Langage C

Wieslaw Zielonka
zielonka@irif.fr

Fonctions et vecteurs

```
void echange(unsigned int n, int t[],
             unsigned int i, int j){
   if( i >= n || j >= n )
       return;
   int c;
   c = t[i];
   t[i] = t[j];
   t[j] = c;
}
```

```
int main(void){
   int tab[] = {4, 6, 8, -11, -8, 2};
   echange(6, t, 1, 3);
   printf("t[1]=%d t[3]=%d\n", t[1], t[3]);
   return 0;
}
Avant l'appel à echange() :  tab[1]=6, tab[3]=-11
Après l'appel à echange() :  tab[1]=-11, tab[3]=-6
```

préprocesseur -> compilation -> linkage

Quand on lance gcc il y a en fait trois programmes différents qui s'exécutent à tour de rôle :

1. Le **préprocesseurs** transforme le texte du programme suivant les directives commençant par #. Le résultat c'est un fichier texte.

2. Le **compilateur** traduit le texte du programme vers un code binaire qui n'est pas encore exécutable. Il manque, par exemple, les fonctions de la bibliothèque standard, comme printf().

3. Le linker rassemble les différentes parties du code binaire, ajoute les références vers les fonctions des bibliothèques. Le résultat est un code binaire exécutable sur la machine.

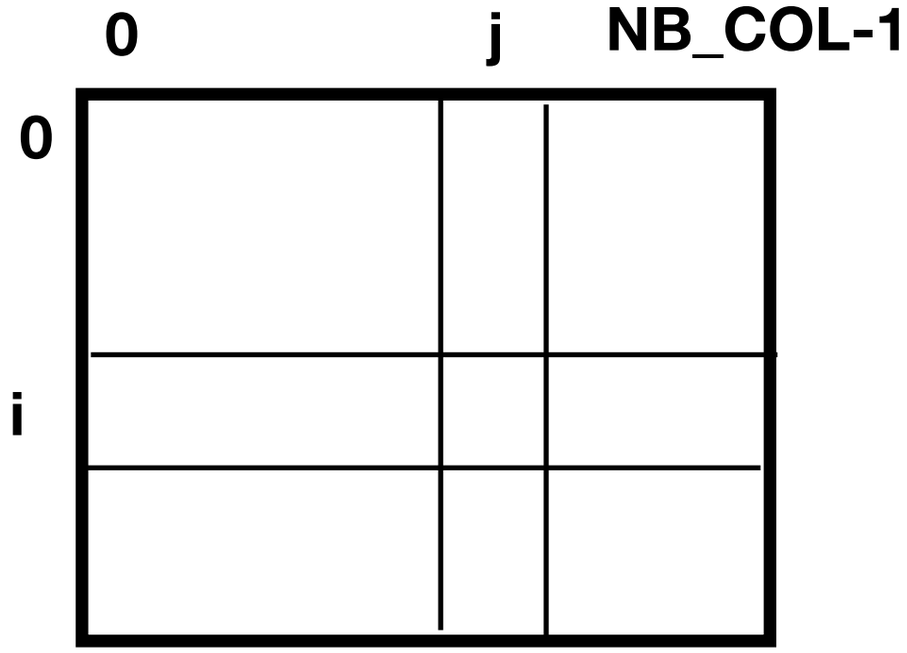
code source

```c
#include <stdio.h>
#define NB_LIN 10
#define NB_COL 20

int fun(int,int);0

int main(void){
   double tab[ NB_LIN * NB_COL ];
   int i, j;
   for(int i=0; i < NB_LIN ; i++){
      for( int j = 0; j < NB_COL; j++){
          tab[i * NB_COL + j] = fun(i,j);
      }
   }

.....
```



le fichier texte obtenu à la sorite du préprocesseur

```
#include <stdio.h> remplacé par  le contenu du fichier
stdio.h (plusieurs centaines de lignes).

int main(void){
   double tab[10*20];
   int i, j;
   for(int i=0; i < 10*20; i++){
       tab[i]=0;
   }
   .....
   tab[i * 20 + j] = 20*20 - 10;
```

Le préprocesseur remplace toutes les occurrences des macro-constantes NB_LIN par 10 et les occurrences de NB_COL par 20.

Formatage du code

une déclaration par ligne

~~`int x; int y =1;`~~

```
int  x;
int y = 1;
```

une instruction élémentaire par ligne

~~`i=0; j=10;`~~

```
i=0;
j=10;
```

accolade ouvrante termine la ligne

~~`if(x<y){ t=x; x=y; y=t;}`~~

```
if(x<y){
  t =x;
  x=y;
  y=t;
}
```

```
    if(x<y)
    {
     t =x;
      x=y;
      y=t;
    }
```

Formatage du code

Une seule accolade fermante sur une ligne :

```
}}
```

```
    }
}
```

l'étiquette seule sur une ligne

```
erreur: free(p);
        return 0;
```

```
erreur :
    free( p);
    return 0;
```

`if` et `else` sur la même colonne:

```
if()

else
    if
```

```
if()

else if()
```

Formatage du code

boucle vide, virgule sur la ligne suivante (et commentaire)

```
while( tab[i++] > 0 )
        ; /* boucle vide */

for( i=0; i < NB_ELEM && tab[i]>0 ; i++)
            ; /* boucle vide */
```

Pour obtenir les indentations correctes sous emacs :

- choisir la partie à formater :
    - ctrl-espace pour marquer le début et déplacer le cursor à la fin du code sélectionné
- taper TAB pour formater

les opérateurs à effet de bord

```
int x = 7, y ;

y = ++ x;

printf("x=%d y=%d\n") ; /* affiche x=8 y=8 */
```

L'effet de bord de ++ : incrémentation de x

La valeur de l'expression ++x : 8 (la valeur de x **après** incrémentation).

```
x = 7;

y = x ++;

printf("x=%d y=%d\n") ; /* affiche x=8 y=7 */
```

l'effet de bord de ++ : incrémentation de x

la valeur de l'expression x++ : 7 (la valeur de x **avant** incrémentation)

expression ternaire

```
(condition) ? val1 : val 2
```

Si condition est vraie la valeur de l'expression est val1 sinon val2

```
int a, b, c;

c = (a < b)? a-1 : b+2;
```

c reçoit la valeur de a-1 si a < b et la valeur de b+2 sinon.

```
printf("%d\n", (a < b)? a : b );
```

affiche plus petit de deux nombres a, b

vecteurs de longueur variable (variable length arrays VLA - optionnel en C11)

C99 a introduit variable length arrays mais C11 a rétrogradé cette possibilité en option. Donc un compilateur conforme à C11 n'est pas obligé d'implementer VLA.

gcc et beaucoup d'autres compilateurs acceptent VLA.

```
int somme(int nb, int t[nb]){
  int s = 0;
  for(int i=0; i<nb ; i++){
        s += tab[i];
   }
   return s;
}
```

Le paramètre qui indiquent la dimension du vecteur doit précéder le vecteur sur la liste de paramètres.

vecteur de longueur variable (VLA - optionnel en C11)

```
double fun(unsigned int n){

    unsigned int k;

    k = 100;

    double tab[k * n * 2  + 10];

}
```

Le nombre d'éléments de tab dépend de k et de n.

Structures

Vecteurs : pour agréger plusieurs éléments du même type

structures : pour agréger des éléments de types différents.

```
struct personne{
   char sex;
   unsigned int annee; /* annee de naissance */
   char  nom[20];
   char  prenom[20];
};
```

Notez ; (point-virgule) après chaque champs, y compris le dernier.

```
struct personne beta; /* déclarer la variable beta de type struct
                         personne */

/* déclarer et initialiser une variable de type struct */
struct personne robert = { .sex = 'm',  .nom ="Tituti",
                           .annee = 1956, .prenom = "Vlad"};
```

Structures

Une fois une variable de type structure déclarée :

struct personne delta ;

impossible de mettre des valeurs dans plusieurs champs d'un seul coup :

~~delta = { .sex = 'm', .nom ="Tituti", .annee = 1956,~~
~~.prenom = "Vlad"};~~

il faut mettre les valeurs champs par champs :

delta.sex = 'm';

delta.annee = 1995;

delta.nom[0]='T'; delta.nom[1]='i'; delta.nom[2]='t'; delta.nom[3]='u';
delta.nom[4]='t'; delta.nom[5]='i'; delta.nom[6]='\0';

Structures

```
struct point{
  int x;
  int y;
};
struct point p1,p2,p3;
```

Définition de la structure struct point
et la déclaration de trois variables de type struct point.

```
p1.x = 2;
p1.y = -5;
p2.x = - p1.x - 1;

p3 = p1;   /* l'affection est possible entre deux variables
            * de type struct de même type  */
```

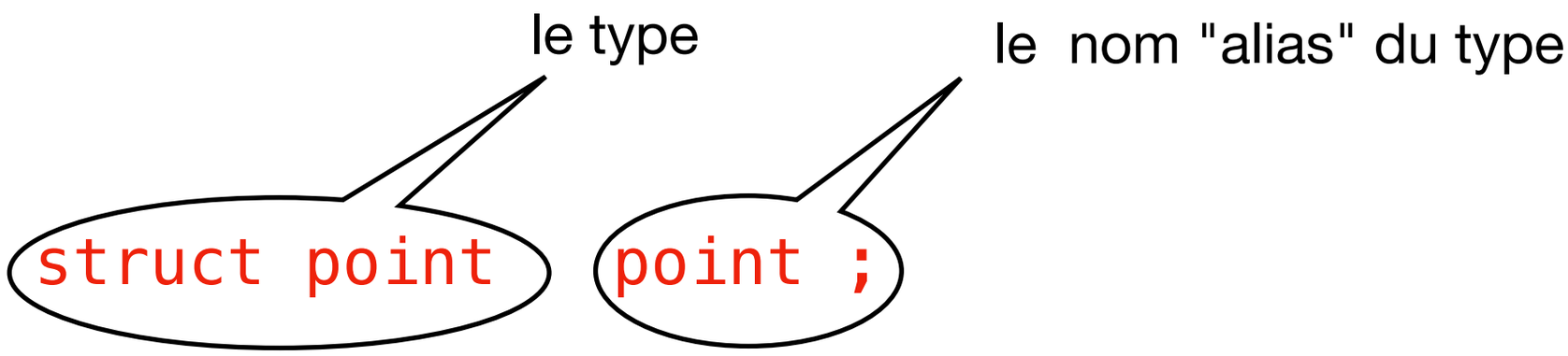
Impossible de comparer les valeur de deux variables de type structure à l'aide de ==

~~`if( p1 == p3 )`~~

```
if ( p1x  ==  p3.x && p1.y  ==  p3.y )
```

Marre de taper struct? Utilisez typedef

```
struct point{
   int x;
   int y;
} ;
```




```
typedef struct point  point ;

point p3 = {.x = 3, .y = -7};
```

Possible de définir une structure et le nom alias en même temps:

```
typedef struct point{
int x;
int y;
} point;
```

Marre de taper struct? Utilisez typedef

Et même définir une structure sans nom et le type en même temps :

```
typedef struct{
   int x;
   int y;
} point;
```

```
point p4;
point p5 = {.x = 3, .y = -7};

struct point p5;   /* il n'y a pas de type
                    *   struct point */
```

Vecteur de structures

```
typedef struct point{
   int x;
   int y;
} point ;

point tab_points[10]; /* vecteur de
                         structures */

tab_points[0].x = 2;

tab_points[0].y = tab_points[0].x - 20;
```

Structures dans les structures

```
typedef struct point{
   int x;
   int y;
} point ;

typedef struct rectangle{
     point pa;
     point pb;
} rectangle;

rectangle r = { .pa = { .x = 1, .y=1 },
                .pb = { .x = 2, .y = 3} };

rectangle d;

d.pa.x = 5; d.pa.y = 5;
d.pb.x = 8; d.pb.y = 23;
```

Structures dans les structures

Deux définition de types avec les structures anonymes :

```
typedef struct{
   int x;
   int y;
} point ;
```

```
typedef struct{
    point pa;
    point pos;
} rectangle;

/* declarer les variables */
rectangle r;

point p;
```

vecteurs dans les structures

```
typedef struct{
  int x;
  int y;
} point ;
#define NB_MAX 20

typedef struct{
   unsigned int nb_sommets;
   point sommets[NB_MAX];
} polygone;
```

`polygone` utilisé pour mémoriser les sommets d'un polygone. nb_sommets : le nombre de sommets au maximum 20.

```
polygone triangle, tr;

triangle.nb_sommets = 3;

triangle.sommets[0].x = -1; triangle.sommets[0].y = 0;
triangle.sommets[1].x = 1; triangle.sommets[1].y = 0;
triangle.sommets[2].x = 1; triangle.sommets[2].y = 1;

tr = triangle;

/* OK, on peut faire une affectation entre deux variables de type structure
qui contiennent des vecteurs même si on ne peut pas faire affectation entre
deux vecteurs ! */
```

Structures comme paramètres de fonctions

```
typedef struct{
  int x;
  int y;
} point;

void mirror(point p){
  p.x = -p.x;
  p.y = -p.y;
}

int main(void){
  point p = { .x = 9, .y = -11 };
  printf("p.x=%d p.y=%d\n", p.x, p.y);
  mirror(p);
  printf("p.x=%d p.y=%d\n", p.x, p.y);
  return 0;
}
```

Quelles valeurs affichées par chaque printf()?

Structures comme paramètres de fonctions

```
typedef struct{
  int x;
  int y;
} point;
void mirror(point p){
  p.x = -p.x;
  p.y = -p.y;
}
int main(void){
  point q = { .x = 9, .y = -11 };
  printf("q.x=%d q.y=%d\n", q.x, q.y);
  mirror(q);
  printf("q.x=%d q.y=%d\n", q.x, q.y);
  return 0;
}
```

Quelles valeurs affichées par chaque printf()?

q.x=9 q.y=-11
q.x=9 q.y=-11

Le paramètre `p` de la fonction `mirror()` est initialisé avec les valeurs de champs qui viennent de la variable `q`. `q` dans `main()` ne sera pas modifié par la fonction `mirror()`.

Structure comme valeur de retour de fonction

```
typedef struct{
  int x;
  int y;
} point;

/* une fonction peut retourner une structure */
point inverser(point p){
  point q = { .x = p.y, .y = p.x };
  return q;
}
```

```
int main(void){
  point p = { .x = 9, .y = -11 };
  printf("p.x = %d p.y = %d\n", p.x, p.y);
  point a = inverser(p);
  printf("a.x = %d a.y = %d\n", a.x, a.y);
  return 0;
}
```

```
p.x = 9 p.y = -11
a.x = -11 a.y = 9
```

Structures et fonctions

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

typedef struct{
  double x;
  double y;
} point;

#define NB_MAX 20
typedef struct{
  unsigned int nb_sommets;        /* le nombre de sommets */
  point sommets[NB_MAX];          /* le tableau de sommets de polygone,
}polygone ;                        *  NB_MAX le nombre maximal de sommets */

double distance(point a, point b){
  return sqrt( (a.x-b.x)*(a.x-b.x)+ (a.y-b.y)*(a.y-b.y) );
}

double perimetre(polygone poly){
  double s = 0;
  for(int i = 0; i < poly.nb_sommets-1; i++){
    s += distance(poly.sommets[i], poly.sommets[i+1]);
  }
  s += distance(poly.sommets[poly.nb_sommets-1], poly.sommets[0]);
  return s;
}
```

Structures et fonctions (cont)

```
int main(void){

  polygone triangle = { .nb_sommets = 3,
        { { .x = -1.0, .y = 0.0 },
          { .x = 1.0, .y = 0.0 },
          { .x = 0.0, .y = 5.0 } } };

  double p = perimetre(triangle);
  printf("perimetre = %f\n",p);
  return 0;
}
```

Structures et fonctions

- une structure peut être utilisée comme un paramètre d'une fonction,

- comme pour d'autres paramètres, le paramètre de type structure est un variable locale à la fonction initialisée à l'appel de fonction

- la fonction peut retourner une structure

- en particulier, une fonction peut retourner une structure qui contient un tableau (même quand c'est le seul champ de la structure) alors qu'en C les fonction ne peuvent pas retourner des tableaux

Fonctions mathématiques

Les fonctions mathématiques, comme `sqrt()`, sont déclarées dans le fichier en-tête

`math.h`

Mais

```
#include <math.h>
```

n'est pas suffisant.

Il faut ajouter dans Makefile la ligne:

```
LDLIBS=-lm
```

LDLIBS - les bibliothèques utilisées pour trouver des fonctions

Le fonction mathématiques se trouvent dans la bibliothèque libm.

Toutes les autres fonctions du C standard se trouvent dans la bibliothèque glibc qui est automatiquement recherché par le linker.

les fonctions mathématiques

```
#include <math.h>
```

quelques exemples de fonctions mathématiques :

`sin()`, `cos()`, `asin()`, `sqrt()`, `log()`, `exp()` etc

la page man de math :
man math.h sous linux
man math sous MacOS

enumeration

enumeration définit un type composé de constantes numériques:

```
enum color{ BLUE , RED , GREEN };

/* comme pour les structures on peut définir un type */
typedef enum color color;

color feu; /* déclarer une variable */

enum  color autres_feu;  /* et encore une */

feu = RED ;
if( feu == RED ){ /* condition satisfaite ici */

}
```

Par défaut les valeurs de constantes dans la liste sont 0,1,2,3 etc. Il est possible de spécifier explicitement les valeurs des constantes:

```
typedef enum color{ BLUE =1 , RED = 2, GREEN = 4 } color;
```

enumeration

enumeration définit un type composé de constantes numériques:

```
enum color{ BLUE , RED , GREEN };

enum color feu; /* déclarer une variable de type enum color*/

enum  color autres_feu;  /* et encore une */

feu = RED ;
if( feu == GREEN ){
}
```

Par défaut les valeurs de constantes dans la liste sont 0,1,2,3 etc. donc BLUE == 0, RED == 1

GREEN == 2. La variable `feu` est juste une variable qui peut prendre une de trois valeurs entières.

Il est possible de spécifier explicitement les valeurs des constantes:

```
enum color{ BLUE = 1 , RED = 2, GREEN = 4 };
```

enumeration

enumeration définit un type composé de constantes numériques:

```
enum color{ BLUE , RED , GREEN };

/* comme pour les structures on peut définir un type */


typedef enum color color;

color feu; /* déclarer une variable de type color */

feu = RED ;
if( feu == RED ){ }
```

Compilation avec make

La commande make facilite la compilation de programmes en C (et pas seulement en C). Elle lit les directives de compilation dans un fichier Makefile.

Supposons que dans le répertoire courant nous avons trois fichiers sources :

prog1.c prog2.c prog3.c

avec trois programmes (chaque fichier contient sa propre fonction main()).

Préparer avec emacs le fichier Makefile avec le contenu suivant:

```
CC=gcc

CFLAGS=-Wall -g -std=c11

LDLIBS= -lm

ALL = prog1 prog2 prog3

all : $(ALL)

prog1 : prog1.c

prog2 : prog2.c

prog3 : prog3.c

cleanall:

	rm -rf *~ $(ALL)
```

variables de make :
CC le nom du compilateur
CFLAGS les options de compilateur
LDLIBS les bibliothèques

ALL ma variable avec les noms d'exécutables

prog1 : prog1.c ligne de dépendance
indique que pour obtenir
prog1 il faut compiler le fichier
prog1.c

la ligne avec la commande rm commence par le caractère TAB (et non pas par des espaces)

Compilation avec make

Si vous taper la commande `make` sur le terminal alors alors les trois programmes seront compilés avec les bonnes options. Mais avant de lancer la compilation make vérifie pour chaque `prog1, prog2, prog3` si la compilation est nécessaire.

make considère que la compilation est inutile si la date de la dernière modification de fichier source prog.c est antérieure à la date de la création/modification de fichier exécutable correspondant `prog` (`c-à-d` la dernière compilation de `prog` avait lieu après la dernière modification de `prog.c`).

Taper sur le terminal

make

pour compiler tous les trois programmes (si nécessaire).

Taper

make prog2

si vous voulez juste recompiler prog2

Compilation avec make

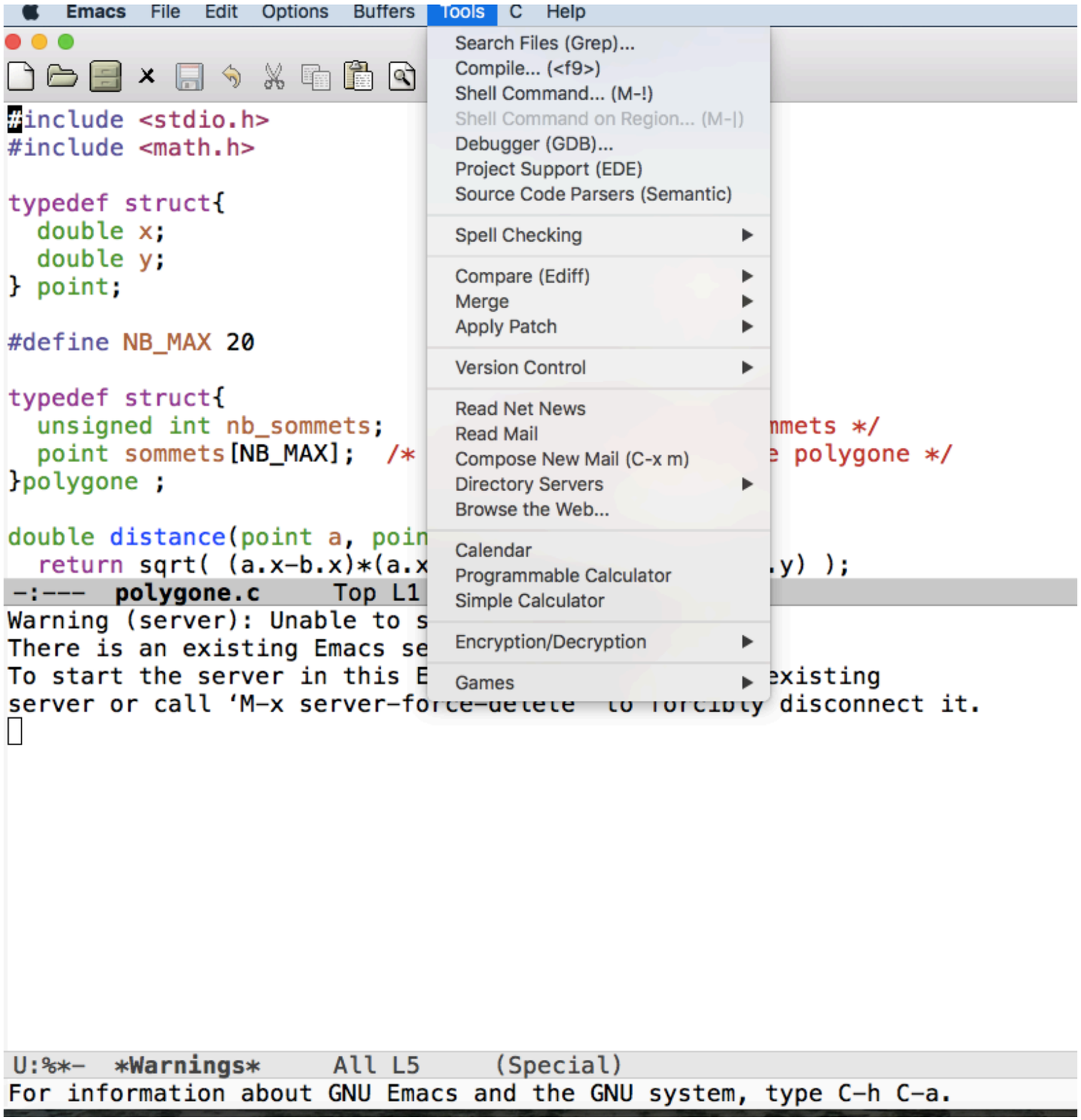
taper

```
make cleanall
```

pour exécuter la commande rm qui suit cleanall dans Makefile et qui supprime le fichiers exécutables et le fichier qui termine par tilde laissé par emacs

Compiler avec make et emacs :

sélectionner Tools -> Compile



```
} point;

#define NB_MAX 20

typedef struct{
  unsigned int nb_sommets;         /* le nombre d
  point sommets[NB_MAX];  /* le tableau de somme
}polygone ;

double distance(point a, point b){
  return sqrt( (a.x-b.x)*(a.x-b.x)+ (a.y-b.y)*(a
-:---  polygone.c     Top L1     (C/*l Abbrev)
Warning (server): Unable to start the Emacs serv
There is an existing Emacs server, named "server
To start the server in this Emacs process, stop
server or call `M-x server-force-delete' to forc
```

```
U:%*-  *Warnings*     All L5     (Special)
Compile command: make -k
```

taper **Enter** quand la commande **make -k** s'affiche en bas de la fenêtre d'emacs

polygone.c

```c
typedef struct{
  unsigned int nb_sommets;        /* le nombre de sommets */
  point sommets[NB_MAX];  /* le tableau de sommets de polygone */
}polygone ;

double distance(point a, point b){
  return sqrt( (a.x-b.x)*(a.x-b.x)+ (a.y-b.y)*(a.y-b.y) )

}

double perimetre(polygone poly){
  double s =0;
  for(int i = 0; i < poly.nb_sommets-1; i++){
    s += distance(poly.sommets[i], poly.sommets[i+1]);
  }
  s += distance(poly.sommets[poly.nb_sommets-1], poly.sommets[0]);
```

```
-:---  polygone.c     12% L17    (C/*l Abbrev)
-*- mode: compilation; default-directory: "~/Documents/Enseignement/C/2018/prog
COURS02/" -*-
Compilation started at Sat Sep 15 21:02:55

make -k
for  file in tableau structures enums literal mirror mirror_ret polygone ; do\
          gcc -Wall -pedantic -std=c11  -lm $file.c -o $file  ;\
        done
polygone.c:17:58: error: expected ';' after return statement
  return sqrt( (a.x-b.x)*(a.x-b.x)+ (a.y-b.y)*(a.y-b.y) )
                                                         ^
                                                         ;
1 error generated.
make: *** [tableau] Error 1

Compilation exited abnormally with code 2 at Sat Sep 15 21:02:56
U:%*-  *compilation*   All L8     (Compilation:exit [2] [1 0 0])
```

Les erreurs s'affichent dans la deuxième partie de la fenêtre, il suffit de cliquer sur le message d'erreur pour que emacs se positionne sur la ligne correspondant. Corrigez les erreurs un après l'autre et recompilez.