SYSTEM PROGRAMMING LABORATORY

(CSX-326)

LAB PRACTICALS RECORD



Department of Computer Science & Engineering

Dr. B R Ambedkar National Institute of Technology

Jalandhar-144011,Punjab (India)

**LIST OF PRACTICALS**

|  |  |
| --- | --- |
| **S.No** | **Contents** |
| 1. | Introduction to CPU SIM |
| 2. | Introduction to SIC |
| 3. | Implement Linear Search |
| 4. | Implement Binary Search |
| 5. | Implement Hash Table |
| 6. | Implement Linked List and its operations |
| 7. | Implement Pass 1 and Pass 2 of Assembler |
| 8. | Implement Program for Macro Expansion |
| 9. | Implement Top Down Parser |
| 10. | Implement Bottom Up Parser |
| 11. | Implement Heapsort. |
| 12. | Implement program for linkers. |

**1. Introduction to CPU Simulation**

CPU Sim is a Java application that allows users to design simple computer CPUs at the microcode level and to run machine-language or assembly-language programs on those CPUs through simulation. It can be used to simulate a variety of architectures, including accumulator-based, RISC-like, or stack-based (such as the JVM) architectures. It is a useful tool for instructors who want their students to get hands-on exposure to a variety of architectures and to get a chance to design and implement their own architectures and write programs in machine language and assembly language for their architectures.

The CPU Sim application is a fully-integrated development environment that includes the following features.

• Tools for designing a CPU at the register-transfer level:

• Dialogs for specifying the number and width of registers, register arrays, and RAMs.

• Dialogs for specifying the microinstructions (e.g., bit transfers between registers) that are used to implement the machine instructions

• A dialog for specifying the machine instructions, including:

• the number of bits in each instruction

• the opcode value and the number of bits the opcode occupies

• the number of the operands and the properties of each operand

• the semantics of each instruction (as specified by a sequence of microinstructions)

• A text editor with syntax highlighting for writing assembly language programs

• An assembler for converting assembly programs into machine code for the user's CPU.

• A debugger for stepping forward and backward through the execution of such programs, inspecting and optionally changing the machine state after each step.

CPU SIM is an interactive low-level computer simulation package that runs on the Macintosh computer. Users of the package specify the details of the CPU to be simulated, including the register, i/o channels, main memory, the microinstruction set, machine instructions, and assembly language instructions. User can then create machine or assembly language programs using the built-in text editor and assembler and can run their programs in the simulator. It is possible to step through the execution of a program one machine instruction at a time. Also, users may choose to edit the contents of any component and then continue execution. The main window displays the state of the machine at every step.

**2. SIC - Simplified Instructional Computer**

Simplified Instructional Computer (SIC) is a hypothetical computer that has hardware features which are often found in real machines. There are two versions of this machine:

1. SIC standard Model

2. SIC/XE(extra equipment or expensive)

Object program for SIC can be properly executed on SIX/XE which is known as upward compatability.

SIC Machine Architecture/Components –

1. Memory –

• Memory is byte addressable that is words are addressed by location of their lowest numbered byte.

• There are 2^15 bytes in computer memory (1 byte = 8 bits)

3 consecutive byte = 1 word (24 bits = 1 word)

2. Registers –

There are 5 registers in SIC. Every register has an address associated with it known as register number. Size of each register is 4 bytes. On basis of register size, integer size is dependent.

I. A(Accumulator-0): It is used for mathematical operations.

II. X(Index Register-1): It is used for addressing.

III. L(Linkage Register-2): It stores the return address of instruction in case of subroutines.

IV. PC(Program Counter-8): It holds the address of next instruction to be executed.

V. SW(Status Word-9): It contains the variety of information

Status Word Register:

• mode bit refers to user mode(value=0) or supervising mode(value=1). It occupies 1 bit.[0]

• state bit refers whether process is in running state(value=0) or idle state(value=1). It also occupies 1 bit.[1]

• id bit refers to process id(PID). It occupies 3 bits.[2-5]

• CC bit refers to condition code i.e. It tells whether device is ready or not. It occupies 2 bits.[6-7]

Mask bit refers to interrupt mask. It occupies 4 bits.[8-11]

• X refers to unused bit. It also occupies 4 bits.[12-15]

• ICode refers to interrupt code i.e. Interrupt Service Routine. It occupies the remaining bits.[16-23]

3. Data Format –

• Integers are represented by 24 bit.

• Negative numbers are represented in 2’s complement.

• Characters are represented by 8 bit ASCII value.

• No floating point representation is available.

4. Instruction Format –

All instructions in SIC have 24 bit format.

• If x=0 it means direct addressing mode.

• If x=1 it means indexed addressing mode.

5. Instruction Set –

• Load And Store Instructions: To move or store data from accumulator to memory or vice-versa. For example LDA, STA, LDX, STX etc.

• Comparison Instructions: Used to compare data in memory by contents in accumulator. For example COMP data.

• Arithmetic Instructions: Used to perform operations on accumulator and memory and store result in accumulator. For example ADD, SUB, MUL, DIV etc.

• Conditional Jump: compare the contents of accumulator and memory and performs task based on conditions. For example JLT, JEQ, JGT

• Subroutine Linkage: Instructions related to subroutines. For example JSUB, RSUB

6. Input and Output –

It is performed by transferring 1 byte at a time from or to rightmost 8 bits of accumulator. Each device has 8 bit unique code.

There are 3 I/O instructions:

• Test Device (TD) tests whether device is ready or not. Condition code in Status Word Register is used for this purpose. If cc is < then device is ready otherwise device is busy.

• Read data(RD) reads a byte from device and stores in register A.

• Write data(WD) writes a byte from register A to the device.

**3. To implement Simple Linear Search**

Code:-

#include <iostream>

using namespace std;

int LinearSearch(int arr[], int l, int r, int x)

{

int index = -1;

for(int i=0;i<r;i++)

{

if(arr[i]==x)

return i;

}

return -1;

}

int main(void)

{

int arr[] = { 2, 3, 4, 10, 40 };

int x = 10;

int n = sizeof(arr) / sizeof(arr[0]);

int result = LinearSearch(arr, 0, n - 1, x);

(result == -1) ? cout << "Element is not present in array": cout << "Element is present at index " << result;

return 0;

}

OUTPUT

Element is present at index 3.

**4. To implement Binary Search algorithm.**

Code:-

#include <iostream>

using namespace std;

int binarySearch(int arr[], int l, int r, int x)

{

if (r >= l) {

int mid = l + (r - l) / 2;

if (arr[mid] == x)

return mid;

if (arr[mid] > x)

return binarySearch(arr, l, mid - 1, x);

return binarySearch(arr, mid + 1, r, x);

}

return -1;

}

int main(void)

{

int arr[] = { 2, 3, 4, 10, 40 };

int x = 10;

int n = sizeof(arr) / sizeof(arr[0]);

int result = binarySearch(arr, 0, n - 1, x);

(result == -1) ? cout << "Element is not present in array": cout << "Element is present at index " << result;

return 0;

}

OUTPUT

Element is present at index 3.

**5. To implement Hash Table .**

Code:-

#include <bits/stdc++.h>

using namespace std;

struct value

{

int data;

value \*left=NULL,\*down=NULL;

}\*ds=NULL;

int hash\_func(int x)

{

return x%10;

}

bool check\_collision(int x)

{

value \*lists=ds;

for(;lists;lists=lists->down)

{

if(lists->data==x)

{

return true;

break;

}

}

return false;

}

void inserts(int val,int hash\_val)

{

value\* lists=ds;

//input a value node

value\* node=new value();

node->data=hash\_val;

if(lists==NULL)

{

node->down=lists;

lists=node;

}

else

{

//adjusting links;

while(lists->data<hash\_val&&lists->down!=NULL)

lists=lists->down;

//insert

node->down=lists->down;

lists->down=node;

}

//Inserting the number into the chain

value\* num=new value();

num->data=val;

if(node->left==NULL)

{

node->left=num;

// num->left=node;

}

else

{

num->left=node->left;

node->left=num;

}

// return lists;

}

int searchs(int val,int hashs)

{

value\* lists=ds;

if(lists==NULL)

{

return -1;

}

else

{

//adjusting links;

while(lists->data<hashs&&lists->down!=NULL)

lists=lists->down;

if(lists==NULL||lists->data!=hashs)

return -1;

else

{

value\* links=lists->left;

while(links->data!=val&&links->left!=NULL)

{

links=links->left;

}

if(links==NULL)

return -1;

else

return 1;

}

}

}

main()

{

int c,input,hashs,status;

bool collision;

while(1)

{

collision=false;

cout<<"\nEnter your choice : \n1. Insert\n2. Searching\n3. Exit\n";

cin>>c;

switch(c)

{

case 1 : cout<<"Enter the value ";

cin>>input;

hashs=hash\_func(input);

collision=check\_collision(hashs);

if(collision==false)

{

inserts(input,hashs);

}

else

{

cout<<"\nCollision !!!";

}

break;

case 2 : cout<<"Enter the value that you want to search : ";

cin>>input;

hashs=hash\_func(input);

status=searchs(input,hashs);

if(status==-1)

{

cout<<"Value not present !!!";

}

else

{

cout<<"Value is present ";

}

break;

case 3 : exit(1);

default: cout<<"Invalid Input";

}

}

}

Output:-

Enter the no. to be searched:-10000

Value is not present!!!

**6. To implement Linked List.**

Code:-

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

using namespace std;

struct node

{

int info;

struct node \*link;

}\*start=NULL;

int main()

{

void insertbeg();

void insertpos();

void insertend();

void searchele();

void deleteele();

void display();

int kmn,choice;

while(1)

{

cout<<"\nEnter your choice\n1.Insert a node at the beginning\n2.Insert a node at a position\n3.Insert a node at the end\n4.Search a node\n5.Delete a node\n6.Display\n7.Exit\n\n";

cin>>choice;

switch(choice)

{

case 1 : insertbeg();break;

case 2 : insertpos();break;

case 3 : insertend();break;

case 4 : searchele();break;

case 5 : deleteele();break;

case 6 : display();break;

case 7 : exit(1);

default : cout<<"\nWrong choice entered by u !!!\n";break;

}

}

return 0;

}

void insertbeg()

{

struct node \*tmp,\*p;

int data;

//tmp=(struct node \*)malloc(sizeof(struct node));

tmp=new node;

if(tmp==NULL)

{

cout<<"\nNo space available";

exit(1);

}

cout<<"\nEnter the data element\t";

cin>>data;

tmp->info=data;

tmp->link=NULL;

if(start==NULL)

{

tmp->link=start;

start=tmp;

}

else

{

p=start;

start=tmp;

tmp->link=p;

}

}

void insertpos()

{

int pos,data,coun=1;

struct node \*p=start;

cout<<"\nEnter the data element you want to enter\t";

cin>>data;

cout<<"\nEnter the data element position\t";

cin>>pos;

while(p!=NULL)

{

if(coun==pos)

{break;}

coun++;

}

}

void insertend()

{

node \*tmp=new node;

cout<<"\nEnter the data element : ";

cin>>tmp->info;

node \*p=start;

while(p->link!=NULL)

p=p->link;

tmp->link=NULL;

p->link=tmp;

}

void searchele()

{

struct node \*ptr;

int coun=1,data;

cout<<"\nEnter the data element you want to search\t";

cin>>data;

ptr=start;

while(ptr!=NULL)

{

if(ptr->info==data)

{ break;}

ptr=ptr->link;

coun++;

}

cout<<"\nData element found at location "<<coun;

}

void deleteele()

{

int d;

cout<<"\nEnter the element that you want to delete : ";

cin>>d;

node \*p=start,\*q;

if(start->info==d)

{

q=start;

start=start->link;

free(q);

}

else

{

while(!p)

{

if(d==p->info)

{

q->link=q->link->link;

free(p);

break;

}

q=p;

p=p->link;

}

}

}

void display()

{

struct node \*ptr;

ptr=start;

cout<<"\nLinked List : ";

while(ptr!=NULL)

{

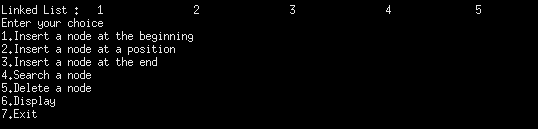
cout<<"\t"<<ptr->info<<"\t";

ptr=ptr->link;

}

}

OUTPUT



7**. Implement Pass 1& Pass2 of Assembler**

Code:-

#include <bits/stdc++.h>

using namespace std;

#define MAX 100

//Symbol Table

set <string> symbol;

int sym\_tab\_add[MAX]={0};

int stptr=0;

//Literal Table

int lit\_tab[MAX][2]={0};

int litptr=0;

//string pooltable[MAX][2]={0};

int pt=0;

int loc\_counter[MAX]={0};

int lc=0;

int AREG=0,BREG=0,CREG=0;

void Pass1(string str)

{

int i=0,j;

string str2="";

while(i<str.length())

{

string tmp="";

for(j=i;j<str.length();j++)

{

if(str[j]!=' ')

tmp=tmp+str[j];

else

break;

}

//cout<<tmp<<" ";

i=i+(j-i+1);

//Pass-1 Interpretation

//START

if(tmp.compare("START")==0)

{

//reading the next term

int q=0;

for(j=i;j<str.length();j++)

{

if(str[j]!=' ')

q=q\*10+(str[j]-48);

else

break;

}

loc\_counter[lc++]=q;

//Output

cout<<"\n( AD, 01 ) , ( C, "<<q<<" )";

}

//END

if(tmp.compare("END")==0)

{

cout<<"\n( AD, 02 ) ";

return;

}

//READ

if(tmp.compare("READ")==0)

{

cout<<"\n( IS, 01 ) , ";

loc\_counter[lc]=loc\_counter[lc-1]+1;

lc++;

//reading the next term

string tmp="";

for(j=i;j<str.length();j++)

{

if(str[j]!=' ')

tmp=tmp+str[j];

else

break;

}

//Put it into the symbol table

if(tmp.compare("AREG")!=0&&tmp.compare("BREG")!=0&&tmp.compare("CREG")!=0&&symbol.find(tmp)==symbol.end())

{

//symbol table entry

symbol.insert(tmp);

sym\_tab\_add[stptr]=loc\_counter[lc-1];

cout<<"( ST, "<<stptr<<" ) , ";

stptr++;

}

else

{

if(tmp.compare("AREG")==0)

{

cout<<"( R,1 ) , ";

}

else if(tmp.compare("BREG")==0)

{

cout<<"( R,2 ) , ";

}

else if(tmp.compare("CREG")==0)

{

cout<<"( R,3 ) , ";

}

else

cout<<"( ST, "<<stptr-1<<" ) , ";

}

}

//MOVER

if(tmp.compare("MOVER")==0)

{

cout<<"\n( IS, 02 ) , ";

loc\_counter[lc]=loc\_counter[lc-1]+1;

lc++;

//reading the first term

string tmp1="";

for(j=i;j<str.length();j++)

{

if(str[j]!=',')

tmp1=tmp1+str[j];

else

break;

}

i=i+(j-i+1);

//reading the second term

string tmp2="";

for(j=i;j<str.length();j++)

{

if(str[j]!=',')

tmp2=tmp2+str[j];

else

break;

}

if(tmp1.compare("AREG")!=0&&tmp1.compare("BREG")!=0&&tmp1.compare("CREG")!=0&&symbol.find(tmp1)==symbol.end())

{

//symbol table entry

symbol.insert(tmp1);

sym\_tab\_add[stptr]=loc\_counter[lc-1];

cout<<"( ST, "<<stptr<<" ) , ";

stptr++;

}

else

{

if(tmp1.compare("AREG")==0)

{

cout<<"( R,1 ) , ";

}

else if(tmp1.compare("BREG")==0)

{

cout<<"( R,2 ) , ";

}

else if(tmp1.compare("CREG")==0)

{

cout<<"( R,3 ) , ";

}

else

cout<<"( ST, "<<stptr-1<<" ) , ";

}

if(tmp2.compare("AREG")!=0&&tmp2.compare("BREG")!=0&&tmp2.compare("CREG")!=0&&symbol.find(tmp2)==symbol.end())

{

//symbol table entry

symbol.insert(tmp2);

sym\_tab\_add[stptr]=loc\_counter[lc-1];

cout<<"( ST, "<<stptr<<" ) , ";

stptr++;

}

else

{

if(tmp2.compare("AREG")==0)

{

cout<<"( R,1 ) , ";

}

else if(tmp2.compare("BREG")==0)

{

cout<<"( R,2 ) , ";

}

else if(tmp2.compare("CREG")==0)

{

cout<<"( R,3 ) , ";

}

else

cout<<"( ST, "<<stptr-1<<" ) , ";

}

}

//MOVEM

if(tmp.compare("MOVEM")==0)

{

cout<<"\n( IS, 03 ) , ";

loc\_counter[lc]=loc\_counter[lc-1]+1;

lc++;

//reading the first term

string tmp1="";

for(j=i;j<str.length();j++)

{

if(str[j]!=',')

tmp1=tmp1+str[j];

else

break;

}

i=i+(j-i+1);

//reading the second term

string tmp2="";

for(j=i;j<str.length();j++)

{

if(str[j]!=',')

tmp2=tmp2+str[j];

else

break;

}

if(tmp1.compare("AREG")!=0&&tmp1.compare("BREG")!=0&&tmp1.compare("CREG")!=0&&symbol.find(tmp1)==symbol.end())

{

//symbol table entry

symbol.insert(tmp1);

sym\_tab\_add[stptr]=loc\_counter[lc-1];

cout<<"( ST, "<<stptr<<" ) , ";

stptr++;

}

else

{

if(tmp1.compare("AREG")==0)

{

cout<<"( R,1 ) , ";

}

else if(tmp1.compare("BREG")==0)

{

cout<<"( R,2 ) , ";

}

else if(tmp1.compare("CREG")==0)

{

cout<<"( R,3 ) , ";

}

else

cout<<"( ST, "<<stptr-1<<" ) , ";

}

if(tmp2.compare("AREG")!=0&&tmp2.compare("BREG")!=0&&tmp2.compare("CREG")!=0&&symbol.find(tmp2)==symbol.end())

{

//symbol table entry

symbol.insert(tmp2);

sym\_tab\_add[stptr]=loc\_counter[lc-1];

cout<<"( ST, "<<stptr<<" ) , ";

stptr++;

}

else

{

if(tmp2.compare("AREG")==0)

{

cout<<"( R,1 ) , ";

}

else if(tmp2.compare("BREG")==0)

{

cout<<"( R,2 ) , ";

}

else if(tmp2.compare("CREG")==0)

{

cout<<"( R,3 ) , ";

}

else

cout<<"( ST, "<<stptr-1<<" ) , ";

}

}

//ADD

if(tmp.compare("ADD")==0)

{

cout<<"\n( IS, 04 ) , ";

loc\_counter[lc]=loc\_counter[lc-1]+1;

lc++;

//reading the first term

string tmp1="";

for(j=i;j<str.length();j++)

{

if(str[j]!=',')

tmp1=tmp1+str[j];

else

break;

}

i=i+(j-i+1);

//reading the second term

string tmp2="";

for(j=i;j<str.length();j++)

{

if(str[j]!=',')

tmp2=tmp2+str[j];

else

break;

}

if(tmp1.compare("AREG")!=0&&tmp1.compare("BREG")!=0&&tmp1.compare("CREG")!=0&&symbol.find(tmp1)==symbol.end())

{

//symbol table entry

symbol.insert(tmp1);

sym\_tab\_add[stptr]=loc\_counter[lc-1];

cout<<"( ST, "<<stptr<<" ) , ";

stptr++;

}

else

{

if(tmp1.compare("AREG")==0)

{

cout<<"( R,1 ) , ";

}

else if(tmp1.compare("BREG")==0)

{

cout<<"( R,2 ) , ";

}

else if(tmp1.compare("CREG")==0)

{

cout<<"( R,3 ) , ";

}

else

cout<<"( ST, "<<stptr-1<<" ) , ";

}

if(tmp2.compare("AREG")!=0&&tmp2.compare("BREG")!=0&&tmp2.compare("CREG")!=0&&symbol.find(tmp2)==symbol.end())

{

//symbol table entry

symbol.insert(tmp2);

sym\_tab\_add[stptr]=loc\_counter[lc-1];

cout<<"( ST, "<<stptr<<" ) , ";

stptr++;

}

else

{

if(tmp2.compare("AREG")==0)

{

cout<<"( R,1 ) , ";

}

else if(tmp2.compare("BREG")==0)

{

cout<<"( R,2 ) , ";

}

else if(tmp2.compare("CREG")==0)

{

cout<<"( R,3 ) , ";

}

else

cout<<"( ST, "<<stptr-1<<" ) , ";

}

}

//MULT

if(tmp.compare("MULT")==0)

{

cout<<"\n( IS, 05 ) , ";

loc\_counter[lc]=loc\_counter[lc-1]+1;

lc++;

//reading the first term

string tmp1="";

for(j=i;j<str.length();j++)

{

if(str[j]!=',')

tmp1=tmp1+str[j];

else

break;

}

i=i+(j-i+1);

//reading the second term

string tmp2="";

for(j=i;j<str.length();j++)

{

if(str[j]!=',')

tmp2=tmp2+str[j];

else

break;

}

if(tmp1.compare("AREG")!=0&&tmp1.compare("BREG")!=0&&tmp1.compare("CREG")!=0&&symbol.find(tmp1)==symbol.end())

{

//symbol table entry

symbol.insert(tmp1);

sym\_tab\_add[stptr]=loc\_counter[lc-1];

cout<<"( ST, "<<stptr<<" ) , ";

stptr++;

}

else

{

if(tmp1.compare("AREG")==0)

{

cout<<"( R,1 ) , ";

}

else if(tmp1.compare("BREG")==0)

{

cout<<"( R,2 ) , ";

}

else if(tmp1.compare("CREG")==0)

{

cout<<"( R,3 ) , ";

}

else

cout<<"( ST, "<<stptr-1<<" ) , ";

}

if(tmp2.compare("AREG")!=0&&tmp2.compare("BREG")!=0&&tmp2.compare("CREG")!=0&&symbol.find(tmp2)==symbol.end())

{

//symbol table entry

symbol.insert(tmp2);

sym\_tab\_add[stptr]=loc\_counter[lc-1];

cout<<"( ST, "<<stptr<<" ) , ";

stptr++;

}

else

{

if(tmp2.compare("AREG")==0)

{

cout<<"( R,1 ) , ";

}

else if(tmp2.compare("BREG")==0)

{

cout<<"( R,2 ) , ";

}

else if(tmp2.compare("CREG")==0)

{

cout<<"( R,3 ) , ";

}

else

cout<<"( ST, "<<stptr-1<<" ) , ";

}

}

//COMP

if(tmp.compare("COMP")==0)

{

cout<<"\n( IS, 06 ) , ";

loc\_counter[lc]=loc\_counter[lc-1]+1;

lc++;

//reading the first term

string tmp1="";

for(j=i;j<str.length();j++)

{

if(str[j]!=',')

tmp1=tmp1+str[j];

else

break;

}

i=i+(j-i+1);

//reading the second term

string tmp2="";

for(j=i;j<str.length();j++)

{

if(str[j]!=',')

tmp2=tmp2+str[j];

else

break;

}

if(tmp1.compare("AREG")!=0&&tmp1.compare("BREG")!=0&&tmp1.compare("CREG")!=0&&symbol.find(tmp1)==symbol.end())

{

//symbol table entry

symbol.insert(tmp1);

sym\_tab\_add[stptr]=loc\_counter[lc-1];

cout<<"( ST, "<<stptr<<" ) , ";

stptr++;

}

else

{

if(tmp1.compare("AREG")==0)

{

cout<<"( R,1 ) , ";

}

else if(tmp1.compare("BREG")==0)

{

cout<<"( R,2 ) , ";

}

else if(tmp1.compare("CREG")==0)

{

cout<<"( R,3 ) , ";

}

else

cout<<"( ST, "<<stptr-1<<" ) , ";

}

if(tmp2.compare("AREG")!=0&&tmp2.compare("BREG")!=0&&tmp2.compare("CREG")!=0&&symbol.find(tmp2)==symbol.end())

{

//symbol table entry

symbol.insert(tmp2);

sym\_tab\_add[stptr]=loc\_counter[lc-1];

cout<<"( ST, "<<stptr<<" ) , ";

stptr++;

}

else

{

if(tmp2.compare("AREG")==0)

{

cout<<"( R,1 ) , ";

}

else if(tmp2.compare("BREG")==0)

{

cout<<"( R,2 ) , ";

}

else if(tmp2.compare("CREG")==0)

{

cout<<"( R,3 )";

}

else

cout<<"( ST, "<<stptr-1<<" ) , ";

}

}

str2=tmp; //It is used to store the just next value of current word in a string

//Useful in case of DS or DC statements

//For Declarative Statements

if(tmp.compare("DS")==0)

{

cout<<"\n( ST, "<<stptr<<" ) , ";

cout<<"( DS, 01 ) , ";

//reading the next term

string tmp1="";

for(j=i;j<str.length();j++)

tmp1=tmp1+str[j];

//Put the value into the literal table

//Converting the string to int value

stringstream strs(tmp1);

int x=0;

strs >> x;

lit\_tab[litptr][0]=x;

lit\_tab[litptr][1]=loc\_counter[lc-1];

cout<<"( LT, "<<lc-1<<" )";

litptr++;

/\*

//Now the symbol will have the same loc add at the time of its initialisation

set <string> :: iterator it;

int pos=0;

for(it=symbol.begin();it!=symbol.end();it++)

{

if(str2.compare(\*it)==0)

break;

pos++;

}

if(pos!=stptr-1||stptr-1==0 )//means it not the new entry

{

sym\_tab\_add[pos]=loc\_counter[lc-1];

}

\*/

}

}

}

int main()

{

//OPCODES

string op\_name[7] = {"START","READ","MOVER","MOVEM","ADD","DS","END"};

string op\_class[7] = {"AD","IS","IS","IS","IS","DL","AD"};

string str[]={"START 101",

"READ N",

"MOVER BREG,N",

"MOVEM CREG,N",

"ADD CREG,BREG",

"MULT CREG,N",

"COMP CREG,N",

"N DS 1",

"END"};

cout<<"------- Intermediate Code -------\n";

for(int i=0;i<9;i++)

{

string s=str[i];

Pass1(s);

}

cout<<endl;

cout<<"\n------- Location counter -------\n";

for(int i=0;i<lc;i++)

cout<<loc\_counter[i]<<"\n";

cout<<"\n------- Symbol Table -------\n";

set <string> ::iterator it;

int i=0;

for( it=symbol.begin();it!=symbol.end();it++)

cout<<\*it<<" : "<<sym\_tab\_add[i++];

cout<<"\n------- Literal Table -------\n";

for(int i=0;i<litptr;i++)

cout<<lit\_tab[i][0]<<" : "<<lit\_tab[i][1]<<"\n";

cout<<"\n------- Opcode Table -------\n";

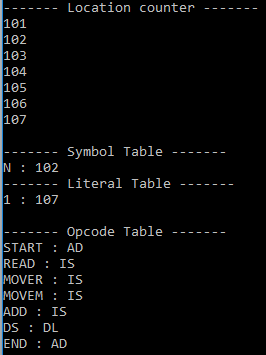
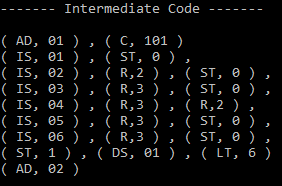
for(int i=0;i<7;i++)

cout<<op\_name[i]<<" : "<<op\_class[i]<<"\n";

return 0;

}

Output:-

**8. To implement program for macro expansion.**

Code:-

#include <bits/stdc++.h>

using namespace std;

#define MAX 50

struct macros

{

string name="";

string def="";

string par="";

string val="";

}m[MAX];

int ctr=0;

void Pass1(string str)

{

int i=0,j;

while(i<str.length())

{

string tmp="";

for(j=i;j<str.length();j++)

{

if(str[j]!=' ')

tmp=tmp+str[j];

else

break;

}

cout<<tmp<<" ";

i=i+(j-i+1);

//Pass-1 Interpretation

//Check whether a macro

if(tmp.compare("#define")==0)

{

string tmp="";

for(j=i;j<str.length();j++)

{

if(str[j]!=' ')

tmp=tmp+str[j];

else

break;

}

m[ctr].name=tmp;

i=i+(j-i+1);

tmp="";

for(j=i;j<str.length();j++)

{

if(str[j]!=' ')

tmp=tmp+str[j];

else

break;

}

m[ctr].val=tmp;

ctr++;

}

}

}

void Pass2(string str)

{

int pos=str.find("MAX");

if(pos>=0)

{

int l=(m[1].val).length();

str.replace(pos,pos+2,m[1].val);

//pos=str.find("MAX");

//cout<<"\nPresent\n";

}

pos=str.find("END");

if(pos>=0)

{

int l=(m[2].val).length();

str.replace(pos,pos+2,m[2].val);

//pos=str.find("MAX");

//cout<<"\nPresent\n";

}

pos=str.find("AREA(l1, l2)");

if(pos>=0)

{

int l=(m[0].val).length();

str.replace(pos,pos+11,m[0].val);

//pos=str.find("MAX");

//cout<<"\nPresent\n";

}

cout<<str;

return;

}

int main()

{

string str[]={"#define AREA(b) (b\*b)",

"#define MAX 5",

"#define END 200",

"#define TIME 0530",

"int main()",

"{",

"int l1 = 10, l2 = MAX;",

"area = AREA(l1, l2);",

"std:: cout << Area of rectangle is: << area \* MAX ;",

"return END ;",

"}"

};

cout<<"------- Pass1 -------\n";

for(int i=0;i<11;i++)

{

string s=str[i];

Pass1(s);

cout<<endl;

}

cout<<"\nAfter Pass - 1\n";

cout<<"\nMacro Names\n";

for(int i=0;i<ctr;i++)

{

cout<<m[i].name<<" : "<<m[i].val<<"\n";

}

cout<<"\nModified Code After Pass 2\n";

for(int i=4;i<11;i++)

{

string s=str[i];

Pass2(s);

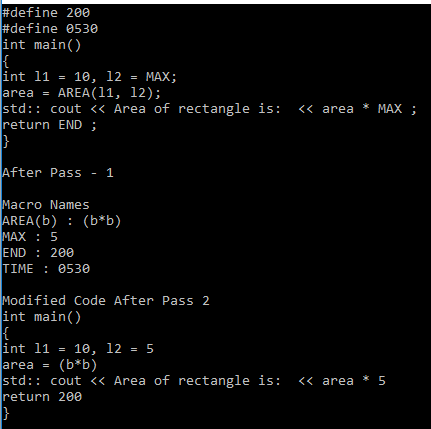
cout<<endl;

}

return 0;

}

Output:-



**9. To implement Top Down Parser.**

Code:-

#include<iostream>

#include<conio.h>

#include<string.h>

#include<stdlib.h>

using namespace std;

int main()

{

int table[5][4] = { {0,-1,-1,-1},

{-1,1,-1,2},

{3,-1,-1,-1},

{-1,2,4,2},

{5,-1,-1,-1} };

char tab[6][5] = { "TF\0","+TF\0","\0","VU\0","\*VU\0","I\0" };

char symbol,left[20],right[20],tok[4],csf[30] = "E",input[50],in[50];

int flag = 1,len1,ssm=0,row,col,loc = 0;

char brk(char\* , int &);

void leftright(char \*,char left[20],char right[20],int ssm );

//clrscr();

cout<<"enter ur expression :";

cin.getline(input,50);

len1 = strlen(input);

if (input[len1-1] != ';')

{

cout<<"please terminate the expression with a ';' ";

exit(0);

}

strcpy(input,strupr(input));

int j = 0;

for(int i = 0;i< len1 - 2; i++)

{

if (input[i] != ' ')

in[j++] = input[i];

}

in[j] = '\0';

cout<<csf<<endl;

symbol = brk(input,loc);

while (flag)

{

if (csf[ssm] == 'E') row = 0;

else if (csf[ssm] == 'F') row = 1;

else if (csf[ssm] == 'T') row = 2;

else if (csf[ssm] == 'U') row = 3;

else if (csf[ssm] == 'V') row = 4;

else row = -1;

if (symbol=='I') col = 0;

else if (symbol== '+') col = 1;

else if (symbol== '\*') col = 2;

else if (symbol == ';') col = 3;

else col = -1;

if (row == -1 || col == -1 || table[row][col] == -1)

{

cout<<endl<<input<<endl;

for(int i=0;i<2\*ssm;i++)

cout<<" ";

cout<<"^"<<endl;

cout<<"error in the expression "<<endl;

cout<<"symbol "<<symbol<<" is not valid in expression ";

exit(0);

}

strcpy(tok, tab[table[row][col]]);

leftright(csf,left,right,ssm);

strcat(left,tok);

strcat(left,right);

strcpy(csf,left);

if (symbol == tok[0])

{

ssm++;

symbol = brk(input,loc);

}

cout<<csf<<endl;

if (!strcmp(in,csf))

flag = 0;

}

cout<<"The Expression is valid";

}

char brk(char \*input, int &loc)

{

char symbol;

if (input[loc] == ' ')

loc++;

symbol = input[loc++];

return symbol;

}

void leftright(char \*csf,char left[20],char right[20],int ssm)

{

int i,len,j=0;

strcpy(left,"\0");

strcpy(right,"\0");

len = strlen(csf);

for(i = 0 ; i < ssm ; i++)

{

left[i] = csf[i];

}

left[i]='\0';

for(i = ssm + 1 ;i<=len;i++)

{

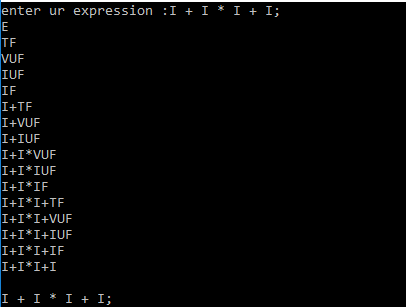
right[j++] = csf[i];

}

right[j] = '\0';

}

Output:-



**10. To implement Bottom Up Parser.**

Code:-

#include<conio.h>

#include<iostream>

#include<string.h>

using namespace std;

struct grammer{

char p[20];

char prod[20];

}g[10];

int main()

{

int i,stpos,j,k,l,m,o,p,f,r;

int np,tspos,cr;

cout<<"\nEnter Number of productions:";

cin>>np;

char sc,ts[10];

cout<<"\nEnter productions:\n";

for(i=0;i<np;i++)

{

cin>>ts;

strncpy(g[i].p,ts,1);

strcpy(g[i].prod,&ts[3]);

}

char ip[10];

cout<<"\nEnter Input:";

cin>>ip;

int lip=strlen(ip);

char stack[10];

stpos=0;

i=0;

//moving input

sc=ip[i];

stack[stpos]=sc;

i++;stpos++;

cout<<"\n\nStack\tInput\tAction";

do

{

r=1;

while(r!=0)

{

cout<<"\n";

for(p=0;p<stpos;p++)

{

cout<<stack[p];

}

cout<<"\t";

for(p=i;p<lip;p++)

{

cout<<ip[p];

}

if(r==2)

{

cout<<"\tReduced";

}

else

{

cout<<"\tShifted";

}

r=0;

//try reducing

getch();

for(k=0;k<stpos;k++)

{

f=0;

for(l=0;l<10;l++)

{

ts[l]='\0';

}

tspos=0;

for(l=k;l<stpos;l++) //removing first caharcter

{

ts[tspos]=stack[l];

tspos++;

}

//now compare each possibility with production

for(m=0;m<np;m++)

{

cr = strcmp(ts,g[m].prod);

//if cr is zero then match is found

if(cr==0)

{

for(l=k;l<10;l++) //removing matched part from stack

{

stack[l]='\0';

stpos--;

}

stpos=k;

//concatinate the string

strcat(stack,g[m].p);

stpos++;

r=2;

}

}

}

}

//moving input

sc=ip[i];

stack[stpos]=sc;

i++;stpos++;

}while(strlen(stack)!=1 && stpos!=lip);

if(strlen(stack)==1)

{

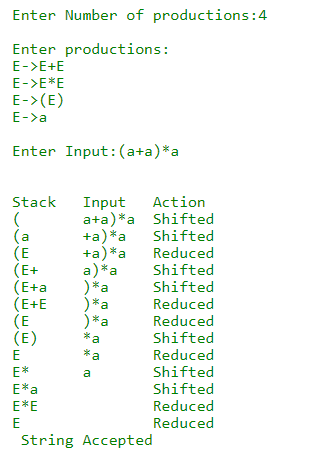
cout<<"\n String Accepted";

}

getch();

}

Output:



**11.To implement Program for Heapsort.**

Code:-

// C++ program for implementation of Heap Sort

#include <iostream>

using namespace std;

// To heapify a subtree rooted with node i which is

// an index in arr[]. n is size of heap

void heapify(int arr[], int n, int i)

{

int largest = i; // Initialize largest as root

int l = 2\*i + 1; // left = 2\*i + 1

int r = 2\*i + 2; // right = 2\*i + 2

// If left child is larger than root

if (l < n && arr[l] > arr[largest])

largest = l;

// If right child is larger than largest so far

if (r < n && arr[r] > arr[largest])

largest = r;

// If largest is not root

if (largest != i)

{

swap(arr[i], arr[largest]);

// Recursively heapify the affected sub-tree

heapify(arr, n, largest);

}

}

// main function to do heap sort

void heapSort(int arr[], int n)

{

// Build heap (rearrange array)

for (int i = n / 2 - 1; i >= 0; i--)

heapify(arr, n, i);

// One by one extract an element from heap

for (int i=n-1; i>=0; i--)

{

// Move current root to end

swap(arr[0], arr[i]);

// call max heapify on the reduced heap

heapify(arr, i, 0);

}

}

/\* A utility function to print array of size n \*/

void printArray(int arr[], int n)

{

for (int i=0; i<n; ++i)

cout << arr[i] << " ";

cout << "\n";

}

// Driver program

int main()

{

int arr[] = {12, 11, 13, 5, 6, 7}

int n = sizeof(arr)/sizeof(arr[0]);

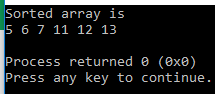
heapSort(arr, n);

cout << "Sorted array is \n";

printArray(arr, n);

}

Output:-



**12. To implement Program for Linker.**

Code:-

#include <bits/stdc++.h>

using namespace std;

void iostream\_header()

{

//printf and scanf

cout<<"int printf( const char\* format, ... );"<<endl;

cout<<"int scanf ( const char \* format, ... );"<<endl;

}

void headers(string s[],int n)

{

for(int i=0;i<13;i++)

{

string a=s[i],b="";

if(i==0)

{

if(a[0]=='#')

{

for(int j=11;a[j]!='>';j++)

b=b+a[i];

}

cout<<b<<"\n";

if(b.compare("iostream")==0)

{

iostream\_header();

}

}

}

}

main()

{

string program[]={"#include <iostream>",

"using namespace std;",

"",

"main()",

"{",

"int a,b,c;",

"cout<<Hello World;",

"",

"c=a+b;",

"cout<<c;",

"",

"return 0;",

"}"};

cout<<"Program before linking : \n";

for(int i=0;i<13;i++)

{

cout<<program[i]<<endl;

}

cout<<"\n-------------------------------------\n";

cout<<"\nProgram after linking : \n";

iostream\_header();

for(int i=1;i<13;i++)

{

cout<<program[i]<<endl;

}

}

Output:-

