///assignment 3

///MD Abdullah Al Nasim

///15.01.04.085

/\*we generate a modified token stream in accordance with the Symbol Table

for processing by the next phase, that is, Syntactic Analysis.

\*/

#include<bits/stdc++.h>

#define M 256

#define CHAIN\_LENGTH 32

using namespace std;

struct symbolTable

{

char siNo[M];

char name[M];

char idType[M];

char valueType[M];

char scope[M];

}SymTbl[M];

struct symbol\_info{

char \*name;

char \*classtype;

struct symbol\_info \*next;

} \*block[CHAIN\_LENGTH];

// Hashing position is calculated using sum of all character ascii values

// Then performing Modulo operation to go to any bucket from 0 to CHAIN\_LENGTH

int cHash(char\* name){

int idx = 0;

for(int i = 0; name[i]; ++i){

idx = idx + name[i];

}

return (idx % CHAIN\_LENGTH);

}

void cInsert(char\* name, char\* classtype){

int pos = cHash(name);

if( block[pos] == NULL ){

block[pos] = new symbol\_info();

block[pos]->name = name;

block[pos]->classtype = classtype;

block[pos]->next = NULL;

}

else{

symbol\_info\* newNode = new symbol\_info();

newNode->name = name;

newNode->classtype = classtype;

// pointer swap

symbol\_info\* nextNode = block[pos];

block[pos] = newNode;

newNode->next = nextNode;

}

}

// Go to certain chain through hashing since we know others will not contain the searched value.

// Next in that chain do a linear search on all element to see if it is there.

bool cSearch(char\* name, char\* classtype){

// Implement

int pos = cHash(name);

symbol\_info\* temp = block[pos];

while( temp != NULL ){

if( !strcmp( temp->name, name ) && !strcmp( temp->classtype, classtype ) ){

return true;

}

temp = temp->next;

}

return false;

}

// If the name and class type both matches then delete the element.

void cDelete(char\* name, char\* classtype){

int pos = cHash(name);

symbol\_info\* temp = block[pos];

if(temp == NULL) return;

// At head but no one to follow

if( temp->next == NULL && !strcmp( temp->name, name ) && !strcmp( temp->classtype, classtype ) ){

block[pos] = NULL;

}

// At head has followers

else if( !strcmp( temp->name, name ) && !strcmp( temp->classtype, classtype ) ){

block[pos] = temp->next;

}

else{

while( temp->next != NULL ){

if ( !strcmp( temp->next->name, name ) && !strcmp( temp->next->classtype, classtype ) ){

printf("FOUND IT %s : %s\n", temp->name, temp->classtype );

break;

}

temp = temp->next;

}

if( temp != NULL ){

symbol\_info\* found = temp->next;

temp->next = found->next;

delete(found);

}

}

}

// Update an old class type with a new one. Change based on your need.

void cUpdate(char\* name, char\* classtype, char\* updatedClasstype){

int pos = cHash(name);

symbol\_info\* temp = block[pos];

while( temp != NULL ){

if( !strcmp( temp->name, name ) && !strcmp( temp->classtype, classtype ) ){

temp->classtype = updatedClasstype;

return;

}

temp = temp->next;

}

}

// Print the symbol table chain wise.

void showSymbolTable(){

// Implement

for(int i = 0; i < CHAIN\_LENGTH; ++i){

printf("%d:", i);

// Do not modify the head

symbol\_info\* temp = block[i];

while( temp != NULL ){

printf("->[%s|%s]", temp->name, temp->classtype);

temp = temp->next;

}

printf("\n");

}

}

int showMenu(){

cout <<endl<< "Menu:\n";

cout << "=====\n";

string message = "Enter 1 to insert(name, class type)\n"

"Enter 2 to update(name, class type, new class type)\n"

"Enter 3 to search(name, class type)\n"

"Enter 4 to delete(name, class type)\n"

"Enter 5 to display the symbol table\n";

cout << message << "\n";

cout << "User Choice: ";

int choice;

scanf("%d", &choice);

return choice;

}

int main()

{

FILE\* input;

FILE\* output;

///STEP 1:(Bracket removed)

char c;

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

output = fopen("Output1.txt", "w");

while((c=fgetc(input))!=EOF)

{

if(c=='\n') fputc(' ', output);

else fputc(c, output);

}

fclose(input);

fclose(output);

printf("INPUT FILE: \n\n");

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

while((c=fgetc(input))!=EOF) printf("%c",c);

printf("\n\n");

input = fopen("Output1.txt", "r");

output = fopen("Output2.txt", "w");

while((c=fgetc(input))!=EOF)

{

if(c=='[' || c==']') fputc(' ', output);

else fputc(c, output);

}

fclose(input);

fclose(output);

input = fopen("Output2.txt", "r");

//printf("Brackets Removed: \n\n");

//while((c=fgetc(input))!=EOF) printf("%c",c);

//printf("\n\n\n");

///STEP 1 final:(ID Isolated)

input = fopen("Output2.txt", "r");

output = fopen("Output3.txt", "w");

char str[M];

while(fscanf(input, "%s", &str)!=EOF)

{

fprintf(output, "[", str);

if(strcmp(str,"id")==0) fprintf(output, "%s ", str);

fscanf(input, "%s", &str);

fprintf(output, "%s]", str);

}

fclose(input);

fclose(output);

printf("STEP 1-IDs Isolated: \n\n");

input = fopen("Output3.txt", "r");

output = fopen("Output4.txt", "w");

while((c=fgetc(input))!=EOF) printf("%c",c);

fclose(input);

fclose(output);

///STEP 1-Extra:(Bracket removed after removed kw,op etc)

input = fopen("Output3.txt", "r");

output = fopen("Output4.txt", "w");

while((c=fgetc(input))!=EOF)

{

if(c=='[' || c==']')fputc(' ', output);

//else if(c==']') fputc(' ', output);

else fputc(c, output);

}

fclose(input);

fclose(output);

//printf("\n\nBrackets Removed For Easier Calculation: \n\n");

input = fopen("Output4.txt", "r");

//while((c=fgetc(input))!=EOF) printf("%c",c);

fclose(input);

///STEP 2:Symbol Table

input = fopen("Output4.txt", "r");

output = fopen("Output5.txt", "w");

char name[M] = "global";

int id = 1;

while(fscanf(input, "%s", &str)!=EOF)

{

if(strcmp(str,"float")==0)

{

fprintf(output,"%s ",&str);

snprintf( SymTbl[id].siNo, 2, "%d", id );//buffer size for integer 1byte= 4bit= 15

snprintf( SymTbl[id].idType, 20, "%s", str );//buffer size 4 byte for 4 character in max idType = "func"

fscanf(input, "%s", &str);

fprintf(output,"%s ",&str);

fscanf(input, "%s", &str);

snprintf( SymTbl[id].name, 20, "%s", str );

fscanf(input, "%s", &str);

fprintf(output,"%s ",SymTbl[id].siNo);

fprintf(output,"%s ",&str);

snprintf( SymTbl[id].scope, 20, "%s", name );

if(strcmp(str,"(")==0)

{

snprintf( name, 4, "%s", SymTbl[id].name );

snprintf( SymTbl[id].idType, 20, "func", name );

}

else snprintf( SymTbl[id].idType, 20, "var", name );

id++;

}

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

{

fprintf(output,"%s ",&str);

snprintf( SymTbl[id].siNo, 2, "%d", id );

snprintf( SymTbl[id].valueType, 20, "%s", str );

fscanf(input, "%s", &str);

fprintf(output,"%s ",&str);

fscanf(input, "%s", &str);

snprintf( SymTbl[id].name, 20, "%s", str );

fscanf(input, "%s", &str);

fprintf(output,"%s ",SymTbl[id].siNo);

fprintf(output,"%s ",&str);

snprintf( SymTbl[id].scope, 20, "%s", name );

if(strcmp(str,"(")==0)

{

snprintf( name,20, "%s", SymTbl[id].name );

snprintf( SymTbl[id].idType, 20, "func", name );

}

else snprintf( SymTbl[id].idType, 20, "var", name );

id++;

}

else if(strcmp(str,"int")==0)

{

fprintf(output,"%s ",&str);

snprintf( SymTbl[id].siNo, 2, "%d", id );

snprintf( SymTbl[id].valueType, 20, "%s", str );

fscanf(input, "%s", &str);

fprintf(output,"%s ",&str);

fscanf(input, "%s", &str);

snprintf( SymTbl[id].name, 20, "%s", str );

fscanf(input, "%s", &str);

fprintf(output,"%s ",SymTbl[id].siNo);

fprintf(output,"%s ",&str);

snprintf( SymTbl[id].scope, 20, "%s", name );

if(strcmp(str,"(")==0)

{

snprintf( name, 20, "%s", SymTbl[id].name );

snprintf( SymTbl[id].idType, 20, "func", name );

}

else snprintf( SymTbl[id].idType, 20, "var", name );

id++;

}

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

{

fprintf(output,"%s ",&str);

snprintf( SymTbl[id].siNo, 2, "%d", id );

snprintf( SymTbl[id].valueType, 20, "%s", str );

fscanf(input, "%s", &str);

fprintf(output,"%s ",&str);

fscanf(input, "%s", &str);

snprintf( SymTbl[id].name, 20, "%s", str );

fscanf(input, "%s", &str);

fprintf(output,"%s ",SymTbl[id].siNo);

fprintf(output,"%s ",&str);

snprintf( SymTbl[id].scope, 20, "%s", name );

if(strcmp(str,"(")==0)

{

snprintf( name, 20, "%s", SymTbl[id].name );

snprintf( SymTbl[id].idType, 20, "func", name );

}

else snprintf( SymTbl[id].idType, 20, "var", name );

id++;

}

else fprintf(output,"%s ",&str);

if(strcmp(str,"}")==0) snprintf( name, 20, "%s", "global" );

}

fclose(input);

fclose(output);

printf("\n\n\n\nStep2-SYMBOL TABLE: ");

for(int i =1; i<=7; i++) printf("\n%s\t%s\t%s\t%s\t%s",SymTbl[i].siNo,SymTbl[i].name,SymTbl[i].idType,SymTbl[i].valueType,SymTbl[i].scope);

///STEP 4-Syntactic Analysis

input = fopen("Output5.txt", "r");

output = fopen("Output6.txt", "w");

while(fscanf(input, "%s", &str)!=EOF)

{

int flag = 0;

for( int j=1; j<=7; j++)

{

if(strcmp(str,SymTbl[j].name) == 0)

{

fprintf(output,SymTbl[j].siNo);

fprintf(output," ");

flag++;

break;

}

}

if(flag == 0)

{

fprintf(output,str);

fprintf(output," ");

}

}

fclose(input);

fclose(output);

input = fopen("Output6.txt", "r");

output = fopen("Output7.txt", "w");

printf("\n\n\nFinal Step-4: \n\n");

while(fscanf(input, "%s", &str)!=EOF)

{

if(strcmp(str,"id")==0)

{

fprintf(output,"[%s ",str);

printf("[%s ",str);

fscanf(input, "%s", &str);

fprintf(output,"%s]",str);

printf("%s]",str);

}

else{

fprintf(output,"[%s]",str);

printf("[%s]",str);

}

}

fclose(input);

printf("\n\n");

///Step 3

cout<<"Step-3:Symbol table name and classtype"<<endl;

input = fopen("Output8.txt", "r");

//output = fopen("Output1.txt", "w");

string s1 = "", s2 = "";

bool spc = false;

while((c=fgetc(input))!=EOF){

cout<<c;

if(c ==' ') {

spc = true;

continue;

}

if(!spc){

s1+=c;

}

if(spc){

s2+=c;

}

if(c=='\n'){

int n1 = s1.length();

int n2 = s2.length();

// declaring character array

char \*char\_array1 = new char[n1+1];

char \*char\_array2 = new char[n2+1];

//char char\_array1[n1+1];

//char char\_array2[n2+1];

// copying the contents of the

// string to char array

//cout<< "s1 and s2"<< s1<<" "<<s2;

strcpy(char\_array1, s1.c\_str());

strcpy(char\_array2, s2.c\_str());

//char\_array1[s1.size()-1] = '\0';

char\_array2[s2.size()-1] = '\0';

cInsert(char\_array1, char\_array2);

s1 = "", s2 = "";

spc = false;

c = 'o';

continue;

}

//fputc(' ', output);

}

fclose(input);

int choice = showMenu();

while( 1 ){

char \*name = new char[M];

char \*classtype = new char[M];

switch(choice){

case 1:

{

cout << "Insert Selected:\n";

scanf("%s%s", name, classtype);

// Omit the comma character

int pos = strlen(name) - 1;

if( name[ pos ] == ',' ){

name[ pos ] = '\0';

}

printf("%s %s\n", name, classtype);

printf("%d\n", cHash(name) );

cInsert(name, classtype);

}

break;

case 2:

{

cout << "Update Selected:\n";

char\* updatedClasstype = new char[M];

scanf("%s%s%s", name, classtype, updatedClasstype);

// Omit the comma character

int pos = strlen(name) - 1;

if( name[ pos ] == ',' ){

name[ pos ] = '\0';

}

pos = strlen(classtype) - 1;

if( classtype[ pos ] == ',' ){

classtype[ pos ] = '\0';

}

printf("%s %s\n", name, classtype);

printf("%d\n", cHash(name) );

cUpdate(name, classtype, updatedClasstype);

}

break;

case 3:

{

cout << "Search Selected:\n";

scanf("%s%s", name, classtype);

// Omit the comma character

int pos = strlen(name) - 1;

if( name[ pos ] == ',' ){

name[ pos ] = '\0';

}

printf("%s %s\n", name, classtype);

printf("%d\n", cHash(name) );

if( cSearch(name, classtype) ){

printf("FOUND\n");

}else{

printf("NOT FOUND\n");

}

}

break;

case 4:

{

cout << "Delete Selected:\n";

scanf("%s%s", name, classtype);

// Omit the comma character

int pos = strlen(name) - 1;

if( name[ pos ] == ',' ){

name[ pos ] = '\0';

}

printf("%s %s\n", name, classtype);

printf("%d\n", cHash(name) );

cDelete(name, classtype);

}

break;

case 5:

cout << "Show Selected\n";

showSymbolTable();

break;

default:

return 0;

};

choice = showMenu();

}

return 0;

}