

201500009075  
Bousri Houssam  
M1-IL groupe 2

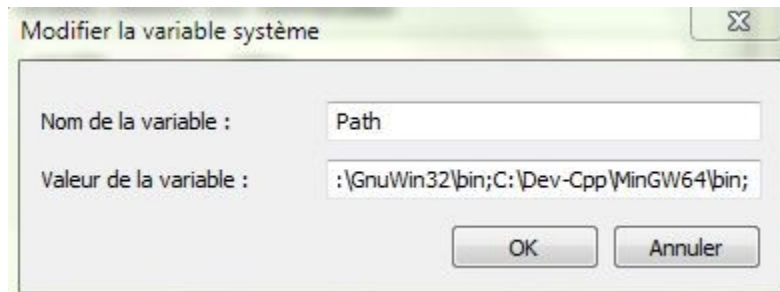
U.S.T.H.B  
Année universitaire 2019/2020  
Module : Compilation

# Rapport Projet Compilation <R>

Avec les outils FLEX et BISON.

## Comment utiliser: SOUS WINDOWS

Les chemin :



dans le dossier /includes a installer/ vous trouviez tout ce qui est nécessaire pour exécuter le code.

la Bibliothèque ▾ Partager avec ▾ Graver Nouveau dossier				
Nom	Modifié le	Type	Taille	
win_flex_bison-latest	12/09/2020 12:57	Dossier de fichiers		
Dev-Cpp 5.11 TDM-GCC 4.9.2 Setup.exe	07/09/2020 12:15	Application	49 252 Ko	
emu8086.exe	11/09/2020 13:37	Application	3 152 Ko	

« Veuillez installer l'émulateur aussi pour vérifier le code machine généré. »

- le code a compiler est écrit dans le fichier in.txt
- le code assembleur est généré dans le fichier code.asem
- les fonctions sont dans le fichier src/\*

Une fois l'installation est terminé, cliquer sur le make.bat pour générer l'exécutable projet.exe.

includes a installer	12/09/2020 12:57	Dossier de fichiers	
src	06/09/2020 15:47	Dossier de fichiers	
code.asm	12/09/2020 12:55	Fichier ASM	1 Ko
in.txt	12/09/2020 12:23	Document texte	1 Ko
make.bat	12/09/2020 12:38	Fichier de comman...	1 Ko
projet.exe	12/09/2020 12:38	Application	181 Ko
projet.l	11/09/2020 18:45	Fichier L	4 Ko
projet.y	11/09/2020 18:43	Fichier Y	7 Ko
README.txt	06/09/2020 18:40	Document texte	1 Ko

## Travail effectué :

« l'ensemble des tokens est dans le fichier projet.l »

« l'ensemble des règles est dans le fichier projet.y »

« pour le signe d'affectation je l'ai remplacé par un = a cause des erreurs de forme de texte unicode vs utf-8 »

### - IDF :

suite alpha-numérique

dans le fichier projet.l on vérifie ici le dépassement de la taille autorisée de l'identifiant :

```
digit [0-9]
entier {digit}+|"{"-"{digit}+"}
idf [A-Z][0-9a-z]*
%%

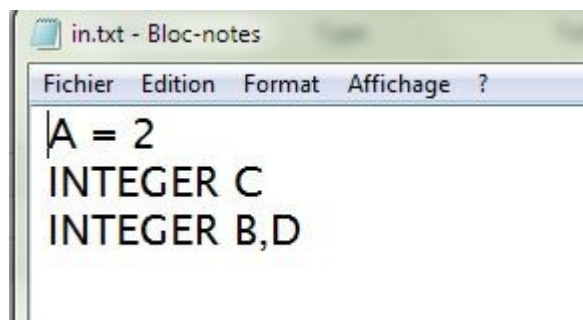
^[ \t]*\n {ligne++;}
"INTEGER" {colonne+=yyleng; yylval.str=strdup(yytext); return(XINT);}
{entier} {colonne+=yyleng; if(StrInt(yytext)<=32767 && StrInt(yytext)>=-32768)yylval.num=StrInt(yytext);
        else {ErrorDepassementTaille(); yylval.num=StrInt("0");} return(ENTIER);}
{idf} {colonne+=yyleng; yylval.str=strdup(yytext);if(yyleng>10){yyerror("IDF Trop Long");} return(IDF);}
```

### - Déclarations :

de type **INTEGER** :

- forme 1 : TYPE Liste\_IDFs
- forme 2 : TYPE IDF = valeur
- forme 3 : IDF = valeur (Erreur de non déclaration)

exemple :



```

C:\Users\Administrateur\Desktop\Compil-R\Compil (R)\projet.exe

Erreur Symantique, Ligne:1, Colone:3, Non Declaration de A

***** Analyse reussite *****

Table des symboles:

*****
*   IDF   *   Type   *   Nature   *
*****
*         *         *         *
*   D     *   Entier *   Variable *
*   B     *   Entier *   Variable *
*   C     *   Entier *   Variable *
*   A     *   Entier *   Variable *
*****

Quads:

```

- **Expressions arithmétique :**  
 (addition, soustraction, multiplication, Division)

exemple :

```

in.txt - Bloc-notes
Fichier  Edition  Format  Affichage
INTEGER C
A =2|
C = 2 * (A / 3)

```

```

C:\Users\Administrateur\Desktop\Compil-R\Compil (R)\projet.exe

***** Analyse reussite *****

Table des symboles:

*****
*   IDF   *   Type   *   Nature   *
*****
*   A     *   Entier *   Variable *
*   C     *   Entier *   Variable *
*****

Quads:

*****
* ID *   OPR *   OP1   *   OP2   *   RES   *
*****
* 00 *   =   *   2     *         *   A     *
* 01 *   /   *   A     *   3     *   t01    *
* 02 *         *   2     *   t01    *   t02    *
* 03 *   =   *   t02    *         *   C     *
*****

```

- **Expressions de comparaison (condition) :**  
(comparaison simple Entier et IDF)

exemple du code du projet.y

```
comparateur: LE {$$ = "BG";}
             GE {$$ = "BL";}
             EQ {$$ = "BNE";}
             NE {$$ = "BE";}
             LT {$$ = "BGE";}
             GT {$$ = "BLE";}
             ;
```

exemple :

```
in.txt - Bloc-notes
Fichier Edition Format Affichage ?
INTEGER C
A =2
IF(A < 1)
{
    C = 4
}
```

```
C:\Users\Administrateur\Desktop\Compil-R\Compil (R)\projet.exe
Erreur Symantique, Ligne:2, Colone:3, Non Declaration de A

***** Analyse reusite *****

Table des symboles:

*****
*   IDF   *   Type   *   Nature   *
*****
*   A   *   Entier   *   Variable  *
*   C   *   Entier   *   Variable  *
*****

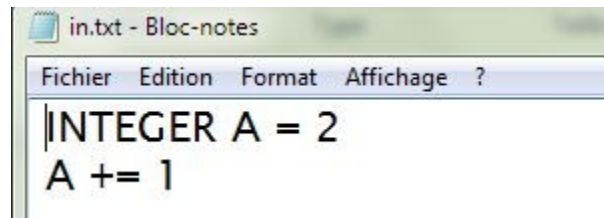
Quads:

*****
* ID *   OPR *   OP1 *   OP2 *   RES *
*****
* 00 *   =   *   2   *       *   A   *
* 01 *  BGE *   03 *   A   *   1   *
* 02 *   =   *   4   *       *   C   *
*****
```

- **Instruction d'affectation avec tous ses forme :**  
« comme déjà montré dans les captures précédentes »

## - Incrémentation de décrémentation par un entier :

exemple :



```
in.txt - Bloc-notes
Fichier  Edition  Format  Affichage  ?
INTEGER A = 2
A += 1
```

Table des symboles:

*****	*****	*****	*****
* IDF *	Type *	Nature *	*
*****	*****	*****	*****
* A *	Entier *	Variable *	*
*****	*****	*****	*****

Quads:

*****	*****	*****	*****	*****	*****
* ID *	OPR *	OP1 *	OP2 *	RES *	*
*****	*****	*****	*****	*****	*****
* 00 *	= *	2 *		A *	*
* 01 *	+	A *	1 *	t01 *	*
* 02 *	= *	t01 *		A *	*
*****	*****	*****	*****	*****	*****

## - Instruction IF :

**remarque : « vous pouvez imbriquer les IF sur N niveaux IF ELSE IF ELSE ....»**

pour le traitement de l'instruction if et if imbriqué j'ai utiliser une pile définit dans le fichier src/pile.c

avec la structure suivante :

```
#include <stdio.h>
#include <stdlib.h>

extern int QC, num_quad;

//definition de element de la pile 1
typedef struct pile{
    int adrCond;
    int nbrif;
    struct pile * suivant;
}pile;

pile * push(pile * tetePile){
    pile * element = malloc(sizeof(pile));
    element->adrCond = QC - 1;
    element->suivant = tetePile;
    tetePile = element;
    return tetePile;
}

pile * pop(pile * tetePile){
    num_quad = tetePile->adrCond;
    tetePile = tetePile->suivant;
    return tetePile;
}

int peek(pile *tetePile){
    if(tetePile == NULL) return 0;
    return tetePile->adrCond;
}
```



exemple de if simple et if else et affectation d'une condition :

```
in.txt - Bloc-notes
Fichier  Edition  Format  Affichage  ?
INTEGER A
IF(1==1)
{
    A = IFELSE((1>2), B, 3+2*7)
}
```

```
C:\Users\Administrateur\Desktop\Compil-R\Compil (R)\projet.exe

Erreur Symantique, Ligne:4, Colone:23, Non Declaration de B

***** Analyse reussite *****

Table des symboles:

*****
*   IDF   *   Type   *   Nature   *
*****
*       B *   Entier *   Variable *
*       A *   Entier *   Variable *
*****

Quads:

*****
* ID *   OPR *   OP1 *   OP2 *   RES *
*****
* 00 *   BNE *   07 *   1 *   1 *
* 01 *   BLE *   04 *   1 *   2 *
* 02 *   =   *   B   *   *   A *
* 03 *   BR  *   07 *   *   *   *
* 04 *   +   *   2   *   7 *   t01 *
* 05 *   +   *   3   *   t01 *   t02 *
* 06 *   =   *   t02 *   *   A *
*****

Quads Optimises:

*****
* ID *   OPR *   OP1 *   OP2 *   RES *
*****
* 00 *   BNE *   07 *   1 *   1 *
* 01 *   BLE *   04 *   1 *   2 *
* 02 *   =   *   B   *   *   A *
* 03 *   BR  *   07 *   *   *   *
* 04 *   +   *   7   *   7 *   t01 *
* 05 *   +   *   3   *   t01 *   t02 *
* 06 *   =   *   t02 *   *   A *
*****

Apuyer sur n'import quel touche pour continuer!
```

## - Instruction WHILE/FOR :

remarque : « vous pouvez aussi imbriquer les WHILE ET FOR loops sur N niveaux »

exemple :

```
in.txt - Bloc-notes
Fichier Edition Format Affichage ?
INTEGER X, A
FOR(X IN 0:1)
{
    A += 1
}
WHILE (A==1)
{
    IF(1==1)
    {
        A = 3
    }
}
```

```
C:\Users\Administrateur\Desktop\Compil-R\Compil (R)\projet.exe

***** Analyse reusite *****

Table des symboles:
*****
*   IDF   *   Type   *   Nature   *
*****
*   A   *   Entier   *   Variable *
*   X   *   Entier   *   Variable *
*****

Quads:
*****
* ID *   OPR   *   OP1   *   OP2   *   RES   *
*****
* 00 *   =   *   0   *       *   X   *
* 01 *   BG   *   07   *   X   *   1   *
* 02 *   +   *   A   *   1   *   t01  *
* 03 *   =   *   t01  *       *   A   *
* 04 *   +   *   X   *   1   *   t02  *
* 05 *   =   *   t02  *       *   X   *
* 06 *   BR   *   01   *       *       *
* 07 *   BNE  *   11   *   A   *   1   *
* 08 *   BNE  *   10   *   1   *   1   *
* 09 *   =   *   3   *       *   A   *
* 10 *   BR   *   07   *       *       *
*****

Quads Optimises:
*****
* ID *   OPR   *   OP1   *   OP2   *   RES   *
*****
* 00 *   =   *   0   *       *   X   *
* 01 *   BG   *   07   *   X   *   1   *
* 02 *   +   *   A   *       *   t01  *
* 03 *   =   *   t01  *       *   A   *
* 04 *   +   *   X   *   1   *   t02  *
* 05 *   =   *   t02  *       *   X   *
* 06 *   BR   *   01   *       *       *
* 07 *   BNE  *   11   *   A   *   1   *
* 08 *   BNE  *   10   *   1   *   1   *
* 09 *   =   *   3   *       *   A   *
* 10 *   BR   *   07   *       *       *
*****

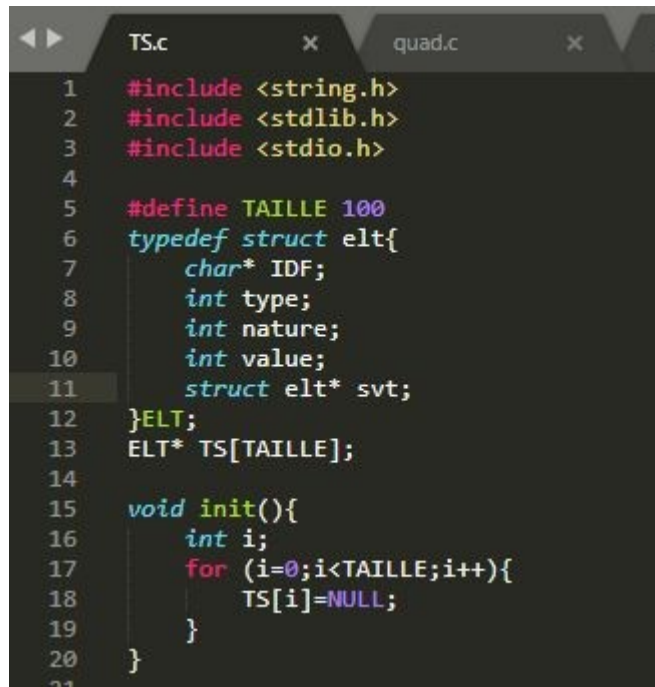
Apuyer sur n'import quel touche pour continuer!
```



## Table des symboles :

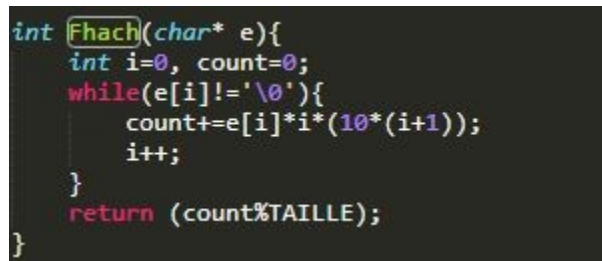
La création de la table des symboles ainsi que sa mise à jour se fait à l'aide de l'ensemble des fonctions dans le fichier src/TS.c

La structure des éléments de la TS est définie comme suit :



```
1  #include <string.h>
2  #include <stdlib.h>
3  #include <stdio.h>
4
5  #define TAILLE 100
6  typedef struct elt{
7      char* IDF;
8      int type;
9      int nature;
10     int value;
11     struct elt* svt;
12 }ELT;
13 ELT* TS[TAILLE];
14
15 void init(){
16     int i;
17     for (i=0;i<TAILLE;i++){
18         TS[i]=NULL;
19     }
20 }
21
```

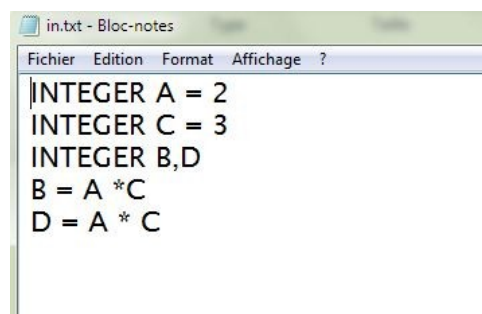
La fonction du hachage :



```
int Fhach(char* e){
    int i=0, count=0;
    while(e[i]!='\0'){
        count+=e[i]*i*(10*(i+1));
        i++;
    }
    return (count%TAILLE);
}
```

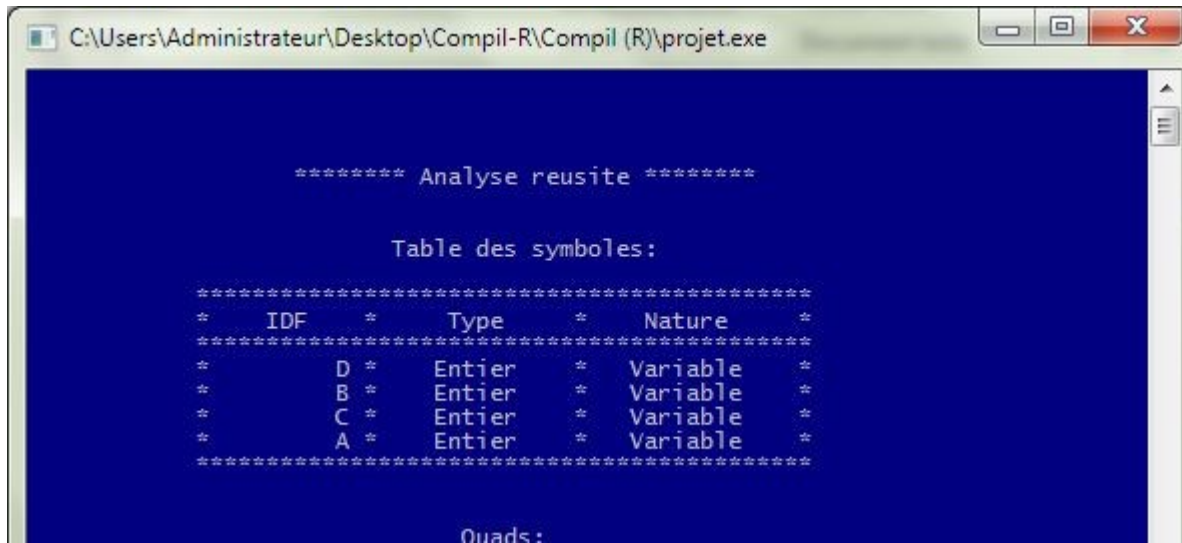
« Le reste des fonction d’affichage et d’insertion dans le fichier src/TS.C »

un exemple en entré :



```
in.txt - Bloc-notes
Fichier Edition Format Affichage ?
INTEGER A = 2
INTEGER C = 3
INTEGER B,D
B = A *C
D = A * C
```

le résultat :



```
***** Analyse reussite *****

Table des symboles:

*****
*      IDF      *      Type      *      Nature      *
*****
*          D *      Entier      *      Variable      *
*          B *      Entier      *      Variable      *
*          C *      Entier      *      Variable      *
*          A *      Entier      *      Variable      *
*****

Quads:
```

## Génération du code intermédiaire :

Notre but ici est de générer le code intermédiaire sous forme de quadruplets en basant sur la table des symboles résultante de la phase précédente.

bien sur on utilisant **une pile** que j'ai défini dans src/pile2 avec la structure suivante :



```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  //definition de element de la pile 2
5  typedef struct pile2{
6      int nbrQuad;
7      struct pile2 * suivant;
8  }pile2;
9
10 pile2 * push2(pile2 * tetePile, int x){
11     pile2 * element = malloc(sizeof(pile2));
12     element->nbrQuad = x;
13     element->suivant = tetePile;
14     tetePile = element;
15     return tetePile;
16 }
17
18 pile2 * pop2(pile2 * tetePile){
19     tetePile = tetePile->suivant;
20     return tetePile;
21 }
22
23 int peek2(pile2 *tetePile){
24     if(tetePile == NULL) return 0;
25     return tetePile->nbrQuad;
26 }
27
28 int empty2(pile2 *tetePile){
29     if(tetePile == NULL) return 1;
30     return 0;
31 }
32
```

Avec les fonctions nécessaires pour ajouter, supprimer un élément ou de vérifier si la pile est vide ou non, aussi bien que **peek2** pour voir le numéro de quad au top de la pile.

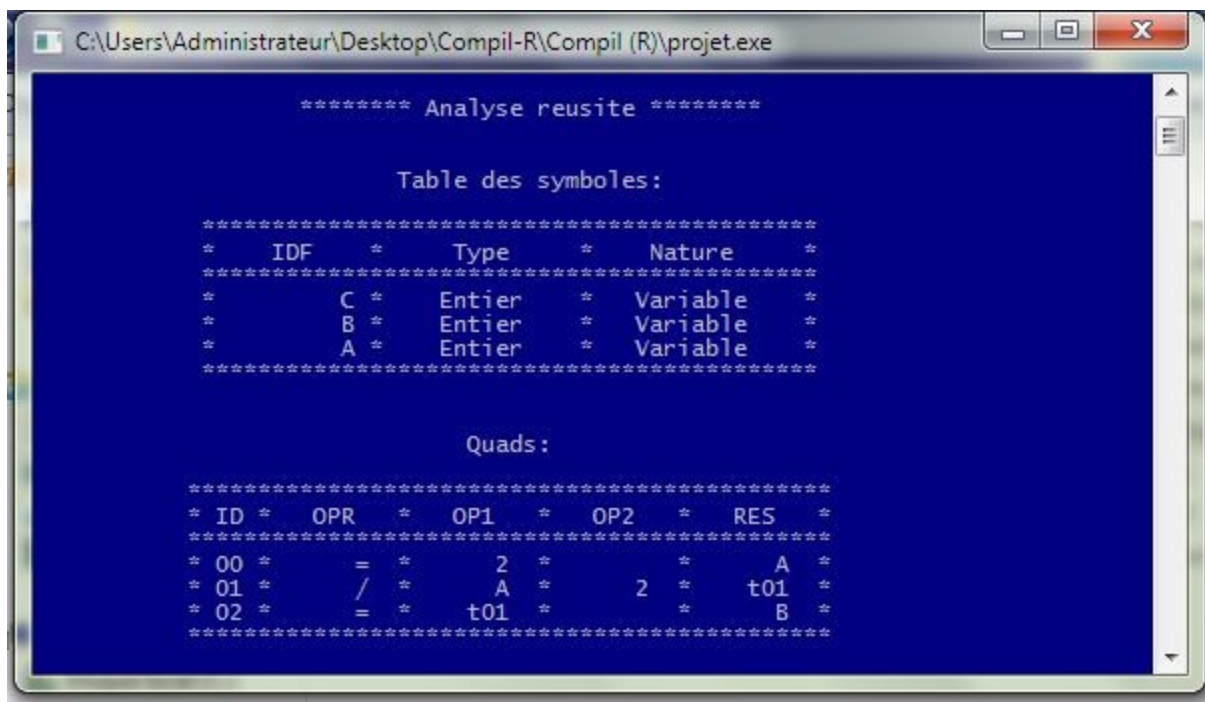
## Exemple d'exécution :

en entré :



```
in.txt - Bloc-notes
Fichier  Edition  Format  Affichage  ?
INTEGER A = 2
INTEGER B,C
B = A / 2
```

résultat :



```
C:\Users\Administrateur\Desktop\Compil-R\Compil (R)\projet.exe

***** Analyse reussite *****

Table des symboles:

*****
*   IDF   *   Type   *   Nature   *
*****
*       C *   Entier  *   Variable *
*       B *   Entier  *   Variable *
*       A *   Entier  *   Variable *
*****

Quads:

*****
* ID *   OPR *   OP1   *   OP2   *   RES   *
*****
* 00 *   =   *     2   *         *   A     *
* 01 *   /   *     A   *     2     *   t01    *
* 02 *   =   *   t01   *         *   B     *
*****
```

Noter bien que la création des quads et l'insertion dans la pile se fait au niveau de la règles au fichier projet.y

Les fonctions d’affichage, modification ou de création des quadruplets ce font par le fichier *src/quad.c* comme indique la figure suivante :

```
C:\Users\Administrateur\Desktop\Compil-R\Compil (R)\src\quad.c - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

quad.c x pile2.c x pile.c x projet.y x
18
19 //création d'un quadruplet
20 void Quad(char* opr, char* op1, char* op2, char* res){
21     q[QC].opr = opr;
22     q[QC].op1 = op1;
23     q[QC].op2 = op2;
24     q[QC].res = res;
25     QC++;
26 }
27
28
29 void afficherQuad(int x){
30     int i;
31     if(x == 1){
32         printf("\n\n\t\t\t\t\tQuads: \n\n");
33     }else{
34         printf("\n\n\t\t\t\t\tQuads Optimises: \n\n");
35     }
36     printf("\t *****\n");
37     printf("\t * ID * OPR * OP1 * OP2 * RES *\n");
38     printf("\t *****\n");
39     for (i=0;i<QC;i++){
40         printf("\t * ");
41         printf("%02d", i);
42         printf(" *");
43         printf(" %5s ", q[i].opr);
44         printf("");
45         printf(" %5s ", q[i].op1);
46         printf("");
47         printf(" %5s ", q[i].op2);
48         printf("");
49         printf(" %5s ", q[i].res);
50         printf("*\n");
51     }
52     printf("\t *****\n");
53 }
54
55 void Maj(int nQc, int jmp){
56     char buff[10];
57     sprintf(buff, "%02d", jmp);
58     q[nQc].op1 = strdup(buff);
59 }
```



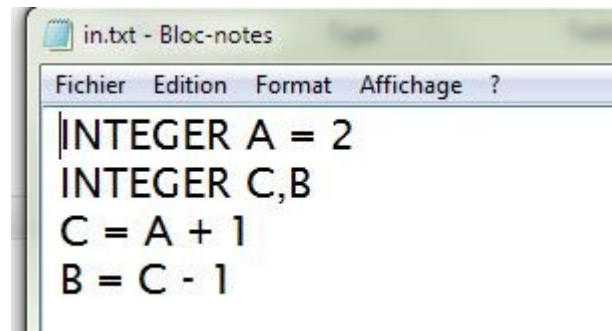
## Optimisation :

Le but de ce point est d'optimiser les quadruplets résultantes de la phase précédente.  
Les fonctions d'optimisation sont dans le fichier src/optim.c

**quelque exemples d'optimisation :**

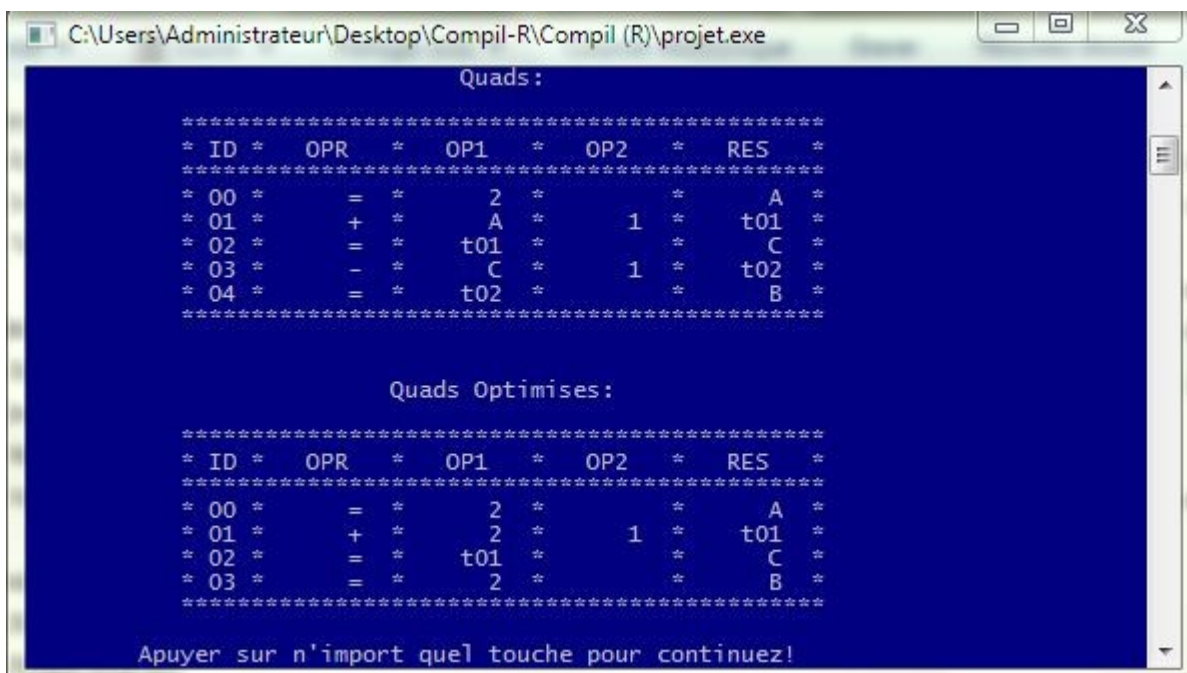
- Simplification Algébrique ( **t01=J+1 ; t02=t01-1 => t02=J**) :

on a comme entrés :



```
in.txt - Bloc-notes
Fichier  Edition  Format  Affichage  ?
INTEGER A = 2
INTEGER C,B
C = A + 1
B = C - 1
```

le résultat des quads et quads optimisées :



```
C:\Users\Administrateur\Desktop\Compil-R\Compil (R)\projet.exe

Quads :
*****
* ID *   OPR *   OP1 *   OP2 *   RES  *
*****
* 00 *   =  *    2  *       *   A   *
* 01 *   +  *    A  *    1  *  t01  *
* 02 *   =  *   t01 *       *   C   *
* 03 *   -  *    C  *    1  *  t02  *
* 04 *   =  *   t02 *       *   B   *
*****

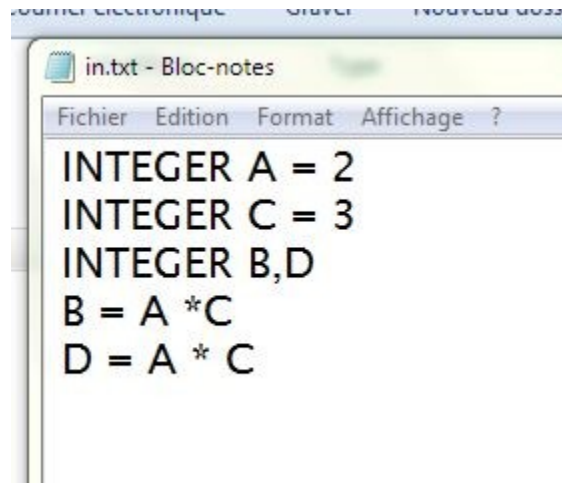
Quads Optimises:
*****
* ID *   OPR *   OP1 *   OP2 *   RES  *
*****
* 00 *   =  *    2  *       *   A   *
* 01 *   +  *    2  *    1  *  t01  *
* 02 *   =  *   t01 *       *   C   *
* 03 *   =  *    2  *       *   B   *
*****

Appuyer sur n'importe quel touche pour continuer!
```

on remarque bien qu'on a remplacé le **t02** par la valeur direct déjà calculé.  
Aussi bien qu'on a remplacé le **C** par son **temporaire** car il est utilisé une seul fois après sa déclaration.

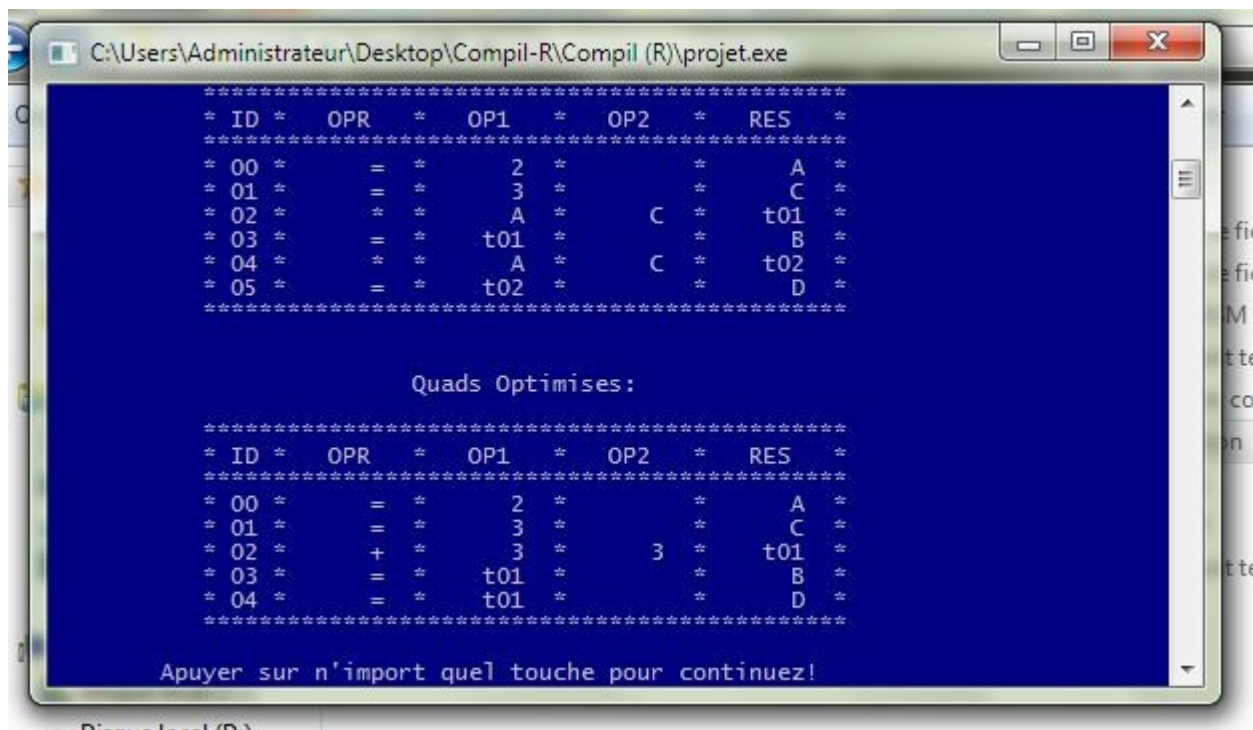
- **expression redondante :**

on a comme entrés :



```
INTEGER A = 2
INTEGER C = 3
INTEGER B,D
B = A * C
D = A * C
```

le résultat :



```
C:\Users\Administrateur\Desktop\Compil-R\Compil (R)\projet.exe

*****
* ID *   OPR *   OP1 *   OP2 *   RES *
*****
* 00 *   = *   2 *   *   A *
* 01 *   = *   3 *   *   C *
* 02 *   * *   A *   C *   t01 *
* 03 *   = *   t01 *   *   B *
* 04 *   * *   A *   C *   t02 *
* 05 *   = *   t02 *   *   D *
*****

          Quads Optimises:

*****
* ID *   OPR *   OP1 *   OP2 *   RES *
*****
* 00 *   = *   2 *   *   A *
* 01 *   = *   3 *   *   C *
* 02 *   + *   3 *   3 *   t01 *
* 03 *   = *   t01 *   *   B *
* 04 *   = *   t01 *   *   D *
*****

Apuyer sur n'import quel touche pour continuer!
```

on remarque que le t02 a été remplacé par t01 ensuite par la valeur direct de la variable.

**Le reste des optimisations prises en considération sont :**

- Simplification Algébrique (  $t01 = J + 1$  ;  $t02 = t01 - 1 \Rightarrow t02 = J$  )
- Remplacer la variable utilisée une fois seulement après sa déclaration par son temporaire.
- élimination des quadruplets non utilisés.
- Réduction de multiplication et division et suite de **-(exp)** quand le quadruplet est constant.
- propagation de copie  **$(t01 = (t02, cst, IDF) \Rightarrow$  on remplace les apparences de  $t01$  par  $(t02, cst, IDF)$ .**
- Élimination d'expression redondante  **$(t01 = A * B, t02 = A * B \Rightarrow t01 = A * B, t02 = t01)$**

« Le code est bien documenté pour chaque une des fonctions d'optimisation »

## Génération du code machine :

Le but de cette étape est de générer le code assembleur correspondant au code introduit dans le fichier d'entrée in.txt.

« Notez svp que : »

1- Le code machine est généré selon les quadruplets optimisées résultante de l'étape précédente.

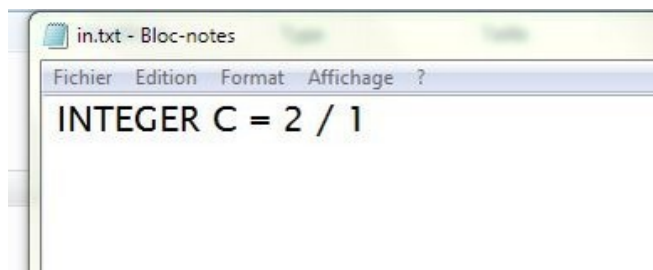
2- vous pouvez tester le code machine en installant l'émulateur 8086 inclus dans le dossier ressources.

La génération du code machine se fait à l'aide de d'une fonction `codeAssem()` dans le fichier src/assem.c et va être exécuté au niveau de main dans le projet.y.

On utilisant les quadruplet et la tables des symboles généré on construit le fichier code.asm

### Rappelons la structure du code résultat :

on a en entrée :



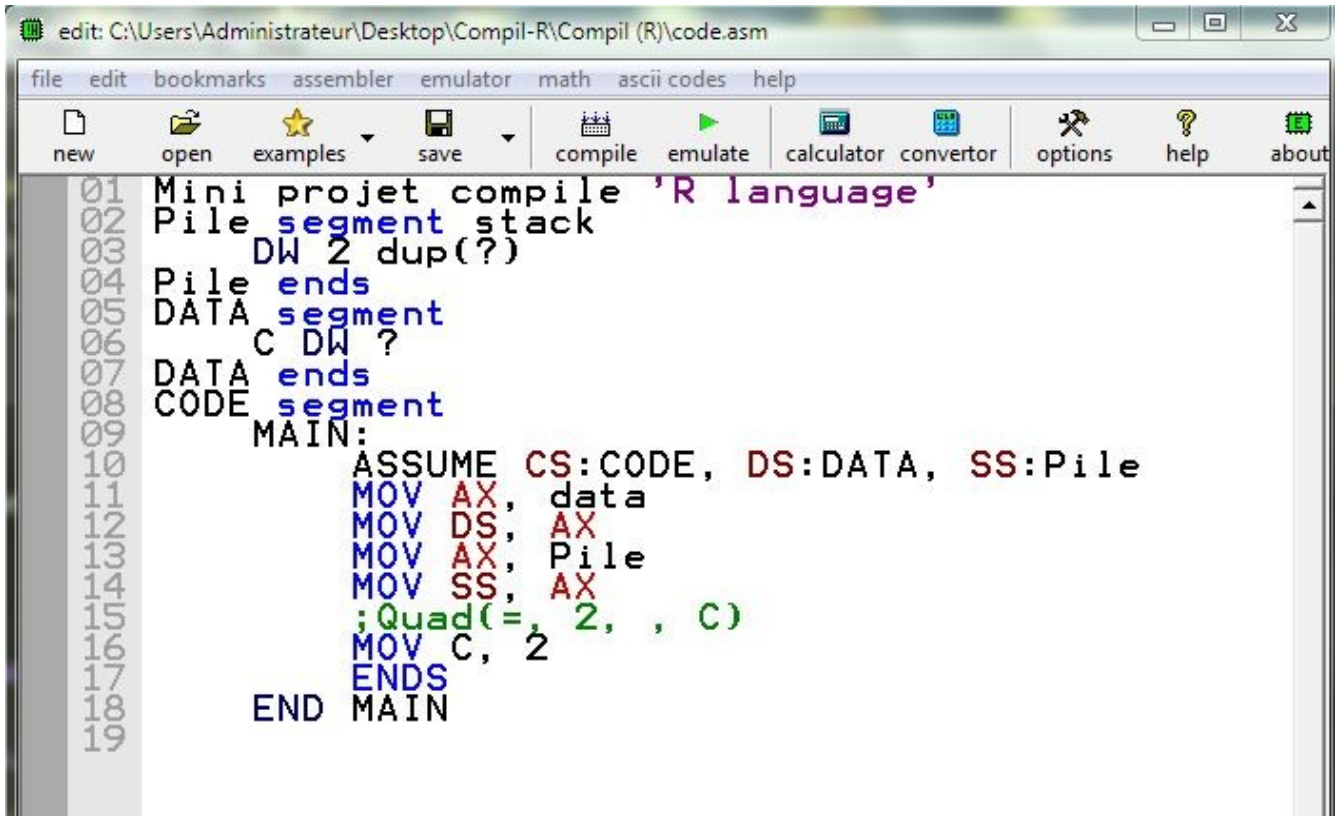
le code pour la génération du segment de la pile :

```
FILE *f = fopen("code.asm", "w");
fputs("Mini projet compile 'R language'\n", f);
fputs("Pile segment stack\n", f);
fprintf(f, "\tDW %d dup(?)\n", nbr_temps+3);
fputs("Pile ends\n", f);
```

le code pour la génération du segment data:

```
fputs("DATA segment\n", f);
int i=0; ELT* p;
for (i=0; i<TAILLE; i++){
    if (TS[i]!=NULL){
        p=TS[i];
        while(p!=NULL){
            fprintf(f, "\t%s DW ?\n", p->IDF);
            p=p->svt;
        }
    }
}
fputs("DATA ends\n", f);
```

le résultat final en sortie on aurait :



The screenshot shows a window titled "edit: C:\Users\Administrateur\Desktop\Compil-R\Compil (R)\code.asm". The window has a menu bar with "file", "edit", "bookmarks", "assembler", "emulator", "math", "ascii codes", and "help". Below the menu bar is a toolbar with icons for "new", "open", "examples", "save", "compile", "emulate", "calculator", "convertor", "options", "help", and "about". The main text area contains the following assembly code:

```
01 Mini projet compile 'R language'
02 Pile segment stack
03     DW 2 dup(?)
04 Pile ends
05 DATA segment
06     C DW ?
07 DATA ends
08 CODE segment
09     MAIN:
10         ASSUME CS:CODE, DS:DATA, SS:Pile
11         MOV AX, data
12         MOV DS, AX
13         MOV AX, Pile
14         MOV SS, AX
15         ;Quad(=, 2, , C)
16         MOV C, 2
17     ENDS
18 END MAIN
19
```

« le reste du code de la fonction codeAssem() est dans le fichier src/assem.c »



## Traitement des erreurs :

Pour le traitement des erreurs, un ensemble des erreurs a été traité (lexical ou sémantique).

Sous la forme suivante :

« **Type de l'erreur**, **ligne n**, **colonne m**, **description de l'entité correspondante**. »

Voici quelque exemple :

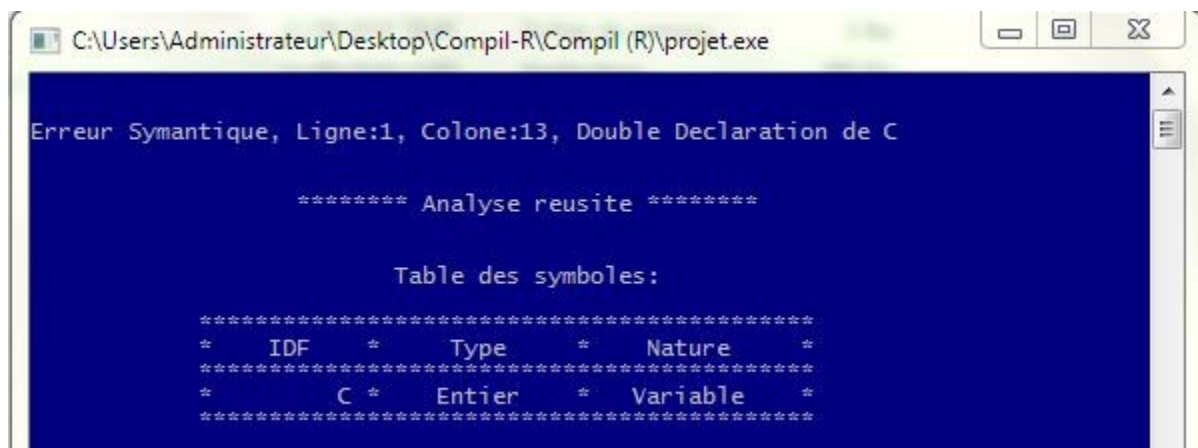
### - Prenant l'exemple de la double déclaration :

en entré dans le fichier in.txt:



```
in.txt - Bloc-notes
Fichier  Edition  Format  Affichage  ?
INTEGER C, C
C = 2 * 7
```

en exécutant notre projet.exe :



```
C:\Users\Administrateur\Desktop\Compil-R\Compil (R)\projet.exe

Erreur Symantique, Ligne:1, Colone:13, Double Declaration de C

***** Analyse reusite *****

Table des symboles:

*****
*   IDF   *   Type   *   Nature   *
*****
*       C *   Entier *   Variable *
*****
```

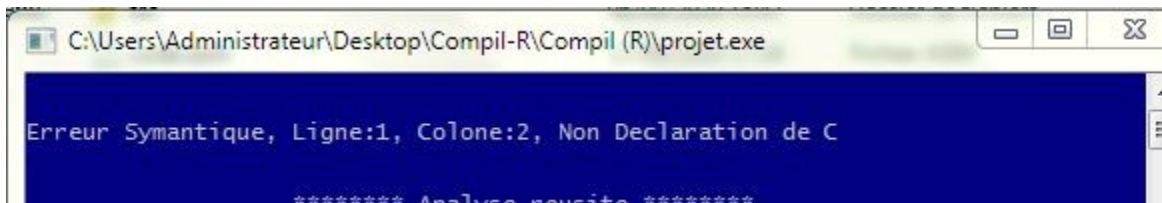
on remarque l'indication d'une erreur sémantique de type double déclaration avec toutes les informations demandés (ligne et colonne et description.)

**- Autre exemple «non déclaration» :**

en entré dans le fichier in.txt:



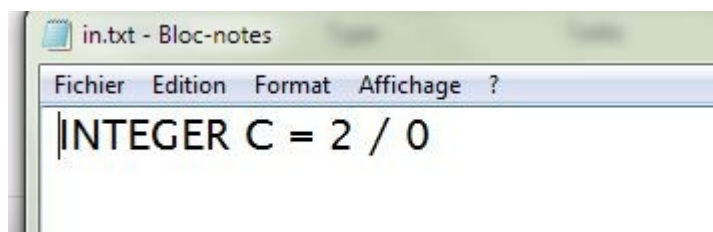
en exécutant projet.exe :



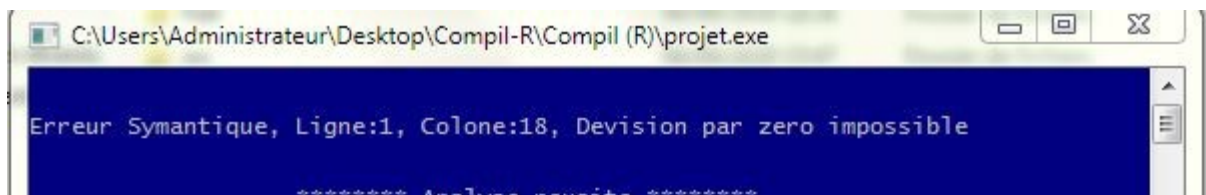
on remarque l'erreur sémantique affichée.

**- Autre exemple «Division par zéro» :**

en entré dans le fichier in.txt:

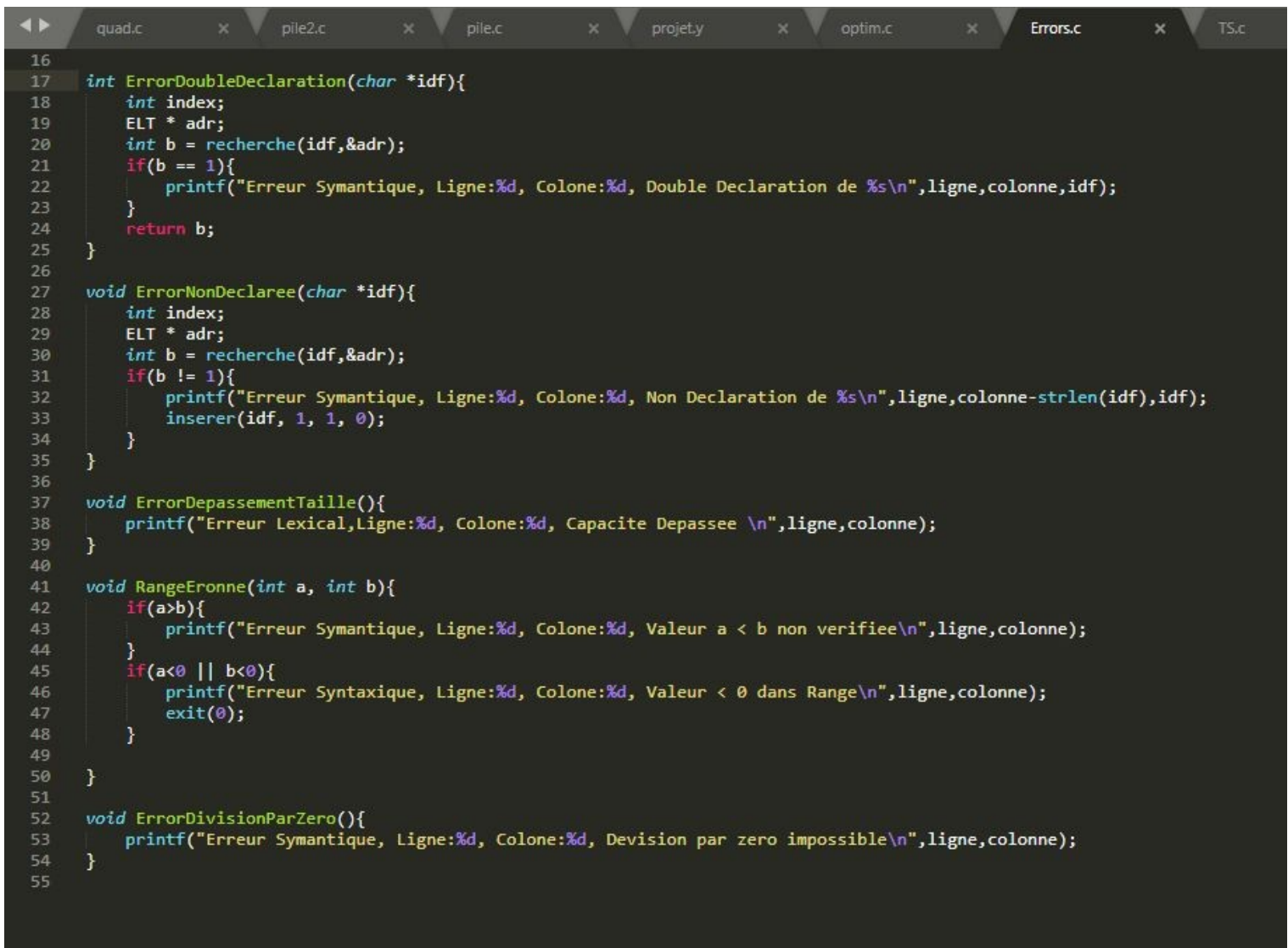


en exécutant projet.exe :



on remarque l'erreur affichée.

L'ensemble des erreurs traitées sont comme présente la figure suivante :

A screenshot of a C code editor with multiple tabs at the top: quad.c, pile2.c, pile.c, projet.y, optim.c, Errors.c, and TS.c. The 'Errors.c' tab is active, displaying the following code:

```
16
17 int ErrorDoubleDeclaration(char *idf){
18     int index;
19     ELT * adr;
20     int b = recherche(idf,&adr);
21     if(b == 1){
22         printf("Erreur Symantique, Ligne:%d, Colone:%d, Double Declaration de %s\n",ligne,colonne,idf);
23     }
24     return b;
25 }
26
27 void ErrorNonDeclaree(char *idf){
28     int index;
29     ELT * adr;
30     int b = recherche(idf,&adr);
31     if(b != 1){
32         printf("Erreur Symantique, Ligne:%d, Colone:%d, Non Declaration de %s\n",ligne,colonne-strlen(idf),idf);
33         inserer(idf, 1, 1, 0);
34     }
35 }
36
37 void ErrorDepassementTaille(){
38     printf("Erreur Lexical,Ligne:%d, Colone:%d, Capacite Depassee \n",ligne,colonne);
39 }
40
41 void RangeEronne(int a, int b){
42     if(a>b){
43         printf("Erreur Symantique, Ligne:%d, Colone:%d, Valeur a < b non verifiee\n",ligne,colonne);
44     }
45     if(a<0 || b<0){
46         printf("Erreur Syntaxique, Ligne:%d, Colone:%d, Valeur < 0 dans Range\n",ligne,colonne);
47         exit(0);
48     }
49 }
50
51
52 void ErrorDivisionParZero(){
53     printf("Erreur Symantique, Ligne:%d, Colone:%d, Devision par zero impossible\n",ligne,colonne);
54 }
55
```

L'utilisation des ces fonctions est bien sur dans la définition des règles dans projet.y !