**Ex. No:**

**TYPE CHECKING**

**Aim:**

To write a program to implement type checking

**Algorithm:**

1. Define symbol table with following attributes <symbol name, type>

2. Get the expression as the input from the user. It should be in the form

Operand1 <operator> Operand2

3. Read operand1 and retrieve the corresponding Type information from symbol table.

4. Read operand2 and retrieve the corresponding Type information from symbol table.

5. If type of both operands are same then print the Type as output

6. Otherwise print “TYPE MISMATCH”

**/\*TYPE CHECKING\*/**

#include<stdio.h>

#include<string.h>

struct symtab{

char symname[25];

char type[25];

}symtab;

struct symtab s1[20];

int findtype(char t, int n){

int i;

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

if(s1[i].symname[0]==t)

return i;

}

return 0;

}

void main() {

char s[20];

char flag[20]="//TypeCheck";

double op1,op2,interrslt;

int flag2=0,n,i;

int tr,top1,top2;

FILE \*fp1,\*fp2;

printf("Enter the number of identifiers : ");

scanf("%d",&n);

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

printf("Enter identifier name and its type : ");

scanf("%s",&s1[i].symname);

scanf("%s",&s1[i].type);

}

fp1 = fopen("input.txt","r");

fp2 = fopen("output.txt","w");

fscanf(fp1,"%s",s);

while(!feof(fp1)) {

if(strcmp(s,flag)==0) {

flag2 = 1;

}

if(flag2==1) {

fscanf(fp1,"%s",s);

tr = findtype(s[0],n);

top1 = findtype(s[2],n);

top2 = findtype(s[4],n);

if(strcmp(s1[tr].type,s1[top1].type)==0 && strcmp(s1[tr].type,s1[top2].type)==0) {

fprintf(fp2,"/\*VALID EXP\*/\n");

else

fprintf(fp2,"/\*TYPE MISMATCH\*/\n");

flag2=0;

}

fprintf(fp2,"%s\n",s);

fscanf(fp1,"%s",s);

}

fclose(fp1);

fclose(fp2);

}

**OUTPUT**

[student@localhost ~]$ cc typecheck.c

[student@localhost ~]$ ./a.out

Enter the number of identifiers : 4

Enter identifier name and its type : a in

Enter identifier name and its type : b in

Enter identifier name and its type : c in

Enter identifier name and its type : d float

**[student@localhost ~]$cat input.txt**

#include<stdio.h>

int main()

{

int a,b,c,d;

//TypeCheck

a=b+c;

//TypeCheck

b=c+d;

}

**[student@localhost ~]$cat Output.txt**

#include<stdio.h>

int

main()

{

int

a,b,c,d;

/\*VALID EXP\*/

a=b+c;

/\*TYPE MISMATCH\*/

b=c+d;

}

**Ex. No:**

**CODE OPTIMIZATION**

**Aim:**

To write a C program to implement Constant Folding (Code optimization Technique).

**Description:**

Constant folding is the process of recognizing and evaluating constant expressions at compile time rather than computing them at runtime. This improves the run-time performance and reduces the code size by avoiding evaluation at compile-time.

**Algorithm**

1. Read the input program from the file.

2. Check if the program contains expression with both the operands as constant.

3. If so, then evaluate the value of the expression and update it in the output file.

4. Else, Just update the same in the output file.

**/\*CODE OPTIMIZATION - CONSTANT FOLDING\*/**

#include<stdio.h>

#include<string.h>

void main() {

char s[20];

char flag[20]="//Constant";

char result,equal,operator;

double op1,op2,interrslt;

int a,flag2=0;

FILE \*fp1,\*fp2;

fp1 = fopen("input.txt","r");

fp2 = fopen("output.txt","w");

fscanf(fp1,"%s",s);

while(!feof(fp1)) {

if(strcmp(s,flag)==0) {

flag2 = 1;

}

if(flag2==1) {

fscanf(fp1,"%s",s);

result=s[0];

equal=s[1];

if(isdigit(s[2])&& isdigit(s[4])) {

if(s[3]=='+'||'-'||'\*'||'/') {

operator=s[3];

switch(operator) {

case '+':

interrslt=(s[2]-48)+(s[4]-48);

break;

case '-':

interrslt=(s[2]-48)-(s[4]-48);

break;

case '\*':

interrslt=(s[2]-48)\*(s[4]-48);

break;

case '/':

interrslt=(s[2]-48)/(s[4]-48);

break;

default:

interrslt = 0;

break;

}

fprintf(fp2,"/\*Constant Folding\*/\n");

fprintf(fp2,"%c = %lf\n",result,interrslt);

flag2 = 0;

}

} else {

fprintf(fp2,"Not Optimized\n");

fprintf(fp2,"%s\n",s);

}

} else {

fprintf(fp2,"%s\n",s);

}

fscanf(fp1,"%s",s);

}

fclose(fp1);

fclose(fp2);

}

**OUTPUT**

[student@localhost ~]$ cc codeopt.c

[student@localhost ~]$ ./a.out

**[student@localhost ~]$cat input.txt**

#include<stdio.h>

int main()

{

//Constant

a=2+4;

b=a+10;

}

**[student@localhost ~]$cat Output.txt**

#include<stdio.h>

int

main()

{

/\*Constant Folding\*/

a = 6.000000

b=a+10;

}

**Ex. No:**

**STORAGE ALLOCATION STRATEGIES**

**Aim:**

To write a program to implement storage allocation strategies (Heap,Stack,Static)

**Description:**

## Storage Allocation

Runtime environment manages runtime memory requirements for the following entities:

* **Code** : It is known as the text part of a program that does not change at runtime. Its memory requirements are known at the compile time.
* **Procedures** : Their text part is static but they are called in a random manner. That is why, stack storage is used to manage procedure calls and activations.
* **Variables** : Variables are known at the runtime only, unless they are global or constant. Heap memory allocation scheme is used for managing allocation and de-allocation of memory for variables in runtime.

## Static Allocation

In this allocation scheme, the compilation data is bound to a fixed location in the memory and it does not change when the program executes. As the memory requirement and storage locations are known in advance, runtime support package for memory allocation and de-allocation is not required.

**Algorithm-Static Allocation**

1. Assume some starting address(init\_addr=1000).
2. Repeat the following steps until all the variables in the declaration sections are processed

For each variable compute the address using the following formula,

addr=addr+width(variable)

store the variable in the computed address addr.

**Source Code:**

#include<stdio.h>

//#include<conio.h>

void strip(char \*s) {

char \*p2 = s;

while(\*s != '\0') {

if(\*s != '\t' && \*s != '\n' && \*s != '&' && \*s != '"') {

\*p2++ = \*s++;

} else {

++s;

}

}

\*p2 = '\0';

}

void main()

{

FILE \*fp;

int addr = 1000,flag1=0,flag=0;

char str[100][100];

char \*ch;

fp = fopen("strg.txt","r");

while(!feof(fp))

{

fread(str,sizeof(str),1,fp);

ch = strtok(str," ;");

while(ch != NULL)

{

strip(ch);

if(strcmp(ch,"int")==0 )

{

flag1 = 1;

flag++;

}

else if(strcmp(ch,"float")==0 )

{

flag1 = 2;

flag++;

}

else if(strcmp(ch,"long")==0)

{

flag1 = 3;

flag++;

}

else if(strcmp(ch,"char")==0 )

{

flag1 = 4;

flag++;

}

else if(strcmp(ch,"double")==0)

{

flag1 = 5;

flag++;

}

else

{

if(flag1>0)

{

if(flag1==1)

{

addr+=sizeof(int);

printf("%s %d\n",ch,addr);

}

else if(flag1==2)

{

addr+=sizeof(float);

//printf("%d",sizeof(double));

printf("%s %d\n",ch,addr);

}

else if(flag1==3)

{

addr+=sizeof(long);

printf("%s %d\n",ch,addr);

}

else if(flag1==4)

{

addr+=sizeof(char);

printf("%s %d\n",ch,addr);

}

else if(flag1==5)

{

addr+=sizeof(double);

printf("%s %d\n",ch,addr);

}

}

}

ch = strtok(NULL," ;");

}

}

}

***strg.txt***

int a;

double b;

long c;

**Output:**

[student@localhost ~]$ cc storage.c

[student@localhost ~]$ ./a.out

a 1004

b 1012

c 1016