Algorithmique et Programmation C (CS101) – Examen Session 1 2023-24

Durée totale : 1 heure 30 / Tiers Temps = 2 heures

Toute communication (orale, téléphonique, par messagerie, etc.) avec les autres étudiant·e·s est interdite. Aucun document autorisé (on fournit une fiche de syntaxe C)

— Il est **indispensable de rédiger vos réponses, et d'expliquer vos programmes.** Un programme non expliqué n'aura pas la totalité des points.

— Les questions ne **SONT PAS** de difficulté croissante.

— Cet examen est **très probablement trop long**, le barème en tiendra compte. Les points donnés dans les questions (pour un total de 28 points) sont des indications de pondération respective (difficulté, longueur) de chaque question.

— Ne pas rendre l'énoncé.

Solution: En rouge, mes commentaires après correction. La fiche de syntaxe C était fournie, il y a trop d'erreurs de syntaxe et de grosses erreurs avec le passage de paramètres tableaux, qui SONT SUR LA FICHE :

```
int t[N]=fonction_qq(); // non!

fonction2qq(int t2[N]); // non plus.
```

De plus, les boucles if n'existent pas.

Sauf mentionné, il est **interdit** d'utiliser des pointeurs (et donc autre chose que des tableaux statiques).

1 Programme mystère

Soit la suite récursive suivante : $u_0 = u_1 = 1$, $u_n = 2 + u_{n-1} + u_{n-2}$

Question #1 (1 point)

Écrire une fonction C récursive `int toto(int x)` qui calcule et renvoie le $x$ième terme de la suite u.

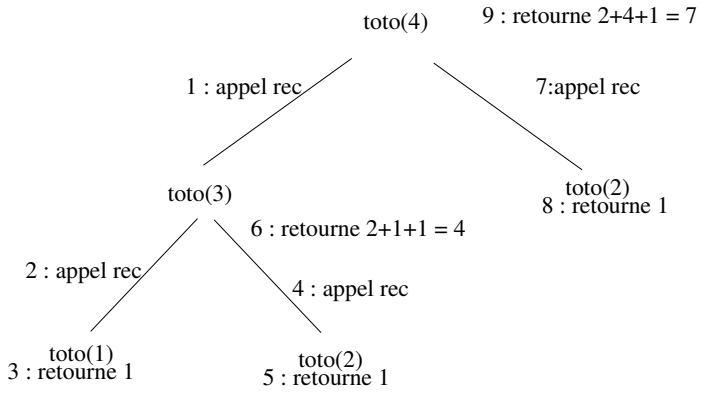
Solution: Je demandais une fonction récursive, donc 0 point pour des calculs non récursifs. Une fonction récursive utilise des appels à son propre code. Ce n'est pas parce qu'on calcule une suite récurrente/récursive, que notre fonction est forcément récursive..

```
int toto(int x){
  if (x<0) return -1; // cas non demandé
  else if (x==1 || x==0) return 1;
  else return 2+toto(x-1)+toto(x-2);
}
```

Question #2 (1 point)

Dessiner l'arbre des appels récursifs pour l'appel `toto(4)`. Numéroter les appels récursifs et les retours des appels récursifs. Quel est le résultat du calcul ?

Solution: Quand on demande un arbre, svp, dessinez ... un arbre, de ce style :



Attention erreur sur mon dessin, toto(2) doit encore faire 2 appels recursifs, et retourner 4. La valeur finale est 13.

Question #3 (1 point)

Expliquer rapidement pourquoi l'algorithme de calcul de `toto(x)` n'est pas optimal, en donnant un exemple de calcul redondant.

Solution: Le calcul de `toto(2)` est effectué deux fois dans le dessin précédent.

Soit a_n le nombre d'appels récursifs lors de l'appel à la fonction `toto` avec le paramètre n.

Question #4 (1 point)

Donner une formule récursive pour a_n (pour $n \geq 2$).

Solution: Pas très difficile, pour calculer toto(n), on a besoin de

- calculer toto(n-1) : donc on a besoin de a_{n-1} appels récursifs pour cela
- calculer toto(n-2) : on a besoin de a_{n-2} appels récursifs pour cela.
- et on ajoute les 2 appels récursifs.

On obtient donc : $a_n = a_{n-1} + a_{n-2} + 2$. Ceci est bien une formule récursive, puisqu'elle utilise a_{n-1} pour définir a_n.

Question #5 (1 point)

On admet que la formule précédente se résoud en $a_n \sim (\frac{1+\sqrt{5}}{2})^n$. Conclure sur la complexité de la fonction `toto(x)`.

Solution: Attention mauvaise terminologie de ma part, j'aurai du écrire "conclure sur la complexité de la fonction toto". La fonction est donc de complexité exponentielle (la quantité sous l'exposant est clairement >1), ie $O(2^x)$, avec x l'argument passé à toto.

On désire maintenant supprimer les calculs redondants. Soit N une constante symbolique positive. On suppose dans la suite que $x < N$.

Question #6 (1 point)

Écrire une fonction `void toto2(int t[N], int x)` qui remplit le tableau t passé en paramètre avec les valeurs de la suite u_i, pour i de 0 à $x \geq 2$ compris. (Ne pas utiliser la fonction `toto`)

Solution:

```
void toto2(int t[N], int x){
  int i;
  t[0] = 1;
  t[1] = 1;
  for (i=2; i<= x; i++)
    {
      t[i]=2+t[i-1]+t[i-2];
    }
}
```

Il était attendu ici une fonction impérative, sans calculs redondants, donc.

Question #7 (1 point)

Combien d'étapes de calcul (tours de boucles) effectue votre fonction en fonction de x ?

Solution: $x - 1$ (prendre par exemple $x = 5$, et trouver 4)

Question #8 (2 points)

Programme principal Écrire (toujours en C) un programme complet qui :

- Définit une constante symbolique `N` fixée à 21.
- Définit les fonctions écrites dans les questions précédentes.
- (Questions suivantes : dans le *main*) Définit un tableau t statique N. Il n'est pas nécessaire d'initialiser ce tableau.
- Demande un entier k à l'utilisateur-rice ($0 \leq k < N$).
- Utilise la fonction `toto2`.
- Imprime la valeur calculée pour u_k.

Solution:

```
#include <stdio.h>
#define N 21

int toto(int x) ...//inutile de recopier ...

void toto2(int t[N], int x) ... //idem ...

int main(){

  int t[N];
  int k;
  scanf("%d", &k);
  toto2(t, k);
  printf("u_%d␣=␣%d\n",k,t[k]);


  return 0;
}
```

La fonction toto2 est de type de retour *void*, donc elle ne renvoie aucun résultat. Vous ne pouvez donc pas récupérer une valeur du tableau comme ceci : `res=toto2(...)`. Il y a trop d'erreurs dans l'usage de scanf, qui ... est sur la fiche de syntaxe fournie !

2 Algorithmique : un problème électoral

D'après un problème X2005.

Ce problème consiste à faire le décompte du résultat des élections dans le pays X. La règle électorale y est très laxiste, puisqu'on peut être élu·e sans s'être présenté·e. Plus prosaïquement, chaque votant i ($0 \leq i < N$) vote pour $t[i]$, où $t[i]$ est un entier positif quelconque (les habitants du pays X sont identifiés par un entier représentant leur numéro de sécurité sociale, entre 0 et $N-1$ compris).

Contrainte supplémentaire : dans tout le problème, il est interdit de déclarer des tableaux auxiliaires : vos fonctions n'auront aucune variable locale de type tableau. De plus les tableaux passés en paramètres seront des tableaux statiques de taille N, N supposée une constante symbolique déclarée à l'avance.

2.1 Dépouillement du premier tour

Dans le pays X, seul· le·a candidat·e ayant obtenu strictement plus de 50% des voix exprimées est élu·e au premier tour des élections.

Question #1 (1 point)

Écrire une fonction `int count_occ(int t[N], int el)` qui calcule et renvoie le nombre d'occurrences de l'élément `el` dans le tableau d'entiers t.

Solution:

```
int count_occ(int t[N], int el){
  int count = 0;
  for (int i=0; i<N-1 ; i++){
    if (t[i]==el) count++;
  }
  return count;
}
```

Question #2 (1 point)

Écrire la fonction `int gagnante_tour1` qui calcule et renvoie (le numéro de) la personne gagnante du premier tour si elle existe. Si elle n'existe pas, on retournera -1. On se contentera d'une fonction faisant le calcul en temps quadratique par rapport à N (au pire), en utilisant un algorithme de la forme :

```
pour toute personne (
    compter c le nombre de votes en sa faveur
    si c>n/2 terminer (la personne a gagné au premier tour)
)
```

Solution:

```
int gagnante_tour1(int t[N]){
  // cf algo donné.
  int c;
  for (int p=0; p<N; p++){ // pour toute personne
    c = count_occ(t, p);
    if (c>N/2) return p;
  }
  return -1; // pas de gagnante au premier tour.
}
```

Il est préférable, toujours, d'utiliser les fonctions écrites dans les fonctions précédentes. C'est plus lisible, et ne propage pas les erreurs.

Question #3 (1 point)

Supposons le tableau t trié en ordre croissant, vérifiant $t[0] \leq t[1] \leq \ldots \leq t[N-1]$. Expliquer (en français) comment compter les votes concernant une même personne.

Solution: Les votes concernant une même personne sont sur des cases adjacentes du tableau t. En gardant cette valeur en mémoire jusqu'à ce qu'elle change, on est certains de compter exactement le nombre de ces valeurs pour le tableau entier.

Question #4 (2 points)

En supposant le tableau des votes triés, écrire une fonction `int gagnante_tour1_aux` qui retourne la personne gagnante du premier tour (ou -1) en temps linéaire par rapport à N.

Solution:

```
int gagnante_tour1_aux(int t[N]){
  int g = t[0];
  int v = 1;
  for (int i=1; i<N; i++){
    if (t[i]==g) v++; // on augmente le compteur de g
    else {
      // on passe à un autre g
      g = t[i];
      v = 1;
    }
    if (v>N/2) return g;
  }
  return -1;
}
```

Une copie me donne l'algo suivant, qui fonctionne aussi, saurez-vous expliquer pourquoi ?

```
int gagnante_tour1_aux(int t[N]){
  for (int i=0; i<N/2; i++){
    if (t[i]==t[i+N/2]) return i;
  }
  return -1;
}
```

Question #5 (1 point)

Si on sait trier un tableau de taille n en temps $n \ln n$, quel est l'ordre de grandeur du temps pris par le dépouillement du premier tour ?

Solution: On commence par trier, donc, et ensuite on utilise la fonction linéaire de la question précédente, ce qui donne une complexité : $O(N \ln N) + O(N) = O(N \ln N)$, avec N la taille du tableau des votes (le nombre de personnes).
Attention, on trie le tableau INITIAL, pas le nombre d'occurrences !

Pour réaliser un tri en $n \ln n$, nous désirons utiliser la fonction de bibliothèque `qsort`, dont une documentation (manuel en français) est fournie ci-dessous.

NOM

qsort - Trier une table

SYNOPSIS

```
#include <stdlib.h>

void qsort(void *base, size_t nmemb, size_t size,
           int(*compar)(const void *, const void *));
```

DESCRIPTION

La fonction **qsort**() trie une table contenant *nmemb* éléments de taille *size*. L'argument *base* pointe sur le début de la table.

Le contenu de la table est trié en ordre croissant, en utilisant la fonction de comparaison pointée par *compar*, laquelle est appelée avec deux arguments pointant sur les objets à comparer.

La fonction de comparaison doit renvoyer un entier inférieur, égal, ou supérieur à zéro si le premier argument est respectivement considéré comme inférieur, égal ou supérieur au second. Si la comparaison des deux arguments renvoie une égalité (valeur de retour nulle), l'ordre des deux éléments est indéfini.

VALEUR RENVOYÉE

La fonction **qsort**() ne renvoie pas de valeur.

Il n'est pas nécessaire de comprendre les pointeurs de fonctions pour la suite L'invocation de cette fonction de tri sera faite comme ceci (t est le tableau d'entiers à trier) :

```
qsort(t, N, sizeof(int), &compare);
```

Question #6 (1 point)

Recopier et compléter la fonction compare sur des variables entières. *On admettra que les deux premières lignes du corps de la fonction permettent de récupérer deux valeurs entières à partir des deux pointeurs pa et pb.*

```
int compare(const void* pa, const void* pb){
  int a = *((int*) pa);
  int b = *((int*) pb);
  // TODO comparer les entiers a et b.
  ...
}
```

Solution:

```
int compare(const void* pa, const void* pb){
  int a = *((int*) pa);
  int b = *((int*) pb);

  if (a==b) return 0;
  if (a>b) return 1;
  else return -1;
}
```

Question #7 (1 point)

Donner le nom et le principe (en 5 lignes, pas de code, merci) d'un des tris vus en cours (à l'exception du tri-bulle). Quel est sa complexité en nombre de comparaisons au pire ?

Solution: cf le cours. Les tri insertion et sélection sont simples à expliquer, et leur complexité en nombre de comparaisons est au pire quadratique en la taille du tableau t. ($O(N^2)$) Une complexité compte des opérations, ou des conditionnelles, ou etc, mais il faut le dire. Dans l'explication des tris, faites attention à bien utiliser des échanges de cases/valeurs, sinon rien ne garantit que vous n'écrasez pas de valeurs.

Question #8 (1 point)

En utilisant les fonctions précédentes, implémenter une fonction `gagnante_tour1_bis` de complexité égale à la réponse à la question 5).

Solution: Il suffit de recoller les bouts :

```
int gagnante_tour1_bis(int t[N]){

  // appel de la fonction qsort, qui trie en place
  qsort(t, N, sizeof(int), &compare);
  return (gagnante_tour1_aux(t));
}
```

2.2 Dépouillement du deuxième tour

Au deuxième tour des élections, la personne candidate ayant obtenu le plus de voix est élue. Si il y a égalité de voix, toutes les personnes candidates ayant obtenu le plus de voix sont élues (ex-aequo).

Question #9 (2 points)

Écrire une procédure `void gagnantes_tour2` qui imprime (affiche) le(s) personnes gagnante(s) du deuxième tour (le tableau t n'étant pas trié). On se contentera d'une fonction faisant le calcul en temps quadratique par rapport à n (et refaisant des calculs déjà réalisés) :

- `calculer maxic le maximum de voix pour une personne`
- `imprimer les personnes dont le nombre de voix est égal à maxic`

Solution:

```
    void gagnantes_tour2(int t[N]){
      int c;
      int maxic=0;
      for (int p=0; p<N; p++){ // pour toute personne
        c = count_occ(t, p);
        if (c>maxic) maxic=c;
      }
      printf("maxi␣nb␣de␣voix␣=␣%d\n", maxic);
      // parcours et affichage
      for (int p=0; p<N; p++){ // pour toute personne
        c = count_occ(t, p);
        if (c==maxic) printf("%d␣gagne␣au␣tour␣2\n", p);
      }
    }
```

Question #10 (2 points)

Plus difficile On suppose maintenant le tableau t trié en ordre croissant (avec donc la même remarque structurelle qu'à la question 3). Écrire une procédure `void gagnantes_tour2_aux` qui

imprime le(s) personnes gagnante(s) du deuxième tour en temps linéaire par rapport à N. *Toute solution partielle rédigée (notamment le calcul en temps linéaire du nombre de voix maximum) sera valorisée.*

Solution: Cette question est un peu moins immédiate que la précédente, car les boucles imbriquées sont maintenant interdites. Commençons par calculer le nombre maximum de votes pour une personne donnée (en considérant des cases adjacentes) :

```
// Récupérer le nb de voix max (vmax) en 1 parcours.
  int g = t[0];
  int v = 1;
  int vmax=1;
  for (int i=1; i<N; i++){
    if (t[i]==g) {
      v++; // on augmente le compteur de g
      if (v>vmax) vmax = v;
    }
    else {
      // on passe à un autre g
      g = t[i];
      v = 1;
    }
  }
```

Pour l'impression, on peut refaire la même chose. La correctrice n'a pas trouvé de moyen plus simple.

```
    g = t[0];
    v = 1;
    for (int i=1; i<N; i++){
      if (t[i]==g) {
        v++; // on augmente le compteur de g
      }
      else {
        // on regarde si v=vmax (pour t[i-1])
        if (v==vmax) printf("%d gagne au tour 2\n", t[i-1]);
        // on passe à un autre g
        g = t[i];
        v = 1;
      }
    }
    // attention au cas du dernier.
    if (v==vmax) printf("%d gagne au tour 2\n", t[N-1]);

  }
```

Notez qu'il n'y pas de question bilan pour cette partie.

Solution: Un collègue propose en temps linéaire une solution que je vous laisse méditer (au delà de ce qui est exigible en première année, bien sûr) :

```
void gagnantes_tour2 (int t[N]) {
   for(int i = 0; i < N; i++)
       t[t[i] & 0xffff] += 0x10000;

   int max = 0;
   for(int i = 0; i < N; i++)
```

```
        max = (max > t[i] ? max : t[i]) & ~0xffff;

    for(int i = 0; i < N; i++) {
        if(t[i] >= max) printf("%d␣gagne␣au␣tour␣2\n", i);
        t[i] &= 0xffff;
    }
}
```

2.3 Dépouillement rapide du premier tour

On suppose à nouveau le tableau t non trié. On utilise ici la notation $t[i..j]$ pour parler du sous-tableau de t entre les indices i et j compris. Un multi-ensemble E d'éléments de D est un ensemble où on autorise chaque élément à apparaître plusieurs fois (formellement, E est une fonction de D dans $\mathbb{N}$ où $E(d)$ est le nombre d'occurrences de d dans E). Par exemple, $\{0,2,2,3,0,0,0,5,0\}$ est un multi-ensemble. On dit que x est un élément majoritaire dans E contenant n éléments si x apparaît strictement plus que $\frac{n}{2}$ fois dans E. La personne gagnante du premier tour est donc l'élément majoritaire (s'il existe) dans le multi-ensemble $t[0], t[1], \ldots t[N-1]$. Dans ce dernier cas, on dira plus simplement que x est majoritaire dans le tableau t.

Question #11 (1 point)

Démontrer que si x est majoritaire dans le multi-ensemble E à n éléments et que y et z sont deux éléments distincts ($y \neq z$) de E, alors x est majoritaire dans le multi-ensemble obtenu en retirant de E les éléments y et z.

Solution:

Soit k le nombre d'occurrences de x dans E. Comme x est majoritaire dans E, $k > \frac{n}{2}$. Considérons l'ensemble $E' = E \setminus \{y, z\}$ (de cardinal $n-2$). En retirant deux éléments de E différents, on a retiré au plus une occurrence de x, donc il y a au moins $k-1$ occurrences de x dans E'. Comme $k - 1 > \frac{n}{2} - 1 = \frac{n-2}{2}$, il suit que x est majoritaire dans E'.

Pour améliorer le résultat de la première partie, on commence par écrire une fonction qui calcule et retourne la personne gagnante si elle existe, et qui renvoie n'importe quelle personne sinon. On propose la fonction suivante, que nous allons analyser dans la suite.

```
1  int gagnante_potentielle(int t[N]){
2    int g = t[0];
3    int v = 1;
4    for (int i=1; i<N; i++){
5      if (v==0) {
6        g = t[i];
7        v = 1;
8      }
9      else if (t[i]==g)
10       v++;
11     else
12       v--;
13   }
14   return g;
15 }
```

Question #12 (1 point)

Soit le vote suivant : $[1,2,1,1,2]$ (et $N = 5$). Quelle valeur l'appel `gagnante_potentielle(t)` renvoie-t-il ? *On listera les étapes d'exécution de cette fonction, en distinguant g et v.*

Solution:

— candidate gagnante $g = 1$, avec une occurrence $v = 1$ au début
— comme $t[1]$ est différent de g, v devient nul. (on a grosso-modo supprimé de notre parcours une paire d'élements différents ; 1, 2).
— pour i valant 2, la nouvelle candidate g vaut de nouveau 1, et v reprend la valeur 1.
— pour i valant 3, comme on rencontre un nouveau 1, g est inchangé et v vaut 2.
— pour i valant 4, on rencontre 2, donc g inchangé.
— La fonction renvoie g, donc 1. Et il se trouve que 1 est bien majoritaire.

Question #13 (2 points)

Difficile Prouver qu'à chaque tour de boucle, le multi-ensemble composé des éléments de t vus jusqu'à présent peut être simplifié (en retirant des paires d'élements distincts) en un multi-ensemble ne contenant que v occurrences de g.

Solution: On considère les sous tableaux déliminés par les $v == 0$, c'est à dire les sous-tableaux dans lesquels la valeur g est inchangée, et on les appelle segments.
A la fin d'un tour de boucle, v vaut donc le nombre de valeurs de g (courant) en excès par rapport aux non g, sur le segment courant. On peut donc supprimer toutes les paires $g/nong$ vues dans le tronçon courant, et il reste v occurrences de g.
De plus, tous les segments précédents comportaient un nombre identique de g'/non g', avec g valant la première valeur du tableau sur le segment.
Ceci finit à prouver la propriété voulue.

Question #14 (1 point)

En déduire que si a a une gagnante, alors c'est forcément la valeur retournée par la fonction `gagnante_potentielle`.

Solution: Petite imprécision dans l'énoncé, je voulais évidemment parler d'une gagnante majoritaire !
À la fin de la dernière boucle, si l'on considère le multiensemble associé à a, on peut, en supprimant les paires d'éléments différents, obtenir un multiensemble $A_{fin} = \{g, \dots g\}$ composé de v copies de la valeur g. D'après la question 11, s'il existe un élément majoritaire, c'est forcément g.

Question #15 (1 point)

En déduire la fonction `gagnante_tour1_ter` qui retourne la personne gagnante du premier tour en temps linéaire par rapport à N. *Indication : faire deux passes sur le tableau t ; dans la première passe, on cherche une personne majoritaire ; dans la deuxième passe, on vérifie qu'elle est bien majoritaire.*

Solution: Il n'y a à ce stade qu'à recoller les morceaux.

```
int gagnante_tour1_ter(int t[N]){
  int g = gagnante_potentielle(t);
  int c = count_occ(t,g);
  if (c>N/2) return g;
  else return -1;
}
```

Pour aller plus loin. Il existe un algorithme récursif de type diviser pour régner linéaire également : cherchez "élément majoritaire dans un tableau" dans votre moteur de recherche préféré.

1 Premier programme

```c
/* Source code (hello.c) */
#include <stdio.h>

int main(){
  printf("Hello World!\n");
  return 0;
}
```

Compilation : dans un *terminal* :

```
clang-14 hello.c -o hello
```

On peut remplacer `clang` par `gcc` et ajouter les options `-Wall` ou `-Werror`.

Exécution : si la compilation réussit (sans erreur), on peut alors exécuter ce binaire, toujours dans un terminal, avec `./hello`.

2 Structure générale d'un programme

(ici, un seul fichier source, les seules inclusions sont des entêtes de bib. standard)

```c
#include <stdio.h> //en-têtes de bibliothèques (sans ;)
#define TAILLE 100 //constante symbolique
int c;                          //définitions de variables globales (optionnel)

int mafonction(char u){         //définitions de fonctions (optionnel)
  ...
  return -42;
}

int main(){                     //fonction principale du programme (obligatoire)
  mafonction('u');
  ...
  return 0;                                          //ou EXIT_SUCCESS
}
```

3 Types de données

- **Types de base :** `int` (entier signé), `char` (caractère, sur 1 octet), `bool` (avec `stdbool.h`)
- **Type composé :** les tableaux statiques : `int tab[12]` par exemple.

4 Conditionnelles

```c
  if (x%2==0){
    //instructions dans le cas "x pair"
  } else {
    //autre cas
  }
```

La deuxième branche est optionnelle.

5 Boucles for

À utiliser en priorité lorsque l'on connaît à l'avance le nombre d'itérations.

Exemple : somme des éléments de 1 à 12 :

```c
int i;
int s = 0;
for (i=1; i<=12; i++) // attention aux points-virgules (;)
  s = s +i ;
```

S'il y a plusieurs instructions dans le *corps*, alors les accolades sont utiles :

```c
for (int j=12; j>=0; j--) {// boucle décroissante
  // instruction1
  // instruction2
}
```

6 Boucles while

Lorsque la condition est plus complexe, et qu'on ne peut pas facilement borner le nombre d'itérations :

```c
int c = 0;
while(b > 1){
  b = b/2;
  c++;
} // calcule le log_2 de b.
```

La condition du while, qui est une *condition booléenne* (s'évalue à vrai/faux) peut être composée avec non (`!`), ou (`||`), et (`&&`). Attention aux parenthèses.

7 Fonctions

Une fonction sert à créer une portion de code que l'on peut appeler plusieurs fois.

Déclaration et implémentation d'une fonction *nommée* `mafonction`, qui a un unique *paramètre* de type caractère, et qui renvoie ("retourne") un entier (son *type de retour* est `int`) :

```c
int mafonction(char c){
  return -42;
}
```

Une fonction peut ne rien retourner, le type de retour est alors `void`. Une fonction sans paramètre (appelée *procédure*) a `void` tout seul entre les parenthèses. Les modifications de paramètres (sauf tableaux) ne sont pas enregistrées à l'extérieur de la fonction.

Appel d'une fonction (à l'intérieur d'une autre fonction, le main, par exemple) :

```c
int resu; // variable du type de retour
resu = mafonction('u'); //le paramètre formel est remplacé par une valeur
```

Alternativement, on peut utiliser le résultat directement :

```c
printf("Le resultat est = %d", mafonction('Z')); //%d car le résultat est un entier (int)!
```

8 Tableaux

Déclaration d'un tableau :

```c
int t[12]; // comme variable locale d'une fonction
```

ou alors, avec une constante symbolique

```c
#define N 100 // au début du programme
...
int tab[N]; // quelque part dans une fonction
```

⚠ Le tableau n'est **pas** automagiquement initialisé. par contre :

```c
int tab[23]={1,12}; // le reste des cases est initialisé à 0.
```

Lecture, écriture

```c
t[2] = ...; // écriture dans la 3ième cellule
... = t[9]; // lecture de la 10ième cellule
```

Passage de paramètre tableau

```c
imprime_tableau(t); // et pas t[N] !
```

⚠ Toute modification du contenu tableau est enregistrée.

Tableaux de caractères Pas de "string" en C, mais des tableaux de `char` avec une sentinelle :

```c
// déclaration et initialisation
char s[21]="hello"; // ajoute `\0` à la fin
```

On peut utiliser les fonctions de la bibliothèque `string`, ou parcourir comme ceci :

```c
while(s[i] != '\0') {...
```

9 Entrées-sorties

Sortie : impression sur le terminal (printf a un nombre de paramètres variable) :

```c
printf("%d", x);// imprime la valeur entière contenue dans x
printf("Ou alors %c, %s\n", caractere, chaine);// autant de % que de variables à imprimer
```

Le premier argument est appelé *chaîne de formattage*.

Entrée : récupération d'une information entrée au clavier :

```c
scanf("%d",&x); // modifie la valeur de x (int) avec la valeur entrée au clavier
scanf("%c",&c); // idem, pour c un caractère (char)
scanf("%s", s); // une chaîne est un tableau, pas de &
```