**بناء مترجم لتعريف متغير**

**.**

وهيب الحمودي / اعداد

**مترجم لتعريف متغير**

**القواعد المنتظمة**

variabledefine datatype varlist end

datatype **int** |**char**|**double**|**string**|**float**

varlist var

variablelist var,varlist

var letter{letter|digit}

letter a|b|c|….|z

digit 0|1|2|…|9

end ;

**التعابير المنتظمة**

(datatype)(letter){letter|digit};

((**int**|**double**|**char**|**string**|**float**)(a|b|c|…|z){ a|b|c|…|z |0|1|2|…|9}

{(,){ a|b|c|…|z |0|1|2|…|9}};)

**NFA**

 a

 b a

 0 b Int,double,char,float

 0 z z

 9

**خوارزمية دالة التنفيذ main والتي يتم بها استدعاء كافة الدوال لاختبار صحة تعريف المتغيرات عن طريق المتغير المدخل \*var**

Char \*var;var=getNextToken()

While(var[0]!=';')

 yes no

if(kword(ch))

 no yes

Print('error');

Continue;

if(isID (ch))

 no yes

Print('error');

Continue;

Return true;

خوارزمية دالة لإختبار الكلمات المحجوزة (الانواع البيانية)kword(s) مع تمرير المتغير الذي يتم اختبارة واسمة s

char \*kw[4] ={ "int","double","char","flaot"}; int i=0;bool found=true;

if(strcmp(s,kw[i])==0)

دالة strcmp() للمطابقة

 yes no

found=true;

i=4;

Print(notfound)

 yes

Loop i<4

 no

Return found;

دالة bool isID(char\* s) التي تختبر هل المدخل عبارة عن متغير وهل اسمة صحيح ام لا

while(s[0]!=0)

 yes no

if((s[0]>='a's[0]<='z') ||((s[0]>='0's[0]<='9'))

no yes

Return found;

Print(error)

found=true;

s++;

#include<string.h>

#include<iostream>

//defines keywords

char \* keywords[3] =

{

 "double",

 "int",

 "char",

"float",

};

char symbols[1] = { ',',';'};

//check if string is a keyword

bool isKeyword(char\* s)

{

 bool foundKeyword=false;

 for (int i=0; i<31; i++) //iterate through all keywords

 {

if (strcmp(s,keywords[i])==0) //check to see if there is a match

{

foundKeyword=true;

 i=31; //break loop

}

}

return foundKeyword;

}

//check if string is a symbol

bool isSymbol(char s)

{

 bool foundSym=false;

 for (int i=0; i<10; i++) //iterate through all symbols

 {

 if (s == symbols[i]) //check to see if there is a match

 {

 foundSym=true;

 i=10; //break loop

 }

 }

 return foundSym;

}

//check to see if char c is a letter

bool isLetter(char c)

{

 if ((c>='A'&& c<='Z') || (c>='a'&& c<='z'))

 returntrue;

 else

 returnfalse;

}

//check to see if char c is a numeric digit

bool isDigit(char c)

{

 if (c>='0'&& c<='9')

 returntrue;

 else

 returnfalse;

}

//check to see if char c is the end of file character

bool isEOF(char c)

{

 if (c<=0)

 returntrue;

 else

 returnfalse;

}

//check to see if string s is an ID

bool isID(char\* s)

{

//prereq: first character has already been determined to be a letter

 s++; //advance string pointer past first character

 while (s[0]!=0) //continue until end of string

 {

 if (isLetter(s[0])==false&& isDigit(s[0])==false&& s[0]!='\_')

returnfalse; //if character is neither a letter, number or underscore, return false

 s+=1; //move to next character

 }

 returntrue;

}

//======================================================دالة التنفيذ

char \* text;

bool previousTokenWasInteger;

void process(char \* textPtr)

{

 text = textPtr;

 char \* token;

 while (isEOF(text[0])==false)

 {

 if (isSymbol(text[0])) //found symbol - symbol - special case

 {

 printf("%c",text[0]);

 printf(" <SYMBOL> ");

 text++;

 continue;

 }

 //tokenizable lexical classes

 //---------------------------

 token = getNextToken(); //get the next token

 printf("%s", token); //output token

 if (isKeyword(token)) //is token a keyword

 {

 printf(" <KEYWORD> ");

 continue;

 }

 //is token a letter (possibily the start of an ID)

if (isLetter(token[0]))

 {

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

if (isID(token)) //is token an ID

 {

 printf(" <ID> ");

 continue;

 }

 else

 {

 printf(" error on ID ");

 exit (-1);

 }

 }

 }

//gets the next token from the global text pointer

//دالة قراءة الtokenالتالي

char\* getNextToken()

{

 char \* tempPtr = text; //save pointer to text

 char \* newToken;

 int count;

 bool skippedWhitespace = false;

 bool skippedText = false;

 count=0;

 while (isWhitespace(text[0])==false&& isEOF(text[0])==false&& isSymbol(text[0])==false) //keep reading text until whitespace or symbol is reached

 {

 //printf("found %c \n", text[0]);

 skippedText=true;

 text+=1; //advance to next character

 count++;

 }

 if (skippedText==true)

 //printf("Skipped %i characters of text \n", count);

 if (isSymbol(text[0])==true)

 {

 newToken = (char\*)malloc(count+1);

 memcpy(newToken,tempPtr,count);

 newToken[count+1]=0;

 return newToken;

 }

 count=0;

 while (isWhitespace(text[0])==true) //keep reading whitespace until text is reached

 {

 //printf("found %c \n", text[0]);

 skippedWhitespace=true;

 text[0]=0; //zero out whitespace to delimit tokens

 text+=1; //advance to next character

 count++;

 }

 if (skippedWhitespace==true)

 {

 //printf(" <WHITESPACE> "); //output identification of whitespace

 //printf("Skipped %i characters of whitespace \n", count);

 }

 return tempPtr;

}

//check to see if next token is a valid character

if (isDigit(text[0])==true) //octal digit case

{

int i = 0;

while (isDigit(text[0+i]) && i<4) //check how many digits are escaped

 {

 i++;

 }

if (i>=4) //more than three digits, error

 returnfalse;

//3 or less digits

else {

text+=i; //advance text pointer over digits

if (text[0]=='\'') //make sure closing single quote is in place

 {

 text++; //advance over closing single quote

 text[0]=0; //zero out string to terminate token

 text++; //move to begining of next string

 printf("%s", start); //print string

 returntrue;

 }

 else

 returnfalse;

//no closing single quote or more trailing characters! Error

 }

 }

 }

 //Ffall through case. Shouldn't get here. Error

 returnfalse;

}