ID
12	abc	430
8	abc	220
9	amg	520
2	ass	250

Order By :-

1. Ascending

select rollno from sdev order by rollno

ROLLNO
2
8
9
12

2. Descending

select rollno from sdev order by rollno desc

ROLLNO
12
9
8
2

Group By :- having

select name, count(rollno)from sdev group by name;

NAME	COUNT(ROLLNO)
amg	1
abc	2

Between Clause:

Select rollno from sdev where ID between 200 and 400;

Problem statement:

1) Write a program which uses following SQL Commands with check constraint for employee payroll

a. Between
b. Distinct
c. Rollup

2) Write a program which uses following SQL Commands for employee payroll

a. Group by (using Having clause)

b. Order by

Conclusion:

We have studied and implemented the SQL Clauses-Order by, group by and between.

Experiment No. 8

Title : Implementation of SQL Query Aggregate functions & set operations
Aim : To Implement SQL Query Aggregate functions & set operations
Theory :
These functions are used to manipulating data items and returning the results.
1. Group functions or Aggregate functions.
2. Single Row or scalar function.
Group functions or Aggregate functions:
These functions operated a set of values or rows
i. Sum()
ii. Avg()
iii. Min()
iv. Max()
v. Count()

- Write down the syntax for at least 7 Aggregate functions & set operations

Implementation:

1. Create table

```
create table beit(roll_no int,name char(20),ph_no int)
```

ROLL_NO	NAME	PH_NO
15	vishwambhar	8805705733
30	ahwin	8805161129
40	jaydip	9561239599
55	ajinkya	96596235

2. AVG

```
select avg(roll_no)"Roll_no Avg" from beit
```

Roll_no Avg
35

3. MIN

```
select min(roll_no)"Roll_no Avg" from beit
```

Roll_no Avg
15

4. MAX

select max(roll_no)"Max Roll_no" from beit

Max Roll_no
55

5. COUNT

select count(ph_no)"Count Ph_no" from beit

Count Ph_no
4

6. SUM

select sum(ph_no)"SUM Ph_No" from beit

SUM Ph_No
27268702696

7. ABS

select abs(ph_no)"ABS Ph_No" from beit

ABS Ph_No
8805705733
8805161129
9561239599
96596235

8. POWER

select power(7,3)"POWER" from beit

POWER
343
343
343

9. ROUND

select round(34.56375,3)"ROUND" from beit

ROUND
34.564
34.564
34.564
34.564

10. SQRT

select sqrt(25)"ROOT" from beit

ROOT
5
5
5
5

11. EXP(n)

select exp(5)"EXPONENT" from beit

EXPONENT
148.413159102576603421115580040552279624
148.413159102576603421115580040552279624
148.413159102576603421115580040552279624
148.413159102576603421115580040552279624

12. GREATEST

select greatest(56,25)"GREATEST" from beit

GREATEST
56
56
56
56

13. LEAST

select least(56,25)"LEAST" from beit

LEAST
25
25
25
25

14. MOD(m,n)

select mod(56,5)"MOD" from beit

MOD
1
1
1
1

15. TRUNC

select trunc(56.53625,3)"TRUNC" from beit

TRUNC
56.536
56.536
56.536
56.536

16. FLOOR

select floor(56.53625)"floor" from beit

Floor
56
56
56
56

- **Set operations:**
  - **Queries:-**
  - select * from dari

STUD_ROLLNO	STUD_NAME	ADDRESS
1	Daryappa	jaysingpur
2	pravin	pandarpur
3	Ganesh	phalthan

- select * from sdev

ROLLNO	NAME	ID
12	abc	430
8	abc	220
9	amg	520
2	ass	250

Set operator

1. Union: If a={1,2,3}; b={2,4} then aUb: {1,2,3,4}

select rollno from sdev **union** select stud_rollno from dari

ROLLNO
1
2
3
8
9
12

2. Intersect: If a={1,2,3}; b={2,4} then a∩b: {2}

select rollno from sdev **intersect** select stud_rollno from dari

ROLLNO
2

3. Minus: (Set difference) If a={1,2,3}; b={2,4} then a-b: {1,3} & b-a: {4}

select rollno from sdev **minus** select stud_rollno from dari

ROLLNO
8
9
12

Problem statement: Create any database with application of all aggregate functionsand set operations

Conclusion:
We have studied and implemented all aggregate functions and set operations in the SQL database

Experiment No. 9

Title : Implementation of SQL query Join operations.

Aim : To implement SQL query Join operations

Theory/ Design :

- Explain the concept of join and its types in detail

MySQL JOINS are used to retrieve data from multiple tables. A MySQL JOIN is performed whenever two or more tables are joined in a SQL statement. There are different types of MySQL joins:

- MySQL INNER JOIN (or sometimes called simple join)
- MySQL NATURAL JOIN
- MySQL CROSS JOIN
- MySQL FULL OUTER JOIN
- MySQL LEFT OUTER JOIN (or sometimes called LEFT JOIN)
- MySQL RIGHT OUTER JOIN (or sometimes called RIGHT JOIN)

INNER JOIN (simple join) MySQL INNER JOINS return all rows from multiple tables where the join condition is met. Syntax:

SELECT columns From table1 Inner join table2 On table1.column=table2.column;

LEFT OUTER JOIN Another type of join is called a MySQL LEFT OUTER JOIN. This type of join returns all rows from the LEFT-hand table specified in the ON condition and only those rows from the other table where the joined fields are equal.

Syntax:

SELECT columns From table1 left join table2 On table1.column=table2.column;

RIGHT OUTER JOIN Another type of join is called a MySQL RIGHT OUTER JOIN. This type of join returns all rows from the RIGHT-hand table specified in the ON condition and only those rows from the other table where the joined fields are equal.

Syntax: SELECT columns From table1 Right join table2 On table1.column=table2.column;

Implementation:

Queries:-

select * from stud

STUD_ROLLNO	STUD_NAME	ADDRESS
1	Kishor	thane
2	rohit	dadar
3	Siddhant	afrika

select * from cust1

CUST_ID	CUST_NAME	ADDRESS
1	bhavesh	Kalyan
2	amol	karad
3	Kishor	thane

Inner Join

select * from stud inner join cust1 on stud.stud_name=cust1.cust_name

STUD_ROLLNO	STUD_NAME	ADDRESS	CUST_ID	CUST_NAME	ADDRESS
1	Kishor	thane	3	Kishor	thane

Natural Join

select * from stud natural join cust1

ADDRESS	STUD_ROLLNO	STUD_NAME	CUST_ID	CUST_NAME
thane	1	Kishor	3	Kishor

Cross Join

select * from stud cross join cust1

STUD_ROLLNO	STUD_NAME	ADDRESS	CUST_ID	CUST_NAME	ADDRESS
1	Kishor	thane	1	bhavesh	Kalyan
1	Kishor	thane	2	amol	karad
1	Kishor	thane	3	Kishor	thane
2	rohit	dadar	1	bhavesh	Kalyan
2	rohit	dadar	2	amol	karad
2	rohit	dadar	3	Kishor	thane
3	Siddhant	afrika	1	bhavesh	Kalyan
3	Siddhant	afrika	2	amol	karad
3	Siddhant	afrika	3	Kishor	thane

Outer Join

1. **Full outer join**

select * from stud full outer join cust1 on stud.stud_name=cust1.cust_name

STUD_ROLLNO	STUD_NAME	ADDRESS	CUST_ID	CUST_NAME	ADDRESS
1	Kishor	thane	3	Kishor	thane
3	Siddhant	afrika	-	-	-
2	rohit	dadar	-	-	-
-	-	-	2	amol	karad
-	-	-	1	bhavesh	Kalyan

2. Right outer join

select * from stud right outer join cust1 on stud.stud_name=cust1.cust_name

STUD_ROLLNO	STUD_NAME	ADDRESS	CUST_ID	CUST_NAME	ADDRESS
1	Kishor	thane	3	Kishor	thane
-	-	-	2	amol	karad
-	-	-	1	bhavesh	Kalyan

3. Left outer join

select * from stud left outer join cust1 on stud.stud_name=cust1.cust_name

STUD_ROLLNO	STUD_NAME	ADDRESS	CUST_ID	CUST_NAME	ADDRESS
1	Kishor	thane	3	Kishor	thane
3	Siddhant	afrika	-	-	-
2	rohit	dadar	-	-	-

Problem statement: Implement all joins for University management system.

Conclusion:
We have studied and implemented all SQL query Join operations

Experiment No. 10

Title : Implementation of String operations in SQL.

Aim : To implement String operations in SQL

Theory/ Design:

- Write the syntax for all string related operations

Queries using Conversion functions (to_char, to_number and to_date), string functions (Concatenation, lpad, rpad, ltrim, rtrim, lower, upper, initcap, length, substr and instr), date functions (Sysdate, next_day, add_months, last_day, months_between, least, greatest, trunc, round, to_char, to_date)

1. Conversion functions:

To_char: TO_CHAR (number) converts n to a value of VARCHAR2 data type, using the optional number format fmt. The value n can be of type NUMBER, BINARY_FLOAT, or BINARY_DOUBLE.

SQL>select to_char(65,'RN')from dual;
LXV

To_number : TO_NUMBER converts expr to a value of NUMBER data type.

SQL> Select to_number('1234.64') from Dual;
1234.64

To_date: TO_DATE converts char of CHAR, VARCHAR2, NCHAR, or NVARCHAR2 data type to a value of DATE data type.

SQL>SELECT TO_DATE('January 15, 1989, 11:00 A.M.') FROM DUAL;
TO_DATE('
- - - - - -
15-JAN-89

2. String functions:

Concat: CONCAT returns char1 concatenated with char2. Both char1 and char2 can be any of the datatypes

SQL>SELECT CONCAT(‘ORACLE’,’CORPORATION’)FROM DUAL;
ORACLECORPORATION

Lpad: LPAD returns expr1, left-padded to length n characters with the sequence of characters in expr2.

SQL>SELECT LPAD(‘ORACLE’,15,’*’)FROM DUAL;
*********ORACLE

Rpad: RPAD returns expr1, right-padded to length n characters with expr2, replicated as many times as necessary.

SQL>SELECT RPAD (‘ORACLE’,15,’*’)FROM DUAL;
ORACLE*********

Ltrim: Returns a character expression after removing leading blanks.

SQL>SELECT LTRIM(‘SSMITHSS’,’S’) FROM DUAL;
MITHSS

Rtrim: Returns a character string after truncating all trailing blanks

SQL>SELECT RTRIM(‘SSMITHSS’,’S’) FROM DUAL;
SMITH

Lower: Returns a character expression after converting uppercase character data to

lowercase.
SQL>SELECT LOWER(‘DBMS’)FROM DUAL;
dbms
Upper: Returns a character expression with lowercase character data converted to uppercase
SQL>SELECT UPPER(‘dbms’)FROM DUAL;
DBMS
Length: Returns the number of characters, rather than the number of bytes, of the given string expression, excluding trailing blanks.
SQL>SELECT LENGTH(‘DATABASE’)FROM DUAL;
8
Substr: Returns part of a character, binary, text, or image expression.
SQL>SELECT SUBSTR(‘ABCDEFGHIJ’3,4)FROM DUAL;
CDEF
Instr: The INSTR functions search string for substring. The function returns an integer indicating the position of the character in string that is the first character of this occurrence.
SQL>SELECT INSTR('CORPORATE FLOOR','OR',3,2)FROM DUAL;
14
3. Date functions:
Sysdate:
SQL>SELECT SYSDATE FROM DUAL;
29-DEC-08
next_day:
SQL>SELECT NEXT_DAY(SYSDATE,’WED’)FROM DUAL;
05-JAN-09
add_months:
SQL>SELECT ADD_MONTHS(SYSDATE,2)FROM DUAL;
28-FEB-09
last_day:
SQL>SELECT LAST_DAY(SYSDATE)FROM DUAL;
31-DEC-08
months_between:
SQL>SELECT MONTHS_BETWEEN(SYSDATE,HIREDATE)FROM EMP;
4
Least:
SQL>SELECT LEAST('10-JAN-07','12-OCT-07')FROM DUAL;
10-JAN-07
Greatest:
SQL>SELECT GREATEST('10-JAN-07','12-OCT-07')FROM DUAL;
10-JAN-07
Trunc:
SQL>SELECT TRUNC(SYSDATE,'DAY')FROM DUAL;
28-DEC-08
Round:
SQL>SELECT ROUND(SYSDATE,'DAY')FROM DUAL;
28-DEC-08

to_char:
SQL> select to_char(sysdate, "dd\mm\yy") from
dual;
24-mar-05.
to_date:
SQL> select to_date(sysdate, "dd\mm\yy") from dual;
24-mar-o5.

Implementation:
select lower ('kishor')"lower" from dual

lower
kishor

select upper ('kishor')"capittalised" from dual

capittalised
KISHOR

select substr('sekishor',3,6)"substring" from dual

substring
kishor

select translate('1 set523','143','7ag')"change" from dual

change
7 set52g

select initcap('DACOE')"titlecase" from dual

titlecase
Dacoe

select ASCII('DACOE')"ASCII" from dual

ASCII
68

select length('DACOE')"length" from dual

length
5

select lpad('DACOE',9,'*')"lpad" from dual

lpad
****DACOE

select rpad('DACOE',9,'*')"rpad" from dual

rpad
DACOE****

select ltrim('DACOE','D')"ltrim" from dual

ltrim
ACOE

select rtrim('DACOE','E')"rtrim" from dual

rtrim
DACO

select concat('RA','-ONE')"concate" from dual

concate
RA-ONE

Problem statement: Create database of employee payroll system and implement all string and date functions on the database.

Conclusion: We have studied and implemented all string and date functions in the SQL database

Experiment No. 11

Title : Implementation of Views for any created table

Aim : To understand and implement views in SQL table

Theory/ Design:

A view is the tailored presentation of data contained in one or more table and can also be said as restricted view to the data in the tables. A view is a "virtual table" or a "stored query" which takes the output of a query and treats it as a table. The table upon which a view is created is called as base table. A view is a logical table based on a table or another view. A view contains no data of its own but is like a window through which data from tables can be viewed or changed. The tables on which a view is based are called base tables. The view is stored as a SELECT statement in the data dictionary.

Advantages of a view:

a. Additional level of table security.
b. Hides data complexity.
c. Simplifies the usage by combining multiple tables into a single table

Creating and dropping view:

Syntax:

Create or replace view view_name AS SELECT column_name(s) FROM table_name
WHERE condition;
Drop view <view name>;

Example:

create table tv9(mno varchar2(50),brand varchar2(50),price varchar2(50));
Table created.

insert into tv9 values('ms78','samsung','10000');
1 row(s) inserted.

insert into tv9 values('89ru','lg','34000');
1 row(s) inserted.

insert into tv9 values('nve9','sharp','12000');
1 row(s) inserted.

insert into tv9 values('rusk9','onida','17000');
1 row(s) inserted.

select * from tv9

MNO	BRAND	PRICE
ms78	samsung	10000
89ru	lg	34000
nve9	sharp	12000
rusk9	onida	17000

4 rows returned in 0.05 seconds

create view tv9view AS
SELECT brand,price
FROM tv9
WHERE brand IS NOT NULL
AND price IS NOT NULL;

View created.

select*from tv9view

BRAND	PRICE
samsung	10000
lg	34000
sharp	12000
onida	17000

4 rows returned in 0.01 secor

- **Insert:**

INSERT INTO tv9view
(brand,price)
VALUES
('toshiba','8000');

1 row(s) inserted.
select*from tv9view

BRAND	PRICE
samsung	10000
lg	34000
sharp	12000
onida	17000
toshiba	8000

5 rows returned in 0.00 seconds

select * from tv9

MNO	BRAND	PRICE
ms78	samsung	10000
89ru	lg	34000
nve9	sharp	12000
rusk9	onida	17000
-	toshiba	8000

5 rows returned in 0.00 seconds

- **Update**

UPDATE tv9view
SET price = price + 10
where brand='toshiba'

1 row(s) updated.

select*from tv9view

BRAND	PRICE
samsung	10000
lg	34000
sharp	12000
onida	17000
toshiba	8010

5 rows returned in 0.00 seconds

select * from tv9

MNO	BRAND	PRICE
ms78	samsung	10000
89ru	lg	34000
nve9	sharp	12000
rusk9	onida	17000
-	toshiba	8010

5 rows returned in 0.01 seconds

- **Delete**

delete from tv9view where brand='toshiba'

1 row(s) deleted.

select*from tv9view

BRAND	PRICE
samsung	10000
lg	34000
sharp	12000
onida	17000

4 rows returned in 0.00 seconds

Problem statement:

Write the queries for the following

(i) Create a view emp from employee such that it contains only emp_no and emp_name and department.
(ii) Create a view dept from department with only dept_no and location.
(iii) Create a view that contains the details of employees who are managers only.
(iv) drop the views.

Conclusion: We have studied and implemented views with basic operations in the SQL database

Experiment No. 12

Title : Implementation of Hashing Technique on the created data.

Aim : To study and implement Hashing technique.

Theory:

Hashing is one of the way to store the Data into database. In the Hashing one hash function is there that will find address of the bucket where we have to store the particular record after calculation on the particular search key.
There are two types:

1) Static Hashing.

2) Dynamic Hashing.

1) Static Hashing:

In hash file organization we obtain the address of the disk block containing the desired record directly by computing a function on the search key value of record.

Let K denotes the set of all search key values & B denotes the set of all bucket address. A hash function h is a function from K to B Let h denotes a hash function.

To insert a record with search key K we compute h(K:) which gives the address of the bucket for that record.

Hash Functions:

Hash functions maps all search key values to the same bucket Ideal hash function distributes the stored key uniformly across all the buckets so that every bucket has some no. of records.

Handling of bucket overflows:

If the bucket does not have enough space a bucket overflow is said to occur bucket overflow can occur for several reasons.

a) Insufficient Buckets:

The no. of buckets which we denote must be chosen such that n b>h r|f r where nr denotes the total will be stored & f r denotes the no. of records that will fit in a bucket.

b) Skew:

Some buckets are assigns more records than are others so a bucket may overflow even when other bucket still have specials situation is called bucket skew.

c) Hash Indices:

A hash index organizes the search keys their associated pointers into a hash file structure.

2) Dynamic Hashing :

Several dynamic hashing techniques allows the hash functions to be modified dynamically to accurate the growth or shrinkage of database bucket address table.

Hashing is having following advantages-:

1) File Organization is based on the technique of hashing allow us to avoid accessing the index structure.
2) Hashing Provides a way to construct the index.
3) Here we obtain the address by calculating the hash function on the record using any desirable formula.
4) We can insert records using this hash index search key.
5) To insert we follow same procedure as of b –trees.
6) To Delete a record we also do look up and then searching it is to be deleted form file.
7) Store all The records in a file and store their indices block wise.

Example:

```
// CPP Program For Hashing

#include<iostream.h>

#include<fstream.h>

#include<conio.h>

#include<stdio.h>

#include<string.h>

class hash

{

public:

int lo_no,amt;

char name[20];

void get()

{

cout<<"\n Enter the name:";

cin>>name;

cout<<"\n Enter the loan no:";

cin>>lo_no;
```

```
cout<<"\n Enter the amount:";
cin>>amt;
}
void show()
{
cout<<"\n\n\n Name\tloan no\t Amount";
cout<<name<<"\t"<<lo_no<<"\t"<<amt<<endl;
}
}m,n;
int hash_fn(int);
void main()
{
fstream fp,fp1;
hash m,n;
int no,c;
char ch[50]="hash1";
clrscr();
cout<<"\n1:Enter the Record \n2:Serach the Record";
cout<<"\n Enter the choose:";
cin>>c;
if(c==1)
{
cout<<endl;
m.get();
n=m;
no=hash_fn(m.lo_no);
if(no==0)
{
```

```
strcat(ch,"0.txt");
}
if(no==1)
{
strcat(ch,"1.txt");
}
if(no==2)
{
strcat(ch,"2.txt");
}
if(no==3)
{
strcat(ch,"3.txt");
}  if(no==4)
{
strcat(ch,"4.txt");
}
if(no==5)
{
strcat(ch,"5.txt");
}
fp.open(ch,ios::app);
fp.write((char*)&n,sizeof(n));
fp.close();
}
if(c==2)
{
cout<<"Enter Laon no for searching:";
```

```
cin>>m.lo_no;
n.lo_no=m.lo_no;
no=hash_fn(m.lo_no);
if(no==0)
{
strcat(ch,"0.txt");
}
if(no==1)
{
strcat(ch,"1.txt");
}
if(no==2)
{
strcat(ch,"2.txt");
}
if(no==3)
{
strcat(ch,"3.txt");
}
if(no==4)
{
strcat(ch,"4.txt");
}
if(no==5)
{
strcat(ch,"5.txt");
}
fp1.open(ch,ios::in);
```

```
if(fp!=NULL)
{
while(fp1.read((char*)&m,sizeof(m)))
{
fp1.read((char*)&m,sizeof(m));
cout<<"in loop";
if(n.lo_no==m.lo_no)
{
m.show();
}}
fp.close();
}
else
{
cout<<"file not found";
}}
getch();
}
hash_fn(int no)
{
int k;
k=no%5;
return(k);
}
/*OUTPUT */
1:Enter the Record
2:Serach the Record
 Enter the choose:1
```

Enter the name: rohit

Enter the loan no:11

Enter the amount:500

1:Enter the Record

2:Serach the Record

Enter the choose:2

Enter Laon no for searching:11

in loop

Name loan no Amount rohit 11 500

Problem statement: Write a java program to implement the hashing technique

Conclusion:

Thus we can insert & display data after using Hash indices by calculating Hash on the Records

Questions:

1. **What is Indexing and B+ tree index file?**
2. **What is Static Hash Functions?**
3. **What is Dynamic Hash Functions?**
4. **What is difference between Indexing and Hashing?**

Experiment No. 13

Title : Study of NoSQL

Aim : To study and implement NoSQL

Theory/Design:

- Explain NoSQL database in detail.
  Explain features of NoSQL database
- A comparison of different NoSQL databases (w.r.t. Database Tool, Data model, Transaction model, Query support)

- **Key-value store:** In these databases all the stored data is represented by a pair of key and value per record, meaning that each key is unique and it allows accessing record's This structure is also known as "hash table" where data retrieval is usually performed by using key to access value.
- **Document Store.** These databases are designed to manage data stored in documents that use different format standards, such as XML or JSON. This type of storage is more complex in comparison to storage used by Key-value Stores and enables data querying.
- **Column Family.** The database structure of this NoSQL type is similar to the standard Relational Database Management System (RDMS) since all the data is stored as sets of columns and rows. Columns, that store related data that is often retrieved together, may be grouped.
- **Graph Database.** These databases are mostly used when the stored data may be represented as a graph with interlinked elements such as, social networking, road maps or transport routes.

Conclusion: We have studied and understand the NoSQL databases and its various formats

Laboratory Manual
2023-24

Sub : Operating System-II

TY B. Tech. (CSE)

Prepared By,

Prof. Kakade S. P.

G.K. Gujar Memorial Charitable Trust's
DR.ASHOK GUJAR TECHNICAL INSTITUTE'S
DR. DAULATRAO AHER COLLEGE OF ENGINEERING,KARAD
Vidyanagar Extn. Banawadi, Dist. Satara

Prepared by :Mrs. S. P. Kakade	Approved by: Head of Department

G.K. Gujar Memorial Charitable Trust's

DR.ASHOK GUJAR TECHNICAL INSTITUTE'S
DR. DAULATRAO AHER COLLEGE OF ENGINEERING, KARAD

Lab Manual

Subject : Operating System-II

TITLE : Index (List of Experiment)

Sr.No	Name of Experiment
01	Study of UNIX Operating System
02	Implementation of general , directory and file utilities
03	Implementation of Scheduling Algorithms.
04	Study and implementation of building block primitives like redirect I/O and pipe
05	Study & Implementation of process related utilities.
06	Study of file security and text manipulation utilities
07	Write programs using the I/O system calls of UNIX operating system (open, read, write, close etc)
08	Study & implementation of Shell programming
09	Study of system startup & init
10	Study & implementation of utilities for the advanced system administrator Like useradd, passwd, df, du, find, fdisk, mkfs
11	Study of Memory allocation algorithm (best-fit, first-fit, worst-fit).

Prepared by :Mrs. S. P. Kakade	Approved by: Head of Department

G.K. Gujar Memorial Charitable Trust's

DR.ASHOK GUJAR TECHNICAL INSTITUTE'S
DR. DAULATRAO AHER COLLEGE OF ENGINEERING, KARAD

Lab Manual

Subject : Operating System-II

Expt. No: 01	**TITLE** : Study of UNIX Operating System

Aim: To study UNIX Operating System.

Theory:

Features of UNIX Operating System-

- The system is written in a high-level language, making it easy to read, understand, change, and move to other machines. Ritchie estimates that the first system in C was 20 to 40 percent larger and slower because it was not written in assembly language, but the advantages of using a higher-level language far outweigh the disadvantages.
- It has a simple user interface that has the power to provide the services that users want.
- It provides primitives that permit complex programs to be built from simpler programs.
- It uses a hierarchical file system that allows easy maintenance and efficient implementation.
- It uses a consistent format for files, the byte stream, making application programs easier to write.
- It provides a simple, consistent interface to peripheral devices.
- It is a multi-user, multiprocess system; each user can execute several processes simultaneously.
- It hides the machine architecture from the user, making it easier to write programs that run on different hardware implementations.

SYSTEM STRUCTURE

Figure depicts the high-level architecture of the UNIX system. The hardware at the center of the diagram provides the operating system with basic services. The operating system interacts directly with the hardware, providing common services to programs and insulating them from hardware idiosyncrasies. Viewing the system as a set of layers, the operating system is commonly called the system kernel, or just the kernel, emphasizing its isolation from user programs. Because programs are independent of the underlying hardware, it is easy to move them between UNIX systems running on different hardware if the programs do not make assumptions about the underlying hardware. Programs such as the shell and editors (ed and vi) shown in the outer layers interact with the kernel by invoking a well defined set of system calls. The system calls instruct the kernel to do various operations for the calling program and exchange data between the kernel and the program. Several programs shown in the figure are in standard system configurations and are known as commands, but private user programs may also exist in this layer as indicated by the program whose name is a.out, the standard name for executable files produced by the C compiler. Other application programs can build on top of lower-level programs, hence the existence of the outermost layer in the figure. For example, the standard C compiler, cc, is in the outermost layer of the figure: it invokes a C preprocessor, two-pass compiler, assembler, and loader Oink-editor), all separate lower-level programs. Although

the figure depicts a two-level hierarchy of application programs, users can extend the hierarchy to whatever levels are appropriate.
Many application subsystems and programs that provide a high-level view of the system such as the shell, editors, sees (Source Code Control System}, and document preparation packages, have gradually become synonymous with the name UNIX system."

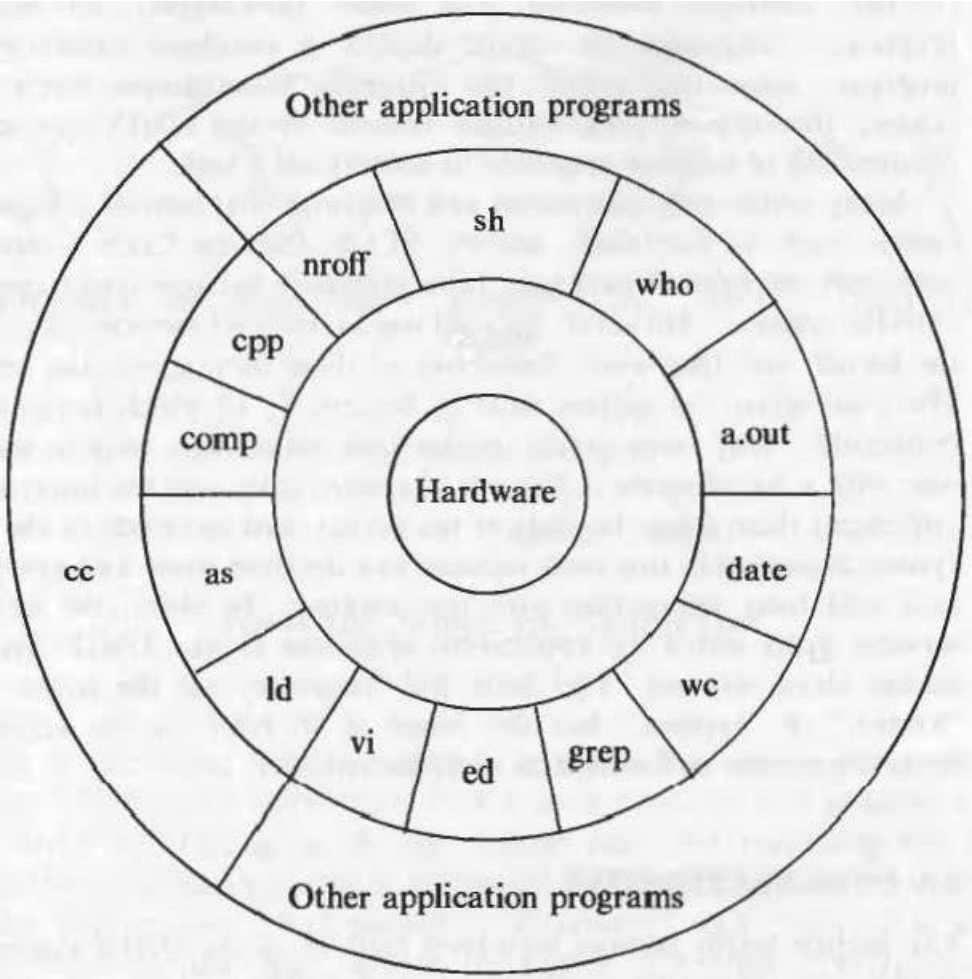


Fig. 1.1 Architecture of UNIX Systems

UNIX File System
The UNIX file system is characterized by,
• a hierarchical structure,
• consistent treatment of file data,
• the ability to create and delete files,
• dynamic growth of files,
• the protection of file data,
• the treatment of peripheral devices (such as terminals and tape units) as files.

The file system is organized as a tree with a single root node called root (written .. /"); every non-leaf node of the file system structure is a directory of files, and files at the leaf nodes of the tree

are either directories, regular files, or special device files. The name of a file is given by a path name that describes how to locate the file in the file system hierarchy. A path name is a sequence of component names separated by slash characters; a component is a sequence of characters that designates a file name that is uniquely contained in the previous (directory) component. A full path name starts with a slash character and specifies a file that can be found by starting at the file system root and traversing the file tree, following the branches that lead to successive component names of the path name.

Thus, the path names .. /etc/passwd", .. /bin/who", and .. /usr/src/cmd/who.c" designate files in the tree shown in Figure below, but .. /bin/passwd" and "/usr/src/date.c" do not. A path name does not have to :start from root but can be designated relative to the current directory of an executing process, by omitting the initial slash in the path name. Thus, starting from directory .. /dev", the path name "ttyOl" designates the file whose fuU path name is .. /dev/ttyOl". Directories are like regular files in this respect; the system treats the data in a directory as a byte stream, but the data contains the names of the files in the directory in a predictable format so that the operating system and programs such as ls (list the names and attributes of files) can discover the files in a directory.

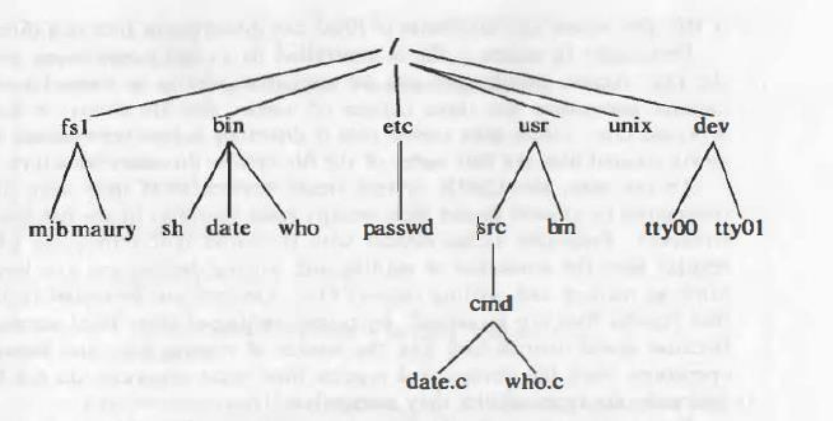


Fig. 1.2 Sample File System Tree

ARCHITECTURE OF THE UNIX OPERATING SYSTEM

Figure 1.3 gives a block diagram of the kernel, showing various modules and their relationships to each other. In particular, it shows the file subsystem on the left and the process control subsystem on the right, the two major components of the kernel. The diagram serves as a useful logical view of the kernel, although in practice the kernel deviates from the model because some modules interact with the internal operations of others. Figure 1.3 shows three levels: user, kernel. and hardware. The system call and library interface represent the border between user programs and the kernel depicted in Figure 1.1. System calls look like ordinary function calls in C programs, and libraries map these function calls to the primitives needed to enter Assembly language programs may invoke system calls directly without a system call library, however. Programs frequently use other libraries such as the standard 1/0 library to provide a more sophisticated use of the system calls. The libraries are linked with the programs at compile time

and are thus part of the user program for purposes of The figure partitions the set of system calls into those that interact with the file subsystem and those that interact with the process control subsystem. The file subsystem manages files, aJlocating file space, administering free space, controlling access to files, and retrieving data for users. Processes interact with the file subsystem via a specific set of system calls, such as open (to open a file for reading or writing}, close, read, write, stat (query the attributes of a file), chown (change the record of who owns the file), and chmod (change the access permissions of a file).

The file subsystem accesses file data using a buffering mechanism that regulates data flow between the kernel and secondary storage devices. The buffering mechanism interacts with block 1/0 device drivers to initiate data transfer to and from the kernel. Device drivers are the kernel modules that control the operation of peripheral devices. Block 1/0 devices are random access storage devices; alternatively, their device drivers make them appear to be random access storage devices to the rest of the system. For example, a tape driver may allow the kernel to treat a tape unit as a random access storage device. The file subsystem also interacts directly with "raw" I/0 device drivers without the intervention of a buffering mechanism. Raw devices, sometimes called character devices, include all devices that are not block devices.

The process control subsystem is responsible for process synchronization, interprocess communication, memory management, and process scheduling. The file subsystem and the process control subsystem interact when loading a file into memory for execution, the process subsystem reads executable files into memory before executing them. Some of the system calls for controlling processes are fork (create a new process), exec (overlay the image of a program onto the running process), exit (finish executing a process), wait (synchronize process execution with the exit of a previously forked process), brk (control the size of memory allocated to a process), and signal (control process response to extraordinary events). Chapter 7 will examine these system calls and others.

The memory management module controls the allocation of memory. If at any time the system does not have enough physical memory for all processes, the kernel moves them between main memory and secondary memory so that all processes get a fair chance to execute. two policies for managing memory: swapping and demand paging. The swapper process is sometimes called the scheduler, because it “schedules" the allocation of memory for processes and influences the operation of the CPU scheduler. However, this text will refer to it asthe swapper to avoid confusion with the CPU scheduler. The scheduler module allocates the CPU to processes. It schedules them to run in turn until they voluntarily relinquish the CPU while awaiting a resource or until the kernel preempts them when their recent run time exceeds a time quantum. The scheduler then chooses the highest priority eligible process to run; the original process will run again when it is the highest priority eligible process available. There are several forms of interprocess communication, ranging from asynchronous signaling of events to synchronous transmission of messages between processes. Finally, the hardware control is responsible for handling interrupts and for communicating with the machine. Devices such as disks or terminals may interrupt the CPU while a process is executing. If so, the kernel may resume execution of the interrupted process after servicing the interrupt: Interrupts are not serviced by special processes but by special functions in the kernel, called in the context of the currently running process.

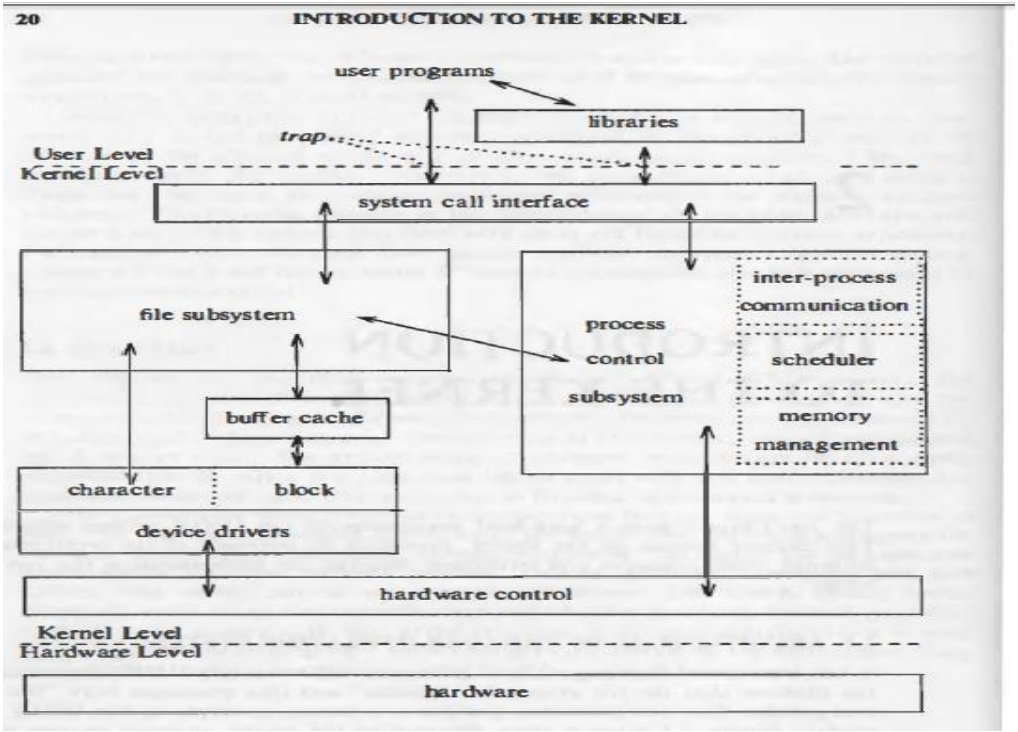


Fig. 1.3 Block Diagram of the System Kernel

G.K. Gujar Memorial Charitable Trust's **DR.ASHOK GUJAR TECHNICAL INSTITUTE'S** **DR. DAULATRAO AHER COLLEGE OF ENGINEERING, KARAD**	
Lab Manual	
Subject : Operating System-II	
Expt. No: 02	**TITLE** : Study and Implementation of general , directory and file utilities

Aim: Implementation of general, directory and file utilities

COMMAND :

1.Date Command :

This command is used to display the current data and time.

Syntax :

$date

$date +%ch

Options : -

a = Abbreviated weekday.
A = Full weekday.
b = Abbreviated month.
B = Full month.
c = Current day and time.
C = Display the century as a decimal number.
d = Day of the month.
D = Day in "mm/dd/yy" format
h = Abbreviated month day.
H = Display the hour.
L = Day of the year.
m = Month of the year.
M = Minute.
P = Display AM or PM
S = Seconds
T = HH:MM:SS format
u = Week of the year.
y = Display the year in 2 digit.
Y = Display the full year.
Z = Time zone.

To change the format:

Syntax :

$date '+%H-%M-%S'

2. Calender Command :

This command is used to display the calendar of the year or the particular month of calendar year.

Syntax :

a. $cal <year>

b. $cal <month> <year>

Here the first syntax gives the entire calendar for given year & the second Syntax gives the calendar of reserved month of that year.

3. Echo Command :

This command is used to print the arguments on the screen .

Syntax : $echo <text>

Multi line echo command :

To have the output in the same line , the following commands can be used.

Syntax : $echo <text\>text

To have the output in different line, the following command can be used.

Syntax : $echo "text

>line2

>line3"

4. Banner Command :

It is used to display the arguments in „#" symbol .

Syntax : $banner <arguments>

5. 'who' Command :

It is used to display who are the users connected to our computer currently.

Syntax : $who – option"s

Options : -

H–Display the output with headers.

b–Display the last booting date or time or when the system was lastely rebooted.

6. 'who am i' Command :

Display the details of the current working directory.

Syntax : $who am i

7. 'tty' Command :

It will display the terminal name.

Syntax : $tty

8. 'Binary' Calculator Command :

It will change the „$" mode and in the new mode, arithematic operations such as +,-,*,/,%,n,sqrt(),length(),=, etc can be performed . This command is used to go to the binary calculus mode.

Syntax :

$bc operations

^d

$

1 base –inputbase
0 base – outputbase are used for base conversions.
Base :
Decimal = 1 Binary = 2 Octal = 8 Hexa = 16

9.'CLEAR' Command :

It is used to clear the screen.
Syntax : $clear

10.'MAN' Command :

It help us to know about the particular command and its options & working. It is like „help" command in windows .
Syntax : $man <command name>

11.MANIPULATION COMMAND :

It is used to manipulate the screen.
Syntax : $tput <argument>

Arguments :

1. Clear – to clear the screen.
2. Longname – Display the complete name of the terminal.
3. SMSO – background become white and foreground become black color.
4. rmso – background become black and foreground becomes white color.
5. Cop R C – Move to the cursor position to the specified location.
6. Cols – Display the number of columns in our terminals.

12.LIST Command :

It is used to list all the contents in the current working directory.
Syntax : $ ls – options <arguments>
If the command does not contain any argument means it is working in the Current directory.

Options :

a– used to list all the files including the hidden files.
c– list all the files columnwise.
d- list all the directories.
m- list the files separated by commas.
p- list files include „/" to all the directories.
r- list the files in reverse alphabetical order.
f- list the files based on the list modification date.
x-list in column wise sorted order.

DIRECTORY RELATED COMMANDS :

1.Present Working Directory Command :

To print the complete path of the current working directory.
Syntax : $pwd

2.MKDIR Command :

To create or make a new directory in a current directory .
Syntax : $mkdir <directory name>

3.CD Command :

To change or move the directory to the mentioned directory .
Syntax : $cd <directory name.

4.RMDIR Command :

To remove a directory in the current directory & not the current directory itself.

Syntax : $rmdir <directory name>

FILE RELATED COMMANDS :

1.CREATE A FILE :

To create a new file in the current directory we use CAT command.

Syntax :

$cat > <filename.

The > symbol is redirectory we use cat command.

2.DISPLAY A FILE :

To display the content of file mentioned we use CAT command without „>" operator.

Syntax :

$cat <filename.

Options –s = to neglect the warning /error message.

3.COPYING CONTENTS :

To copy the content of one file with another. If file doesnot exist, a new file is created and if the file exists with some data then it is overwritten.

Syntax :

$ cat <filename source> >> <destination filename>

$ cat <source filename> >> <destination filename> it is avoid overwriting.

Options : -

-n content of file with numbers included with blank lines.

Syntax :

$cat –n <filename>

4.SORTING A FILE :

To sort the contents in alphabetical order in reverse order.

Syntax :

$sort <filename >

Option : $ sort –r <filename>

5.COPYING CONTENTS FROM ONE FILE TO ANOTHER :

To copy the contents from source to destination file . so that both contents are same.

Syntax :

$cp <source filename> <destination filename>

$cp <source filename path > <destination filename path>

6.MOVE Command :

To completely move the contents from source file to destination file and to remove the source file.

Syntax :

$ mv <source filename> <destination filename>

7.REMOVE Command :

To permanently remove the file we use this command .

Syntax :

$rm <filename>

8. WORD Command :

To list the content count of no of lines , words, characters .

Syntax :

$wc<filename>

Options :

-c – to display no of characters.

-l – to display only the lines.

-w – to display the no of words.

9. LINE PRINTER :

To print the line through the printer, we use lp command.

Syntax :

$lp <filename>

10. PAGE Command :

This command is used to display the contents of the file page wise & next page can be viewed by pressing the enter key.

Syntax :

$pg <filename>

11. FILTERS AND PIPES

HEAD : It is used to display the top ten lines of file.

Syntax: $head<filename>

TAIL : This command is used to display the last ten lines of file.

Syntax: $tail<filename>

PAGE : This command shows the page by page a screenfull of information is displayed after which the page command displays a prompt and passes for the user to strike the enter key to continue scrolling.

Syntax: $ls –a\p

MORE : It also displays the file page by page .To continue scrolling with more command , press the space bar key.

Syntax: $more<filename>

G.K. Gujar Memorial Charitable Trust's

DR.ASHOK GUJAR TECHNICAL INSTITUTE'S
DR. DAULATRAO AHER COLLEGE OF ENGINEERING, KARAD

Lab Manual

Subject : Operating System-II

Expt. No: 03	**TITLE** : Implementation of Scheduling Algorithms.

Aim: Study and Implementation of Scheduling Algorithms.

Description: A Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms. There are six popular process scheduling algorithms which we are going to discuss in this chapter −

- First-Come, First-Served (FCFS) Scheduling
- Shortest-Job-Next (SJN) Scheduling
- Priority Scheduling
- Shortest Remaining Time
- Round Robin(RR) Scheduling
- Multiple-Level Queues Scheduling

These algorithms are either **non-preemptive or preemptive**. Non-preemptive algorithms are designed so that once a process enters the running state, it cannot be preempted until it completes its allotted time, whereas the preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.

1) First Come First Serve (FCFS)

- Jobs are executed on first come, first serve basis.
- It is a non-preemptive, pre-emptive scheduling algorithm.
- Easy to understand and implement.
- Its implementation is based on FIFO queue.
- Poor in performance as average wait time is high.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16

Wait time of each process is as follows −

Process Wait Time : Service Time - Arrival Time

P0 0 - 0 = 0

P1 5 - 1 = 4

P2 8 - 2 = 6

P3 16 - 3 = 13

Average Wait Time: (0+4+6+13) / 4 = 5.75

2) Shortest Job Next (SJN)

- This is also known as **shortest job first**, or SJF
- This is a non-preemptive, pre-emptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known.
- The processer should know in advance how much time process will take.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	3
P1	1	3	0
P2	2	8	16
P3	3	6	8

Wait time of each process is as follows −

Process Wait Time : Service Time - Arrival Time

P0 3 - 0 = 3

P1 0 - 0 = 0

P2 16 - 2 = 14

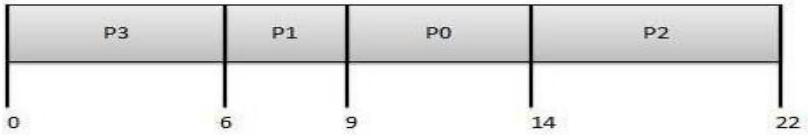
P3 8 - 3 = 5

Average Wait Time: (3+0+14+5) / 4 = 5.50

3) Priority Based Scheduling

- Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems.
- Each process is assigned a priority. Process with highest priority is to be executed first and so on.
- Processes with same priority are executed on first come first served basis.
- Priority can be decided based on memory requirements, time requirements or any other

Process	Arrival Time	Execute Time	Priority	Service Time
P0	0	5	1	9
P1	1	3	2	6
P2	2	8	1	14
P3	3	6	3	0



Wait time of each process is as follows −

Process Wait Time : Service Time - Arrival Time

P0 9 - 0 = 9

P1 6 - 1 = 5

P2 14 - 2 = 12

P3 0 - 0 = 0

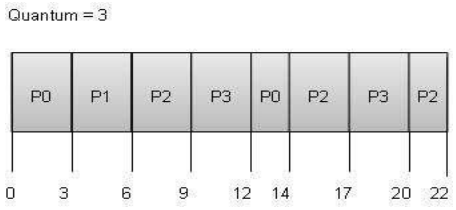
Average Wait Time: (9+5+12+0) / 4 = 6.5

4) Shortest Remaining Time

- Shortest remaining time (SRT) is the preemptive version of the SJN algorithm.
- The processor is allocated to the job closest to completion but it can be preempted by a newer ready job with shorter time to completion.
- Impossible to implement in interactive systems where required CPU time is not known.
- It is often used in batch environments where short jobs need to give preference.

5) Round Robin Scheduling

- Round Robin is the preemptive process scheduling algorithm.
- Each process is provided a fix time to execute, it is called a **quantum**.
- Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
- Context switching is used to save states of preempted processes.



6) Multiple-Level Queues Scheduling

Multiple-level queues are not an independent scheduling algorithm. They make use of other existing algorithms to group and schedule jobs with common characteristics.

- Multiple queues are maintained for processes with common characteristics.
- Each queue can have its own scheduling algorithms.
- Priorities are assigned to each queue.

For example, CPU-bound jobs can be scheduled in one queue and all I/O-bound jobs in another queue. The Process Scheduler then alternately selects jobs from each queue and assigns them to the CPU based on the algorithm assigned to the queue.

Expt. No: 04	**TITLE** : Study and implementation of building block primitives like redirect I/O and pipe

Aim: Study and implementation redirect I/O and pipe

Theory:

Most Unix system commands take input from your terminal and send the resulting output back to your terminal. A command normally reads its input from a place called standard input, which happens to be your terminal by default. Similarly, a command normally writes its output to standard output, which is also your terminal by default. UNIX system is to provide operating system primitives that enable users to write small, modular programs that can be used as building blocks to build more complex programs. One such primitive visible to shell users is the capability to redirect 1/0. Processes conventionally have access to three files: they read from their standard input file, write to their standard output file, and write error messages to their standard error file. Processes executing at a terminal typically use the terminal for these three files, but each may be "redirected" independently.

Output Redirection

The output from a command normally intended for standard output can be easily diverted to a file instead. This capability is known as output redirection: If the notation > file is appended to any command that normally writes its output to standard output, the output of that command will be written to file instead of your terminal –

Check following **who** command which would redirect complete output of the command in users file.

```
$ who > users
```

Notice that no output appears at the terminal. This is because the output has been redirected from the default standard output device (the terminal) into the specified file. If you would check *users* file then it would have complete content −

```
$ cat users
oko         tty01   Sep 12 07:30
ai          tty15   Sep 12 13:32
ruth        tty21   Sep 12 10:10
pat         tty24   Sep 12 13:07
steve       tty25   Sep 12 13:03
$
```

If a command has its output redirected to a file and the file already contains some data, that data will be lost. Consider this example −

```
$ echo line 1 > users
$ cat users
line 1
$
```

You can use >> operator to append the output in an existing file as follows −

```
$ echo line 2 >> users
$ cat users
line 1
line 2
$
```

Input Redirection

Just as the output of a command can be redirected to a file, so can the input of a command be redirected from a file. As the greater-than character > is used for output redirection, the less-than character < is used to redirect the input of a command.

The commands that normally take their input from standard input can have their input redirected from a file in this manner. For example, to count the number of lines in the file *users* generated above, you can execute the command as follows −

```
$ wc -l users
2 users
$
```

Here it produces output 2 lines. You can count the number of lines in the file by redirecting the standard input of the wc command from the file *users* −

```
$ wc -l < users
2
$
```

Note that there is a difference in the output produced by the two forms of the wc command. In the first case, the name of the file users is listed with the line count; in the second case, it is not.

In the first case, wc knows that it is reading its input from the file users. In the second case, it only knows that it is reading its input from standard input so it does not display file name.

Error redirection

When we enter an incorrect command or try to open an nonexistent file certain diagnostic message show up on the screen. This is the standard error stream whose default destination is terminal. Trying to cat a nonexistent file produces the error stream.

```
$ cat foo
cat: cannot open foo
```

cat fails to open the file and writes to standard error. If you are not using the C shell, you can redirect this stream to the file. Using the symbol standard o/p –

```
$ cat foo 2>errorfile
$  cat  errorfile
cat: cannot open foo
```

Pipe:

You can connect two commands together so that the output from one program becomes the input of the next program. Two or more commands connected in this way form a pipe.

To make a pipe, put a vertical bar (|) on the command line between two commands.

When a program takes its input from another program, performs some operation on that input, and writes the result to the standard output, it is referred to as a *filter*.

The grep Command

The grep program searches a file or files for lines that have a certain pattern. The syntax is −

```
$grep pattern file(s)
```

The name "grep" derives from the ed (a UNIX line editor) command g/re/p which means "globally search for a regular expression and print all lines containing it."

A regular expression is either some plain text (a word, for example) and/or special characters used for pattern matching.

The simplest use of grep is to look for a pattern consisting of a single word. It can be used in a pipe so that only those lines of the input files containing a given string are sent to the standard output. If you don't give grep a filename to read, it reads its standard input; that's the way all filter programs work −

```
$ls -l | grep "Aug"
-rw-rw-rw-   1 john  doc     11008 Aug  6 14:10 ch02
-rw-rw-rw-   1 john  doc      8515 Aug  6 15:30 ch07
-rw-rw-r--   1 john  doc      2488 Aug 15 10:51 intro
-rw-rw-r--   1 carol doc      1605 Aug 23 07:35 macros
$
```

Aim: Study & Implementation of process related utilities

Theory:

An instance of a program is called a Process. A process said to be born when the program starts execution and remains alive as long as program is active. After execution is complete, the process is said to be die.

Two important attributes of process are:

PID: Every process has an id associated to it. It's an unique identifier, and that's how we can reference a specific process.

PPID: That's the parent's PID. Every (well, almost) process has a parent process, the process that was responsible for its creation.
Following are some commands that is related to process.

ps(The process status command) :-

The ps command is used to display characteristic of the process. It reads through the kernel data structure or process table to fetch the attributes of the process.

This command without parameter list out processes associated with user at particular terminal.

```
$ ps
PID      TTY    TIME      COMMAND
30       01     0.03         sh
56       01     0.00         ps
$
```

Each line shows PID i.e. process ID , TTY i.e. the terminal with which process is associated, TIME i.e. processor time that has been consumed since the process has stated & COMMAND i.e. the process name.
The ps command by default doesn't give detail of process. To get those you have use the command with –f (Full) option

```
$ ps –f

UID      PID PPID       C      STIME     TTY   TIME     COMMAND
```

UID gives User Id, PPID gives parent ID i.e. ID of parent process.

Shell has PID 30 & PPID is 1 i.e. the process stared shell has PID 1.

Next column indicate the letter C indicate that amount of time consumed by process. Next column STIME indicates time that has elapsed since the process started i.e. start Time.

The option **–u (**user) lets you to know that activities of any user at any time.
This option is followed by UID.

```
$ ps –u student
```

This gives process status that the student (UID) started.

The option **–a** (all) ,list out processes of ALL USERS

```
$ ps –a
```

The option **-l** (long listing) gives long list of status.

```
$ ps –l
```

```
F  S  UID  PID  PPID  C  PRI NI  ADDR  SZ  WCHAN  TTY TIME  CMD
```

Where,

S indicate that whether the process is running or sleeping.(by S or R)

PRI indicate the priority of the process

ADDR indicate the memory /disk address of process.

SZ indicate the size of the process.

When a process executed it may have wait for while a system resources which is not yet available .this is indicated by WCHAN. For running process this column is blank.

Pid (process ID):-

This command is used to identify process ID.

The process is identified by its PID which is stored in special variable $$. To know PID of your current shell, simply type,

```
$ echo $$
```

low PID indicate process was initiated quit early .When logout, & again login shell assign different PID to process.

Background Processes

A background process runs without being connected to your keyboard. If the background process requires any keyboard input, it waits.

The advantage of running a process in the background is that you can run other commands; you do not have to wait until it completes to start another!

& (No Logging Out) and nohup(Log Out Safely)

A multitasking system lets user to do more than one job at a time. There is only one job in foreground & the rest one is executing in background. The '&' operator is provided by shell to run process in background. After command line is terminated with & that command is executed in background.

```
$ sort –o emp.lst emp.lst &
71
$
```

here emp.lst file is sorted & o/p is stored in emp.lst file but this is done in background.
This command simply return PID of process & gives prompt to execute new command.

Suppose after that you execute

```
$ ps –f
```

Then it gives that both sort & ps has same PPID i.e of shell because one process(sort) is running in background & other (ps) running in foreground. You can execute less important job in background & execute most important one in foreground.

Suppose background process is running & if user logout (i.e shell die) then all process(child of shell) will die. So nohup (no hangup) command to permit execution of process even if user logged off.

```
$ nohup sort emp.lst &
      101
      sending o/p to nohup.out.
$
```

use ps command from other terminal to observe effect of nohup.

```
$ ps –f
```

nice(job execution with low priority):-

nice command is used to reduce the priority of job.

```
$ nice wc –l welcome &
```

by default nice reduce the priority by 10 units. The effect of increasing the nice value by 10 units. The higher the nice value lower will be the priority. The amount of reduction will also specified in parameter,

```
$ nice -15 wc  -l welcome &
```

you can see value by imply invoking the

```
$ ps –l
```

wait :-

Waiting for completion of the background process. The background execution is meaningful if we execute more than one job. It is pointless if we executed no. of process in background but idle in foreground. But in some application require that's after execution of background process the foreground is executed, in such cases you require wait till background process completed.

Take one example,

```
$ cut –c1-4 emp.lst > temp.01 &
121
 $ cut –c5-10 emp.lst > temp.02 &
122
$ cut –c11-15 emp.lst > temp.03 &
125
$ cut –c16-20 emp.lst > temp.04 &
132
$ cut –c21-25 emp.lst > temp.05 &
$ cut –c26-30 emp.lst > temp.06 &
144
$
```

there are six background job's . you can wait before pasting this temp.* file .

```
$ wait
$ paste –d "|" temp.??
```

kill (premature termination of process):-

Interprocess communication is one facility available in Unix. The process can communicate by sending the signals. When a process receives signal then it may accept it, ignore it. You can terminate process by using the kill command. The command use one or more PID as argument.

```
$ kill 105
$
```

it will terminate job having PID 105.

Kill ,by default , uses the signal number 15 (SIGTERM) to terminate the process. Some program may accept it or some may ignore it. In that case use signal number 9 (SIGKILL) to kill any process. This sometime known as "sure kill".

```
$ kill -9 121
$
```

this will surly kill the process has PID 121.

Conclusion:-

Thus we have to study the various utility that is used to find out status of the process, Running processes in background & killing them.

G.K. Gujar Memorial Charitable Trust's **DR.ASHOK GUJAR TECHNICAL INSTITUTE'S** **DR. DAULATRAO AHER COLLEGE OF ENGINEERING, KARAD**	
Lab Manual	
Subject : Operating System-II	
Expt. No: 06	**TITLE** : File Security & Text Manipulation Utility

Aim:- Study of security of the file & study of text manipulation utility

THEORY:-

Utility For changing the permission & mode of file:-

For every file & directory in unix has three classes of the users who has access to file :

Owner :-The owner is the user who initially create the file. It is possible to creator to give it to another.(by chown command)

Group:- Several user combined into user group & so there is group ownership of the file & dir.

Public:- All users of the system.

The permission for the file are:-

Read :- A user has permission for read file can read only file.

Write :- The user that has permission for the write can write in file.

Execute :- The user has permission for the execute the file can use file name as command in system .

r w x r w x r w x

Owner Permission Group Permission Others Permission.

So, read only permission can given as,

r--r--r—

For getting the permission for file simply type command **ls –l**

Changing Permission by chmod:-

The format of chmod is as,

$ chmod mode filename.

The chmod command is used to set the permission of one or more files for all three categories of users. It can be run only by the user and the superuser. This command can be used in two ways:

i) In relative manner by specifying the changes to the current permissions.

ii) In an absolute manner by specifying the final permision.

i) Relative permission

In this manner chmod only changes the permission specified in command line and leaves other permissions unchanged. Syntax:

chmod category operation permission filename*(s)*

- User category (User, Group, Others)
- The operation to be performed(assign or remove permission)
- The type of permission(read, write, execute)

Abbreviation used by chmod are given by ,

u :- User(owner) of file.

g:- Group permission.

o :- Other(public) permission.

a:- all user ,group & other permission.

= :- assign a permission absolutely

+ :- add permission

- :- Take away permission.

r :- read permission

w :- write permission

x :- execute permission

To assign execute permission to the user (owner)

$ chmod u+x abc

now user(owner) can execute the file but other categories still can't.

To enable all of them to execute this file, use multiple characters to represent the user category.

$ chmod ugo+x abc

string ugo combines all the three categories

$ chmod a+x abc a implies ugo

$ chmod +x abc by default a is implied

To assign a same set of permission to a group of file-

$ chmod u+x chap1 chap2 chap3

Permissions are removed with the – operator

$ chmod go-x abc *removes the execute permission of group and others*

$ chmod a-x, ug+w abc *removes the execute permission from all & assign the write permission to user & group.*

More than one permission can also be set,

$ chmod u+wx abc

ii) Absolute permission

Initial Permission:-

The permission is assigned when a file or directory is created are under control of the something called as “user mask” i.e **umsask.**

You can display the current value of the umask by

$ umask

0000

This means that when you create the file its permission is by default set **to rw-rw-rw-** (mode 666)

The value of the augment is find by subtracting the mode you want as default from the current default mode.

Suppose that you want default mode is 644(rw--r--r--) for file, then,

666 (current mode)

-644 (mode u want)

022

when you type umask command then give 022 as argument to set default mode

$ umask 022

After this when you create file by default it’s mode is rw—r--r--

In this category octal digits are used to set the permission. To represent the permission of each category by one octal digit:

Binary	**Octal**	**permission**	**Significance**
000	0	– – –	No permission
001	1	– – x	Executable only
010	2	– w –	Writable only
011	3	– w x	Writable & Executable
100	4	r – –	Readable only
101	5	r – x	Readable & Executable

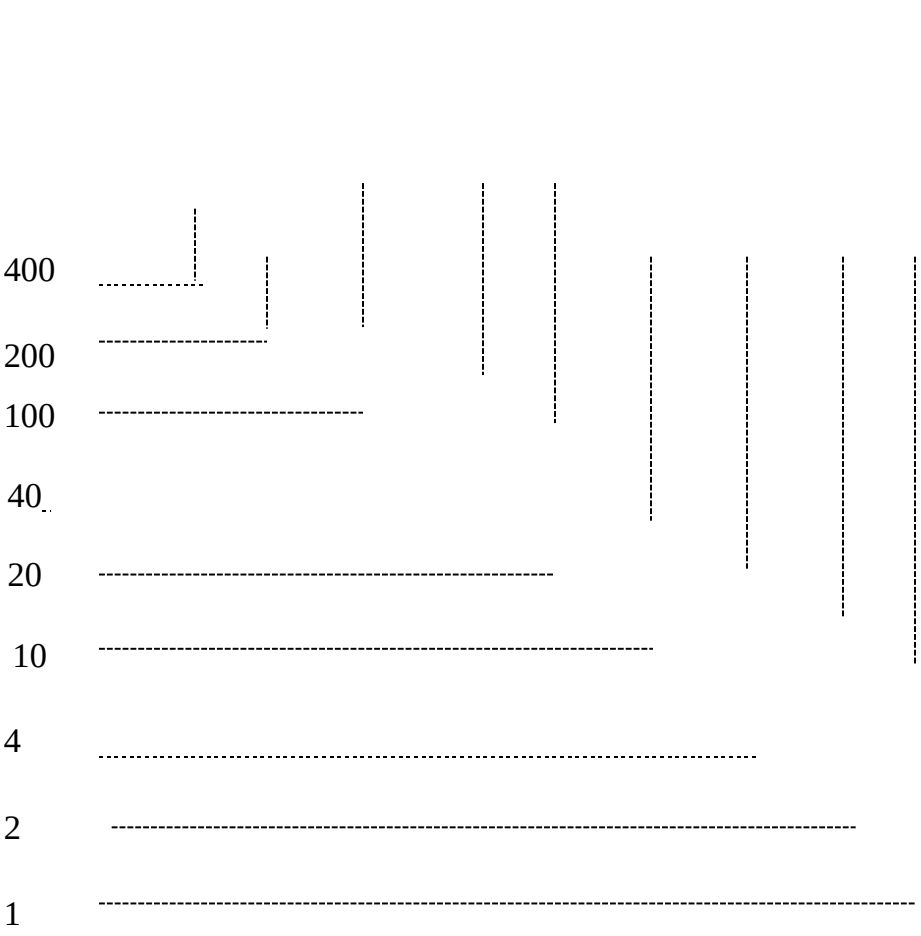
110	6	r w –	Readable & Executable
111	7	r w x	Readable, Writable & Executable

You can write the mode in binary format by, suppose

"rw-r--r--" simply write 110100100 i.e 644

You can simply calculate this by diagram given below,

r w x r w x r w x



So if want the file readable & writable by owner , readable by the group, readable by the all users then,

400
200
40
4

644

so command is,

$ chmod 644 <filename>

Octal Code	Permission	Meaning
644	rw-r--r--	Owner can read & write file & everyone can read
755	rwxr-xr-x	Owner can read & write file & every one can read ,execute file
711	rwx--x--x	Owner can read ,write & execute file & every body else can execute file only.
444	r--r--r--	Read-only for everybody

To assign the read write permission to all the three categories.

$ chmod 666 abc

To remove write permission from group and others

$ chmod 644 abc

To assign all permission to the owner, read write permission to the group, and only execute permission to others, use :

$ chmod 761 abc

Utility to change the ownership of the file:-

File ownership:-

When file is created ,then user of that file who create file become

the owner of the file.& the group in which the user belong becomes the group owner of the file.

There are two command that change the ownership of file but it is used only by owner of the file.

The chown command gives away the ownership of file to another user.

To display file owned by any user(consider file index* i.e index,index1) you can use command,

$ ls –l index*

it display details of the file.(suppose owner of file index* is usercse)

suppose you want to change the ownership to another user (say student) then

$ chown student index*

then the ownership of index* is given to the student, so usercse can not longer edit file.

Similarly you can change the group ownership of the file,

$ chgrp <groupname> <filename>

It change the group ownership of file to group given in command.

G.K. Gujar Memorial Charitable Trust's

DR.ASHOK GUJAR TECHNICAL INSTITUTE'S
DR. DAULATRAO AHER COLLEGE OF ENGINEERING, KARAD

Lab Manual

Subject : Operating System-II

Expt. No: 07	**TITLE** : Write programs using the I/O system calls of UNIX operating system (open, read, write, close etc).

Aim: Implementation of System call for file system.

Theory: Following are the system calls for accessing existing file.

1. open

2. read

3.write

4.lseek

5.close

1. OPEN:-

The open system call is the first step a process must take to access the data in a file. The syntax for the open system call is

fd - open(pathname, flags, modes);

where *pathname* is a file name.

flags indicate the type of open (such as for reading or writing).

and *modes* give the file permissions if the file is being created.

The open system call returns an integer1 called the user file descriptor. Other file operations, such as reading, writing, seeking, duplicating the file descriptor, setting file l/0 parameters, determining file status, and closing the file, use the file descriptor that the open system call returns. The kernel searches the file system for the file name parameter using algorithm namei. It checks permissions for opening the file after it finds the in-core inode and allocates an entry in the file table for the open file. The file table entry contains a pointer to the inode of the open file and a field that indicates the byte offset in the file where the kernel expects the next read or write to begin. The kernel initializes the offset to 0 during the open call, meaning that the initial read or write starts at the beginning of a file by default. Alternatively, a process can open a file in write-append mode, in which case the kernel initializes the offset to the size of the file. The kernel allocates an entry in a private table in the process u area, called the user file descriptor table, and notes the index of this entry. The index is the file descriptor that is returned to the user. The entry in the user file table points to the entry in the global file table.

2. READ:-

The syntax of the read system call is

number - read(fd, buffer, count)

where *fd* is the file descriptor returned by open,

buffer is the address of a data structure in the user process that will contain the read data on successful

completion of the call,

count is the number of bytes the user wants to read, and

number is the number of bytes actually read.

3. WRITE:-

The syntax for the write system call is

number = write(fd, buffer, count);

where the meaning of the variables *fd, buffer, count, and number* are the same as they are for the read system call. The algorithm for writing a regular file is similar to that for reading a regular file. However, if the file does not contain a block that corresponds to the byte offset to be written, the kernel allocates a new block using algorithm alloc and assigns the block number to the correct position in the inode's table of contents. If the byte offset is that of an indirect block, the kernel may have to allocate several blocks for use as indirect blocks and data blocks. The inode is locked for the duration of the write, because the kernel may change the inode when allocating new blocks; allowing other processes access to the file could corrupt the inode if several processes allocate blocks simultaneously for the same byte offsets. When the write is complete, the kernel updates the file size entry in the inode if the file has grown larger.

4. LSEEK:-

The ordinary use of read and write system calls provides sequential access to a file, but processes can use the /seek system call to position the l/0 and allow random access to a file. The syntax for the system call is

position - lseek(fd, offset, reference);

where *fd* is the file descriptor identifying the file,
offset is a byte offset, and
reference indicates whether offset should be considered from the beginning of the file, from the current position of the read/write offset, or from the end of the file. The return value, position, is the byte offset where the next read or write will start**.**

5. CLOSE:-

A process closes an open file when it no longer wants to access it. The syntax for the close system call is

close(fd);

where *fd* is the file descriptor for the open file.

The kernel does the close operation by manipulating the file descriptor and the corresponding file table and inode table entries. If the reference count of the file table entry is greater than 1 because of dup or fork calls, then other user file descriptors reference the file table entry, as will be seen; the kernel decrements the count and the close completes. If the file table reference count is 1 , the kernel frees the entry and releases the in-core inode originally allocated in the open system call (algorithm iput). If other processes still reference the inode, the kernel decrements the inode reference count but leaves it allocated; otherwise, the inode is free for reallocation because its reference count is 0. When the close system call completes, the user file descriptor table entry is empty. Attempts by the process to use that file descriptor result in an error until the file descriptor is reassigned as a result of another system call. When a process exits, the kernel examines its active user file descriptors and internally closes each one. Hence, no process can keep a file open after it terminates.

G.K. Gujar Memorial Charitable Trust's

DR.ASHOK GUJAR TECHNICAL INSTITUTE'S
DR. DAULATRAO AHER COLLEGE OF ENGINEERING, KARAD

Lab Manual

Subject : Operating System-II

Expt. No: 08	**TITLE** : Study and implementation of Shell programming.

Aim: Study and implementation of various commands of Shell programming.

Theory:

Shell programming is a group of commands grouped together under single filename. After logging onto the system a prompt for input appears which is generated by a Command String Interpreter program called the shell. The shell interprets the input, takes appropriate action, and finally prompts for more input. The shell can be used either interactively - enter commands at the command prompt, or as an interpreter to execute a shell script. Shell scripts are dynamically interpreted, NOT compiled.

Common Shells.

C-Shell - csh : The default on teaching systems Good for interactive systems Inferior programmable features

Bourne Shell - bsh or sh - also restricted shell - bsh : Sophisticated pattern matching and file name substitution

Korn Shell : Backwards compatible with Bourne Shell Regular expression substitution emacs editing mode

Thomas C-Shell - tcsh : Based on C-Shell Additional ability to use emacs to edit the command line Word completion & spelling correction Identifying your shell.

Shell Keywords :

echo, read, if fi, else, case, esac, for , while , do , done, until , set, unset, readonly, shift, export, break, continue, exit, return, trap , wait, eval ,exec, ulimit , umask.

Shell Scripts

The basic concept of a shell script is a list of commands, which are listed in the order of execution. A good shell script will have comments, preceded by a pound sign, #, describing the steps.

There are conditional tests, such as value A is greater than value B, loops allowing us to go through massive amounts of data, files to read and store data, and variables to read and store data, and the script may include functions.

Shell scripts and functions are both interpreted. This means they are not compiled.

Example Script

Assume we create a test.sh script. Note all the scripts would have **.sh** extension. Before you add anything else to your script, you need to alert the system that a shell script is being started. This is done using the shebang construct. For example −

```
#!/bin/sh
```

This tells the system that the commands that follow are to be executed by the Bourne shell. *It's called a shebang because the # symbol is called a hash, and the ! symbol is called a bang.*

To create a script containing these commands, you put the shebang line first and then add the commands −

```
#!/bin/bash
```

```
pwd
ls
```

Variable Types

When a shell is running, three main types of variables are present −

- **Local Variables** − A local variable is a variable that is present within the current instance of the shell. It is not available to programs that are started by the shell. They are set at command prompt.

  E*XAMPLE*

  variable_name=value

  name="John Doe" x=5

- **Environment Variables** − An environment variable is a variable that is available to any child process of the shell. Some programs need environment variables in order to function correctly. Usually a shell script defines only those environment variables that are needed by the programs that it runs.

  VARIABLE_NAME=value

  export VARIABLE_NAME

  PATH=/bin:/usr/bin:.

  export PATH

- **Shell Variables** − A shell variable is a special variable that is set by the shell and is required by the shell in order to function correctly. Some of these variables are environment variables whereas others are local variables.

Extracting values from variables: To extract the value from variables, a dollar sign is used.

E*XAMPLE*

echo $variable_name

echo $name

echo $PATH

Rules : -

1.A variable name is any combination of alphabets, digits and an underscore („-„);

2.No commas or blanks are allowed within a variable name.

3.The first character of a variable name must either be an alphabet or an underscore.

4.Variables names should be of any reasonable length.

5.Variables name are case sensitive . That is , Name, NAME, name, Name, are all different variables.

Expression Command :

To perform all arithematic operations .

Syntax :

Var = „expr$value1‟ + $ value2‟

Arithmetic The Bourne shell does not support arithmetic. UNIX/Linux commands must be used to perform calculations.

E*XAMPLE*

n=`expr 5 + 5` echo $n

Operators The Bourne shell uses the built-in test command operators to test numbers and strings.

E*XAMPLE*

Equality:

= *string*

!= *string*

-eq *number*

-ne *number*

Logical:

-a *and*
-o *or*
! *not*

Logical:

AND &&
OR ||

Relational:

-gt *greater than*
-ge *greater than, equal to*
-lt *less than*
-le *less than, equal to*

Arithmetic **:**

+, -, *, /, %

Arguments (positional parameters) Arguments can be passed to a script from the command line. Positional parameters are used to receive their values from within the script.

E*XAMPLE*

At the command line:
$ scriptname arg1 arg2 arg3 ...
In a script:
echo $1 $2 $3 *Positional parameters*
echo $* *All the positional parameters*
echo $# *The number of positional parameters*

Read Statement :

To get the input from the user.

Syntax :

read x y
(no need of commas between variables)

Echo Statement :

Similar to the output statement. To print output to the screen, the echo command is used. Wildcards must be escaped with either a backslash or matching quotes.

Syntax :

Echo “String” (or) echo $ b(for variable).

E*XAMPLE*

echo "What is your name?"

Reading user input The read command takes a line of input from the user and assigns it to a variable(s) on the right-hand side. The read command can accept multiple variable names. Each variable will be assigned a word.

E*XAMPLE*

echo "What is your name?"
read name
read name1 name2 ...

Conditional Statement :

The if construct is followed by a command. If an expression is to be tested, it is enclosed in square brackets. The then keyword is placed after the closing parenthesis. An if must end with a fi.

Syntax :

1.if

This is used to check a condition and if it satisfies the condition if thendoes the next action , if not it goes to the else part.

2.if…else

Syntax :

```
If cp $ source $ target
Then
Echo File copied successfully
Else
Echo Failed to copy the file.
```

3.nested if

here sequence of condition are checked and the corresponding performed accordingly.

Syntax :

```
if condition
then
command
if condition
then
command
else
command
fi
fi
```

4.case ….. esac

This construct helps in execution of the shell script based on Choice.

Loops

There are three types of loops: while, until and for. The while loop is followed by a command or an expression enclosed in square brackets, a do keyword, a block of statements, and terminated with the done keyword. As long as the expression is true, the body of statements between do and done will be executed.

The until loop is just like the while loop, except the body of the loop will be executed as long as the expression is false.

The for loop used to iterate through a list of words, processing a word and then shifting it off, to process the next word. When all words have been shifted from the list, it ends. The for loop is followed by a variable name, the in keyword, and a list of words then a block of statements, and terminates with the done keyword.

The loop control commands are break and continue.

Break Statement :

This command is used to jump out of the loop instantly, without waiting to get the control command.

Arrays

(positional parameters) The Bourne shell does support an array, but a word list can be created by using positional parameters. A list of words follows the built-in set command, and the words are accessed by position. Up to nine positions are allowed. The built-in shift command shifts off the first word on the left-hand side of the list. The individual words are accessed by position values starting at 1.

E*XAMPLE*

```
set word1 word2 word3
echo $1 $2 $3              Displays word1, word2, and word3
set apples peaches plums
shift                      Shifts off apples
echo $1                    Displays first element of the list
echo $2                    Displays second element of the list
 echo $*                   Displays all elements of the list
```

Command substitution To assign the output of a UNIX/Linux command to a variable, or use the output of a command in a string, backquotes are used.

E*XAMPLE*

```
variable_name=`command`
echo $variable_name
now=`date`
echo $now
echo "Today is `date`"
```

File Testing

The Bourne shell uses the test command to evaluate conditional expressions and has a built-in set of options for testing attributes of files, such as whether it is a directory, a plain file (not a directory), a readable file, and so forth.

E*XAMPLE*

-d *File is a directory*
-f *File exists and is not a directory*
–r *Current user can read the file*
–s *File is of nonzero size*
–w *Current user can write to the file*
–x *Current user can execute the file*

Aim: : Study of system startup & init

Theory:

One of the most powerful aspects of Linux concerns its open method of starting and stopping the operating system, where it loads specified programs using their particular configurations, permits you to change those configurations to control the boot process, and shuts down in a graceful and organized way.

Beyond the question of controlling the boot or shutdown process, the open nature of Linux makes it much easier to determine the exact source of most problems associated with starting up or shutting down your system. A basic understanding of this process is quite beneficial to everybody who uses a Linux system.

A lot of Linux systems use **lilo**, the LInux LOader for booting operating systems. We will only discuss GRUB, however, which is easier to use and more flexible.

The boot process

When an x86 computer is booted, the processor looks at the end of the system memory for the BIOS (Basic Input/Output System) and runs it. The BIOS program is written into permanent read-only memory and is always available for use. The BIOS provides the lowest level interface to peripheral devices and controls the first step of the boot process.

The BIOS tests the system, looks for and checks peripherals, and then looks for a drive to use to boot the system. Usually it checks the floppy drive (or CD-ROM drive on many newer systems) for bootable media, if present, and then it looks to the hard drive. The order of the drives used for booting is usually controlled by a particular BIOS setting on the system. Once Linux is installed on the hard drive of a system, the BIOS looks for a Master Boot Record (MBR) starting at the first sector on the first hard drive, loads its contents into memory, then passes control to it.

This MBR contains instructions on how to load the GRUB (or LILO) boot-loader, using a pre-selected operating system. The MBR then loads the boot-loader, which takes over the process (if the boot-loader is installed in the MBR). In the default Red Hat Linux configuration, GRUB uses the settings in the MBR to display boot options in a menu. Once GRUB has received the correct instructions for the operating system to start, either from its command line or configuration file, it finds the necessary boot file and hands off control of the machine to that operating system.

GRUB features

This boot method is called *direct loading* because instructions are used to directly load the operating system, with no intermediary code between the boot-loaders and the operating system's main files (such as the kernel). The boot process used by other operating systems may differ slightly from the above, however. For example, Microsoft's DOS and Windows operating systems completely overwrite anything on the MBR when they are installed without incorporating any of the current MBR's configuration. This destroys any other information stored in the MBR by other operating systems, such as Linux. The Microsoft operating systems, as well as various other proprietary operating systems, are loaded using a chain loading boot method. With this method, the MBR points to the first sector of the partition holding the operating system, where it finds the special files necessary to actually boot that operating system.

GRUB supports both boot methods, allowing you to use it with almost any operating system, most popular file systems, and almost any hard disk your BIOS can recognize.
GRUB contains a number of other features; the most important include:

- GRUB provides a true command-based, pre-OS environment on x86 machines to allow maximum flexibility in loading operating systems with certain options or gathering information about the system.
- GRUB supports Logical Block Addressing (LBA) mode, needed to access many IDE and all SCSI hard disks. Before LBA, hard drives could encounter a 1024-cylinder limit, where the BIOS could not find a file after that point.
- GRUB's configuration file is read from the disk every time the system boots, preventing you from having to write over the MBR every time you change the boot options.

Init

The kernel, once it is loaded, finds **init** in sbin and executes it. When **init** starts, it becomes the parent or grandparent of all of the processes that start up automatically on your Linux system. The first thing **init** does, is reading its initialization file, /etc/inittab. This instructs **init** to read an initial configuration script for the environment, which sets the path, starts swapping, checks the file systems, and so on. Basically, this step takes care of everything that your system needs to have done at system initialization: setting the clock, initializing serial ports and so forth.

Then **init** continues to read the /etc/inittab file, which describes how the system should be set up in each run level and sets the default *run level*. A run level is a configuration of processes. All UNIX-like systems can be run in different process configurations, such as the single user mode, which is referred to as run level 1 or run level S (or s). In this mode, only the system administrator can connect to the system. It is used to perform maintenance tasks without risks of damaging the system or user data. Naturally, in this configuration we don't need to offer user services, so they will all be disabled. Another run level is the reboot run level, or run level 6, which shuts down all running services according to the appropriate procedures and then restarts the system.
Use the **who** to check what your current run level is:

```
willy@ubuntu:~$ who -r
         run-level 2 2006-10-17 23:22                   last=S
```

After having determined the default run level for your system, **init** starts all of the background processes necessary for the system to run by looking in the appropriate rc directory for that run level. **init** runs each of the kill scripts (their file names start with a K) with a stop parameter. It then runs all of the start scripts (their file names start with an S) in the appropriate run level directory so that all services and applications are started correctly.

On system startup, the scripts in rc2.d and rc3.d are usually executed. In that case, no services are stopped (at least not permanently). There are only services that are started

None of the scripts that actually start and stop the services are located in /etc/rc<x>.d. Rather, all of the files in /etc/rc<x>.d are symbolic links that point to the actual scripts located in /etc/init.d. A symbolic link is nothing more than a file that points to another file, and is used in this case because it can be created and deleted without affecting the actual scripts that kill or start the services. The symbolic links to the various scripts are numbered in a particular order so that they start in that order. You can change the order in which the services start up or are killed by

changing the name of the symbolic link that refers to the script that actually controls the service. You can use the same number multiple times if you want a particular service started or stopped right before or after another service, as in the example below, listing the content of /etc/rc5.d, where **crond** and **xfs** are both started from a linkname starting with "S90

After **init** has progressed through the run levels to get to the default run level, the /etc/inittab script forks a **getty** process for each virtual console (login prompt in text mode). **getty** opens tty lines, sets their modes, prints the login prompt, gets the user's name, and then initiates a login process for that user. This allows users to authenticate themselves to the system and use it. By default, most systems offer 6 virtual consoles, but as you can see from the inittab file, this is configurable.

/etc/inittab can also tell **init** how it should handle a user pressing **Ctrl**+**Alt**+**Delete** at the console. As the system should be properly shut down and restarted rather than immediately power-cycled, **init** is told to execute the command **/sbin/shutdown -t3 -r *now***, for instance, when a user hits those keys. In addition, /etc/inittab states what **init** should do in case of power failures, if your system has a UPS unit attached to it.

On most RPM-based systems the graphical login screen is started in run level 5, where /etc/inittab runs a script called /etc/X11/prefdm. The prefdm script runs the preferred X display manager, based on the contents of the /etc/sysconfig/desktop directory. This is typically **gdm** if you run GNOME or **kdm** if you run KDE, but they can be mixed, and there's also the **xdm** that comes with a standard X installation.

But there are other possibilities as well. On Debian, for instance, there is an initscript for each of the display managers, and the content of the /etc/X11/default-display-manager is used to determine which one to start.

The /etc/default and/or /etc/sysconfig directories contain entries for a range of functions and services, these are all read at boot time. The location of the directory containing system defaults might be somewhat different depending on your Linux distribution.

Besides the graphical user environment, a lot of other services may be started as well. But if all goes well, you should be looking at a login prompt or login screen when the boot process has finished.

Init run levels

The idea behind operating different services at different run levels essentially revolves around the fact that different systems can be used in different ways. Some services cannot be used until the system is in a particular state, or *mode*, such as being ready for more than one user or having networking available.

There are times in which you may want to operate the system in a lower mode. Examples are fixing disk corruption problems in run level 1 so no other users can possibly be on the system, or leaving a server in run level 3 without an X session running. In these cases, running services that depend upon a higher system mode to function does not make sense because they will not work correctly anyway. By already having each service assigned to start when its particular run level is reached, you ensure an orderly start up process, and you can quickly change the mode of the machine without worrying about which services to manually start or stop.

Feel free to configure unused run levels (commonly run level 4) as you see fit. Many users configure those run levels in a way that makes the most sense for them while leaving the standard run levels as they are by default. This allows them to quickly move in and out of their custom configuration without disturbing the normal set of features at the standard run levels.

If your machine gets into a state where it will not boot due to a bad /etc/inittab or will not let you log in because you have a corrupted /etc/passwd file (or if you have simply forgotten your password), boot into single-user mode.

Following are the available run levels

- 0 – halt
- 1 – Single user mode
- 2 – Multiuser, without NFS
- 3 – Full multiuser mode
- 4 – unused
- 5 – X11
- 6 – reboot

G.K. Gujar Memorial Charitable Trust's **DR.ASHOK GUJAR TECHNICAL INSTITUTE'S** **DR. DAULATRAO AHER COLLEGE OF ENGINEERING, KARAD**	
Lab Manual	
Subject : Operating System-II	
Expt. No: 10	**TITLE** : Study & implementation of utilities for the advanced system administrator Like useradd, passwd, df, du, find, fdisk, mkfs

Aim : Study and Implementation of utilities for advanced system administrator like useradd, passwd, df, du, find, fdisk, mkfs.

Theory:

System administration is an area with which the average user doesn't need to be connected.

'Root' is the super-user, who has Olympian powers over the running (and the possible distraction) of the system. The Super-user is unconstrained by any protections that the system has: he can get a file, can kill any process.

Starting up & Shutting down system.-

'Booting's the process you must apply to get the system running when the computer is initially switched on, or when the computer is restarted after halt.

Becoming the super user with su:-

Frequently, you will find that you are logged in as regular(non-super)user, & you suddenly have to requires your super user powers .Instead of logging out you simply use the su(for switch user).

```
$ su
Password: *******
#
```

You can use su to switch to be any other user providing you know the password.

```
$ su payroll
Password: ********
$
```

File System: -

A system V file system is a complete directory structure, including a root directory, & all the directories, which live under the root. A file system can correspond to a physical device, or more than one file system can live on a device.

Every file system has the same basic layout as shown in fig. & has 4 fundamentals parts:

1) The boot block:-

The very first block is reserved for a boot strap prog. It can contain

anything you want to place there. All file system information really starts in block one of the device.

2) The super block:-

The first block (block one) of every file system is called the 'super block'. It includes the major pieces of information about the file system name, number of block reserved for i-nodes, the free I-nodes list, & the start of chain of the tree block. All these topics are expanded upon in the section to follow.

3) I-nodes:-

Following the super block come a no. of blocks including I-nodes. The no. of blocks of I-nodes varies depending on the no. of blocks in the file system. The no. of I-nodes is specified in the one super block. There is one I-node for every directory & file in the file system. If an I-node is allocated, it contains a description of a directory or file somewhere in the file system.

4) Data Blocks:-

The rest of logical device is fully of data blocks include the actual data stored in the directories & files. There are also data blocks which server as indirect blocks, including block no. of large files.

- **useradd** - create a new user or update default new user information

Tag	Description
useradd [*options*] *LOGIN* **useradd** -D **useradd** -D [*options*]	

When invoked without the **-D** option, the **useradd** command creates a new user account using the values specified on the command line and the default values from the system.

Depending on command line options, the useradd command will update system files and may also create the new user's home directory and copy initial files.

Note: For these commands to work you must have superuser rights or be logged in as root.

 # useradd -D

Displays the defaults for new users. Output resembles the following:

GROUP=1001
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/bash
SKEL=/etc/skel
CREATE_MAIL_SPOOL=no

 # useradd newperson

Creates **newperson** as a new user. Once the new user has been added, you would need to use the **passwd** command to assign a password to the account.

Once a user has been created, you can modify any of the user settings, such as the user's home directory, using the **usermod** command.

- **Passwd:**

The **passwd** command is used to change the password of a user account. A normal user can run **passwd** to change their own password, and a system administrator (the superuser) can use **passwd** to change another user's password, or define how that account's password can be used or changed.

passwd [OPTION] [USER]

Change Your Own Password

passwd

Running **passwd** with no options will change the password of the account running the command. You will first be prompted to enter the account's current password:

(current) UNIX password:

If it is correct, you will then be asked to enter a new password:

Enter new UNIX password:

...and to enter the same password again, to verify it:

Retype new UNIX password:

If the passwords match, the password will be changed.

Change Another User's Password

#sudo passwd jeff

If you have superuser privileges, you can change another user's password. Here, we prefix the command with **sudo** to run it as the superuser. This command will change the password for user **jeff**. You will not be prompted for **jeff**'s current password.

Change Your Password Without Knowing Your Current Password

If you need to change your password because you forgot it, you will need to log in to the **root** account. To do this, you will need to know the password for user **root**.

Let's say your user name is **sally**, and you can't remember your password. However, you have administrator access to the system: you can log in as **root**, using the password for that account. Log in as **root**, and then from the command line, run:

 # passwd sally

- **du (disk usage):-**

The du command lets the administrator to fine out detail of the disk consumption If command used without option it tells space used by the current & sub directories

 # du

It gives pace usage.

The most of the dynamic space in system is consumed by directory /usr because it is in sub-directory that all users do their work.

du /usr

option:-

1) –a :- produce more detail o/p
2) –s :- produce non-recursive summary of disk usage of directory

specified in command.
e.g # du –s /home/usercse.

- **df (disk free):-**

It report amount of free space available on disk.
 # df

This command gives o/p for only file system . the file system & has how many blocks disk space free each block contain (512 bytes)
Also it indicate free i-nodes available in file system (how many additional file are created)
The system will continue to function find

find searches for files in a directory hierarchy.

find locates files on your system. Within each directory tree specified by the given *path*s, it evaluates the given *expression* from left to right, according to the rules of precedence (see "Operators", below) until the outcome is known. The outcome is "known" when the left hand side of the expression is determined to be FALSE for AND operations, or TRUE for OR operations. At that point **find** moves on to the next *path* until all *path*s have been searched.

find is a fundamental and extremely powerful tool for working with the files on your linux system. It can be used on its own to locate files, or in conjunction with other programs to perform operations on those files.

find syntax

find [-H] [-L] [-P] [-D debugopts] [-Olevel] [path...] [expression]

 # find

When using linux, running the **find** command without any options will locate and print a list of every file in and beneath the current directory. This includes all files in all subdirectories of the current directory.

#find .

Same as the above command. The "**.**" explicitly tells **find** that you want the search to begin in the current directory.

find . /home/jeff /home/stacy

Locate and print all files and directories in and beneath three different starting directories: the current directory, **/home/jeff**, and **/home/stacy**.

*find /usr/bin /usr/lib -name '*zip*'*

Locate and print all files and directories in and beneath either of the directories **/usr/bin** and **/usr/lib** which contains the text "**zip**" anywhere in the file or directory name.

find /home/jeff/fruit | grep 'apple'

This command tells **find** to locate and print a complete list of all files in and beneath the directory **/home/jeff/fruit**, and to pipe this listing to **grep**, which filters out any filename which does not contain the text "**apple**".

find . -name 'apple'

Locate and print a list of any file in or below the current directory whose name is exactly "**apple**", all lower case letters.

find . -iname 'apple'

Locate and print a list of any file in or below the current directory whose name is "**apple**", but match the letters case-insensitively. Therefore, files or directories named "**Apple**", "**ApplE**", and "**ApPLe**" will all be listed by this command.

find . -name 'apple' -type f

Locate and print a list of files in or below the current directory whose name is "**apple**"; do not display directories, sockets, or other non-regular file types.

find . -name 'apple' -type d

Locate a print a list of directories in or below the current directory whose name is "**apple**"; do not display regular files, or file types other than directory entries.

find . -group dev

Locate and print a list of any file in or below the current directory whose owning group is the **dev** group.

- **mkfs**

mkfs is used to build a Linux filesystem on a device, usually a hard disk partition. The device argument is either the device name (e.g. **/dev/hda1**, **/dev/sdb2**), or a regular file that will contain the filesystem. The size argument is the number of blocks to be used for the filesystem.

The exit code returned by **mkfs** is **0** on success and **1** on failure.

In actuality, **mkfs** is simply a front-end for the various file system builders (**mkfs.fstype**) available under Linux. The file system-specific builder is searched for in a number of directories, like perhaps **/sbin**, **/sbin/fs**, **/sbin/fs.d**, **/etc/fs**, **/etc** (the precise list is defined at compile time but at least contains **/sbin** and **/sbin/fs**), and finally in the directories listed in the PATH environment variable

mkfs syntax

mkfs [options] [-t type fs-options] device [size]

mkfs -t ext2 /dev/fd0

- **fdisk**

Partition table manipulator for Linux. Hard disks can be divided into one or more logical disks called partitions. This division is described in the partition table found in sector 0 of the disk.

fdisk -l
Disk /dev/sda: 21.5 GB, 21474836480 bytes
255 heads, 63 sectors/track, 2610 cylinders, total 41943040 sectors
*Units = sectors of 1 * 512 = 512 bytes*
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x000dd3ab

Device	*Boot*	*Start*	*End*	*Blocks*	*Id*	*System*
/dev/sda1	*	*2048*	*499711*	*248832*	*83*	*Linux*
/dev/sda2		*501758*	*41940991*	*20719617*	*5*	*Extended*
/dev/sda5		*501760*	*41940991*	*20719616*	*8e*	*Linux LVM*

TAG	DESCRIPTION
-b sectorsize	Specify the sector size of the disk. Valid values are 512, 1024, 2048 or 4096. (Recent kernels know the sector size. Use this only on old kernels or to override the kernel's ideas.) Since util-linux-ng 2.17 fdisk differentiates between logical and physical sector size. This option changes both sector sizes to sectorsize..
-h	Print help and then exit..
-c	Switch off DOS-compatible mode. (Recommended)
-C cyls	Specify the number of cylinders of the disk. I have no idea why anybody would want to do so
-H heads	Specify the number of heads of the disk. (Not the physical number, of course, but the number used for partition tables.) Reasonable values are 255 and 16.
-S sects	Specify the number of sectors per track of the disk. (Not the physical number, of course, but the number used for partition tables.) A reasonable value is 63.
-l	List the partition tables for the specified devices and then exit. If no devices are given, those mentioned in /proc/partitions (if that exists) are used

G.K. Gujar Memorial Charitable Trust's **DR.ASHOK GUJAR TECHNICAL INSTITUTE'S** **DR. DAULATRAO AHER COLLEGE OF ENGINEERING, KARAD**	
Lab Manual	
Subject : Operating System-II	
Expt. No: 11	**TITLE** : Study of memory allocation algorithm

Aim: Study of memory allocation algorithm

Theory:

Memory allocation is the process of assigning blocks of memory on request. Typically the *allocator* receives memory from the operating system in a small number of large blocks that it must divide up to satisfy the requests for smaller blocks. It must also make any returned blocks available for reuse. There are many common ways to perform this, with different strengths and weaknesses. A few are described briefly below.

- First fit
- Best fit
- Worst fit

These techniques can often be used in combination.

First fit

In the first fit algorithm, the allocator keeps a list of free blocks (known as the free list) and, on receiving a request for memory, scans along the list for the first block that is large enough to satisfy the request. If the chosen block is significantly larger than that requested, then it is usually split, and the remainder added to the list as another free block.

The first fit algorithm performs reasonably well, as it ensures that allocations are quick. When recycling free blocks, there is a choice as to where to add the blocks to the free list—effectively in what order the free list is kept:

Memory location (address)

This is not fast for allocation or recycling, but supports efficient merging of adjacent free blocks (known as *coalescence*). According to *Wilson et al. (1995)*, this ordering reduces *fragmentation*. It can also improve *locality of reference*.

Increasing size

This is equivalent to the *best fit* algorithm, in that the free block with the "tightest fit" is always chosen. The fit is usually sufficiently tight that the remainder of the block is unusably small.

Decreasing size

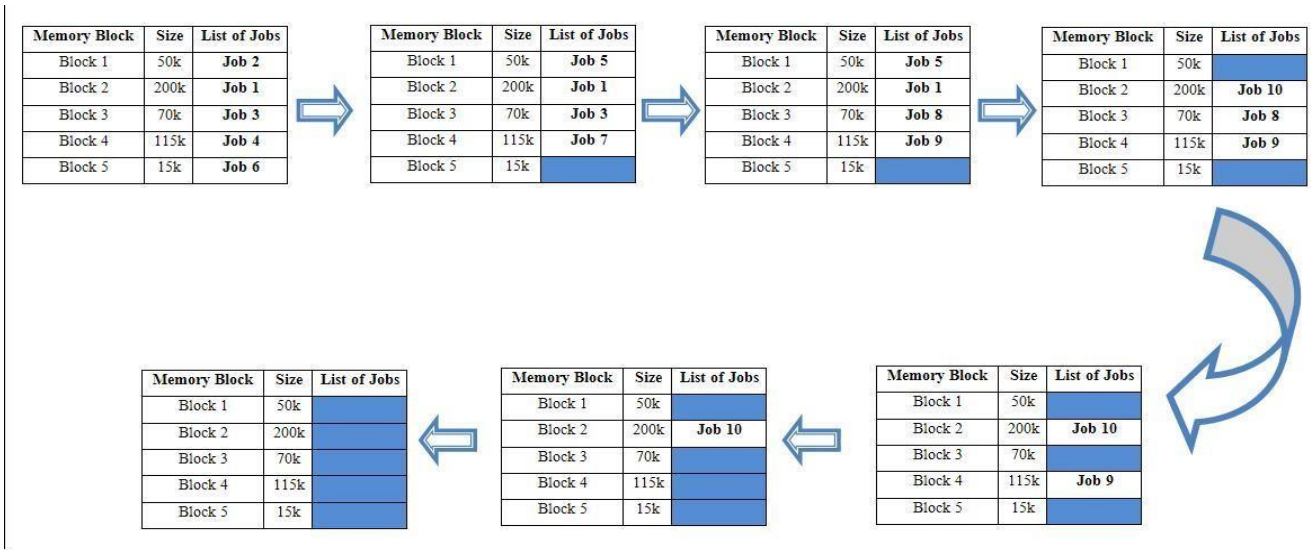
This is equivalent to the *worst fit* algorithm. The first block on the free list will always be large enough, if a large enough block is available. This approach encourages *external fragmentation*, but allocation is very fast.

Increasing time since last use

This is very fast at adding new free blocks, because they are added to the beginning of the list. It encourages good *locality of reference* (where blocks used together are not spread throughout memory), but can lead to bad external fragmentation.

A variation of first fit, known as ***next fit***, continues each search for a suitable block where the previous one left off, by using a roving pointer into the free block chain. This is not usually combined with increasing or decreasing size ordering because it would eliminate their advantages.

The illustration below shows that on the first cycle, job 1 to job 4 are submitted first while job 6 occupied block 5 because the remaining memory space is enough to its required memory size to be process. While job 5 is in waiting queue because the memory size in block 5 is not enough for the job 5 to be process. Then on the next cycle, job 5 replace job 2 on block 1 and job 7 replace job 4 on block 4 after both job 2 and job 4 finish their process. Job 8 is in waiting queue because the remaining block is not enough to accommodate the memory size of job 8. On the third cycle, job 8 replace job 3 and job 9 occupies block 4 after processing job 7. While Job 1 and job 5 remain on its designated block. After the third cycle block 1 and block 5 are free to serve the incoming jobs but since there are 10 jobs so it will remain free. And job 10 occupies block 2 after job 1 finish its turns. On the other hand, job 8 and job 9 remain on their block. Then on the fifth cycle, only job 9 and job 10 are to be process while there are 3 memory blocks free. In the sixth cycle, job 10 is the only remaining job to be process and lastly in the seventh cycle, all jobs are successfully process and executed and all the memory blocks are now free.



Advantage

Fastest algorithm because it searches as little as possible.

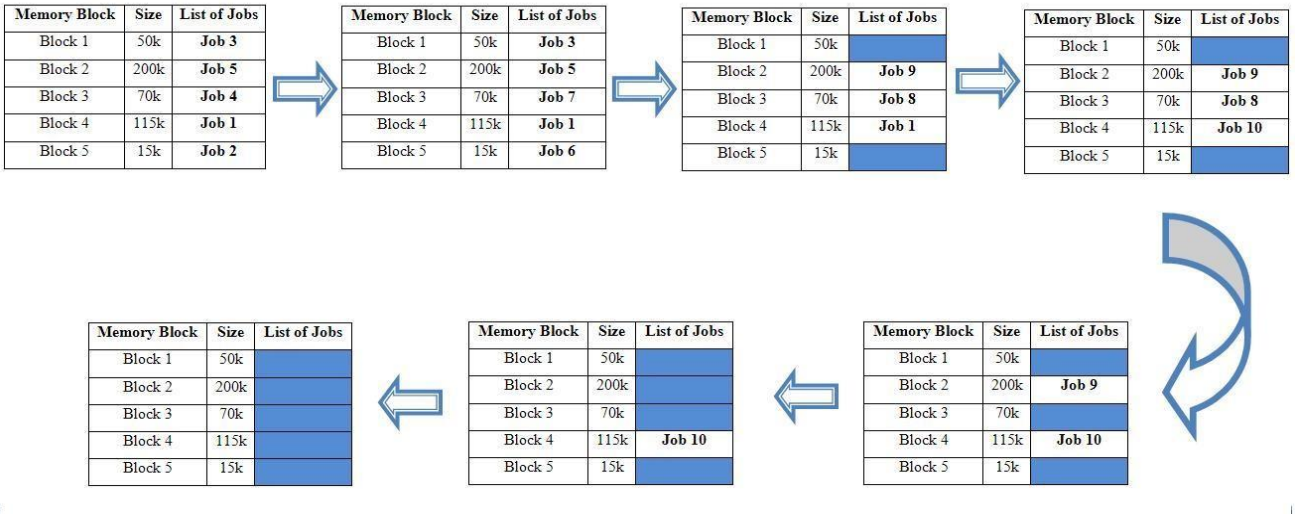
Disadvantage

The remaining unused memory areas left after allocation become waste if it is too smaller. Thus request for larger memory requirement cannot be accomplished.

Best Fit

The best fit deals with allocating the smallest free partition which meets the requirement of the requesting process. This algorithm first searches the entire list of free partitions and considers the smallest hole that is adequate. It then tries to find a hole which is close to actual process size needed. Best-fit memory allocation makes the best use of memory space but slower in making allocation. In the illustration below, on the first processing cycle, jobs 1 to 5 are

submitted and be processed first. After the first cycle, job 2 and 4 located on block 5 and block 3 respectively and both having one turnaround are replace by job 6 and 7 while job 1, job 3 and job 5 remain on their designated block. In the third cycle, job 1 remain on block 4, while job 8 and job 9 replace job 7 and job 5 respectively (both having 2 turnaround). On the next cycle, job 9 and job 8 remain on their block while job 10 replace job 1 (having 3 turnaround). On the fifth cycle only job 9 and 10 are the remaining jobs to be process and there are 3 free memory blocks for the incoming jobs. But since there are only 10 jobs, so it will remain free. On the sixth cycle, job 10 is the only remaining job to be process and finally on the seventh cycle, all jobs are successfully process and executed and all the memory blocks are now free.



.

Advantage

Memory utilization is much better than first fit as it searches the smallest free partition first available.

Disadvantage

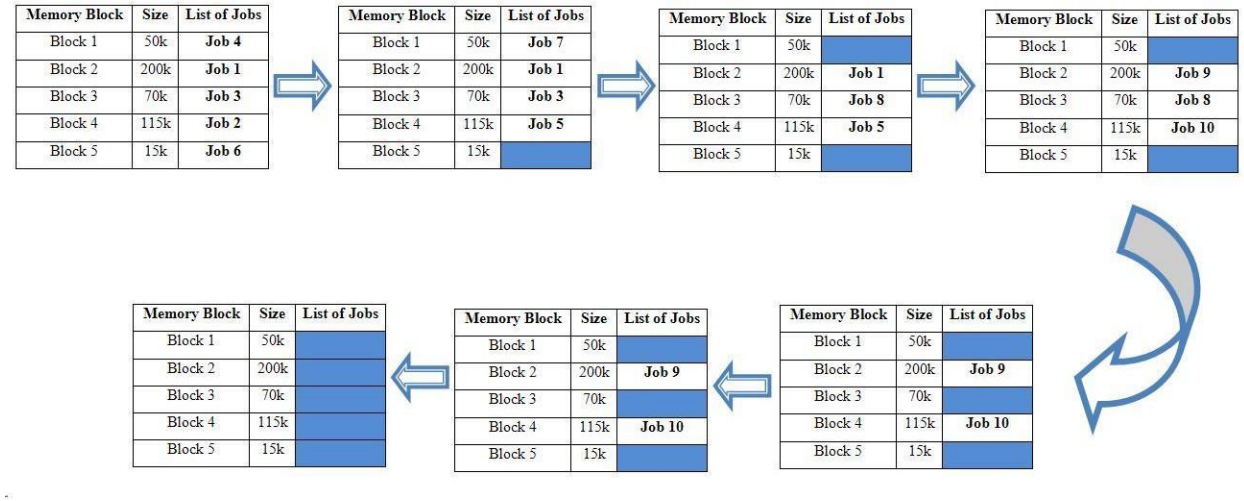
It is slower and may even tend to fill up memory with tiny useless holes.

<u>Worst fit</u>

In worst fit approach is to locate largest available free portion so that the portion left will be big enough to be useful. It is the reverse of best fit.

Worst-fit memory allocation is opposite to best-fit. It allocates free available block to the new job and it is not the best choice for an actual system. In the illustration, on the first cycle, job 5 is in waiting queue while job 1 to job 4 and job 6 are the jobs to be first process. After then, job 5 occupies the free block replacing job 2. Block 5 is now free to accommodate the next job which is job 8 but since the size in block 5 is not enough for job 8, so job 8 is in waiting queue. Then on the next cycle, block 3 accommodate job 8 while job 1 and job 5 remain on their memory block. In this cycle, there are 2 memory blocks are free. In the fourth cycle, only job 8 on block 3 remains while job 1 and job 5 are respectively replace by job 9 and job 10. Just the same in the previous cycle, there are still two free memory blocks. At fifth cycle, job 8 finish its job while the job 9 and job 10 are still on block 2 and block 4 respectively and there is additional memory block free. The same scenario happen on the sixth cycle. Lastly, on the seventh cycle, both job 9

and job 10 finish its process and in this cycle, all jobs are successfully process and executed. And all the memory blocks are now free.



Advantage

Reduces the rate of production of small gaps.

Disadvantage

If a process requiring larger memory arrives at a later stage then it cannot be accommodated as the largest hole is already split and occupied.

Prepared by :Mrs. S. P. Kakade	Approved by: Head of Department