**Practical 6**

**Implementation of 3 Address Code**

**Three address code**

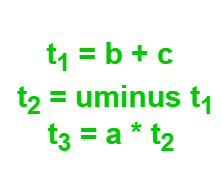
* Three-address code is an intermediate code. It is used by the optimizing compilers.
* In three-address code, the given expression is broken down into several separate instructions. These instructions can easily translate into assembly language.
* Each Three address code instruction has at most three operands. It is a combination of assignment and a binary operator.

**General representation –**

x = y op z

Where a, b or c represents operands like names, constants or compiler generated temporaries and op represents the operator

**Example-1:** Convert the expression a \* – (b + c) into three address code.



**Implementation of Three Address Code –**

There are 3 representations of three address code namely

1. Quadruple
2. Triples
3. Indirect Triples

**1. Quadruple –**

It is structure with consist of 4 fields namely op, arg1, arg2 and result. op denotes the operator and arg1 and arg2 denotes the two operands and result is used to store the result of the expression.

**Advantage –**

* Easy to rearrange code for global optimization.
* One can quickly access value of temporary variables using symbol table.

**Disadvantage –**

* Contain lot of temporaries.
* Temporary variable creation increases time and space complexity.

**2. Triples –**

This representation doesn’t make use of extra temporary variable to represent a single operation instead when a reference to another triple’s value is needed, a pointer to that triple is used. So, it consist of only three fields namely op, arg1 and arg2.

**Disadvantage –**

* Temporaries are implicit and difficult to rearrange code.
* It is difficult to optimize because optimization involves moving intermediate code. When a triple is moved, any other triple referring to it must be updated also. With help of pointer one can directly access symbol table entry.

**3. Indirect Triples –**

This representation makes use of pointer to the listing of all references to computations which is made separately and stored. Its similar in utility as compared to quadruple representation but requires less space than it. Temporaries are implicit and easier to rearrange code.

**Program**

#include<stdio.h>

#include<string.h>

void pm();

void plus();

void div();

int i,ch,j,l,addr=100;

char ex[10], exp[10] ,exp1[10],exp2[10],id1[5],op[5],id2[5];

void main()

{

clrscr();

while(1)

{

printf("\n1.assignment\n2.arithmetic\n3.relational\n4.Exit\nEnter the choice:");

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("\nEnter the expression with assignment operator:");

scanf("%s",exp);

l=strlen(exp);

exp2[0]='\0';

i=0;

while(exp[i]!='=')

{

i++;

}

strncat(exp2,exp,i);

strrev(exp);

exp1[0]='\0';

strncat(exp1,exp,l-(i+1));

strrev(exp1);

printf("Three address code:\ntemp=%s\n%s=temp\n",exp1,exp2);

break;

case 2:

printf("\nEnter the expression with arithmetic operator:");

scanf("%s",ex);

strcpy(exp,ex);

l=strlen(exp);

exp1[0]='\0';

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

{

if(exp[i]=='+'||exp[i]=='-')

{

if(exp[i+2]=='/'||exp[i+2]=='\*')

{

pm();

break;

}

else

{

plus();

break;

}

}

else if(exp[i]=='/'||exp[i]=='\*')

{

div();

break;

}

}

break;

case 3:

printf("Enter the expression with relational operator");

scanf("%s%s%s",&id1,&op,&id2);

if(((strcmp(op,"<")==0)||(strcmp(op,">")==0)||(strcmp(op,"<=")==0)||(strcmp(op,">=")==0)||(strcmp(op,"==")==0)||(strcmp(op,"!=")==0))==0)

printf("Expression is error");

else

{

printf("\n%d\tif %s%s%s goto %d",addr,id1,op,id2,addr+3);

addr++;

printf("\n%d\t T:=0",addr);

addr++;

printf("\n%d\t goto %d",addr,addr+2);

addr++;

printf("\n%d\t T:=1",addr);

}

break;

case 4:

exit(0);

}

}

}

void pm()

{

strrev(exp);

j=l-i-1;

strncat(exp1,exp,j);

strrev(exp1);

printf("Three address code:\ntemp=%s\ntemp1=%c%ctemp\n",exp1,exp[j+1],exp[j]);

}

void div()

{

strncat(exp1,exp,i+2);

printf("Three address code:\ntemp=%s\ntemp1=temp%c%c\n",exp1,exp[i+2],exp[i+3]);

}

void plus()

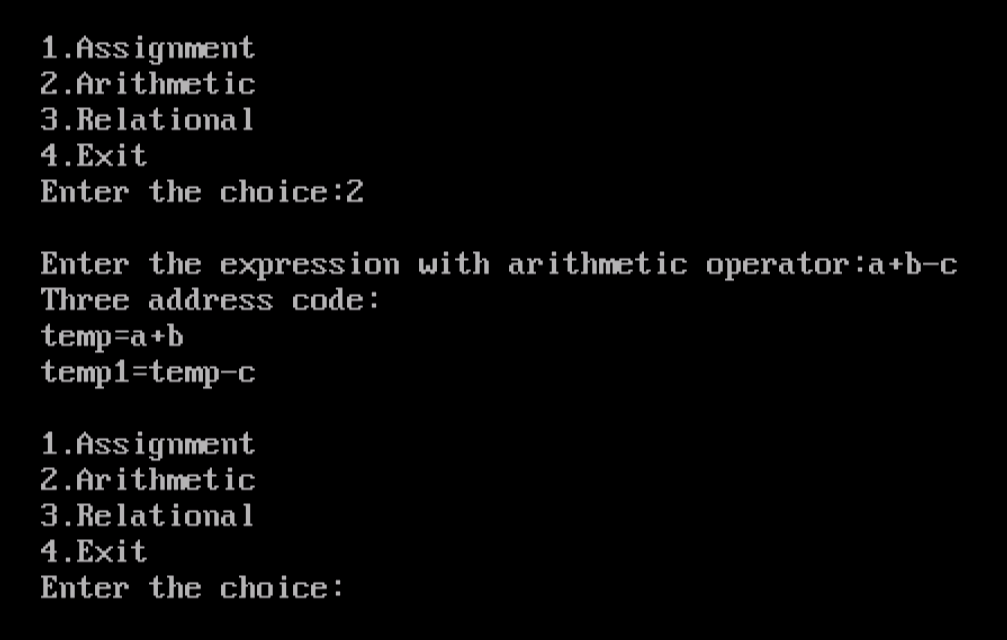
{

strncat(exp1,exp,i+2);

printf("Three address code:\ntemp=%s\ntemp1=temp%c%c\n",exp1,exp[i+2],exp[i+3]);

}

**Output**

****