université de Caen Basse-Normandie

ESIX MeSN
1ère année

TP Langage C

- Consignes générales pour les TP Page 1
- Variables, instructions de base, structures de contrôles, opérateurs Page 2
- Tableaux statiques, pointeurs, allocation dynamique Page 7
- Fonctions, programmation modulaire Page 11
- Structures, listes chaînées, Arbres Page 14
- Fichiers Page 18
- Programmation d'algorithmes de tri de données Page 20
- Approfondissement Page 23

Ph. Dorléans, Y. Chahir

CONSIGNES GENERALES POUR LE DEVELOPPEMENT DES PROGRAMMES

Les TP de langage C consistent à développer de multiples programmes abordant l'ensemble des concepts abordés en cours. Les TP se présentent sous forme d'exercices qui peuvent être réalisés indépendamment les uns des autres pour certains mais il est conseillé de les faire dans l'ordre. Vous devez faire tous les programmes.

Recommandation préliminaire : il est fortement recommandé de respecter les directives qui suivent. L'expérience montre que de nombreux étudiants perdent inutilement des points simplement parce qu'ils ne respectent pas rigoureusement les consignes.

1 - Remarques générales

Il est vivement conseillé de concevoir la structure générale du programme avant de se lancer dans le l'écriture du code C. Cette réflexion peut conduire à la conception d'un organigramme ou l'écriture du programme sous forme d'un pseudo-code.

1) Tous les programmes doivent être développés dans un même répertoire.
2) Pour chaque programme, vous devez créer un projet spécifique qui portera le numéro de l'exercice (exemple exo1.cbp).
3) Les programmes développés doivent :
   - être concis (pas d'instructions inutiles)
   - être écrits proprement (facilité de lecture du programme, indentation, choix du nom des variables, utilisation d'instructions appropriées au problème posé…)
   - respecter l'énoncé à la lettre
   - fonctionner correctement et dans tous les cas limites
   - produire des affichages simples et lisibles
   - être parfaitement commentés pour les programmes qui concernent les listes chaînées, les arbres, les algorithmes de tri de données et d'algorithmes divers et surtout s'il y a des astuces de programmation.
4) Pour la mise au point des programmes, n'oubliez pas d'utiliser toutes les fonctionnalités de débogage de CodeBlocks pour corriger vos erreurs éventuelles.

2 - Modalités de remise du travail

Lorsque l'enseignant le demandera, vous déposerez sur l'espace de cours ecampus votre travail sous forme d'une archive (fichier compressé). Cette dernière devra se nommer de la manière suivante :

Nom_du_binôme_date.zip

Tout retard sera pénalisé au niveau de la notation.

3 - La fraude

Vous avez bien évidemment le droit de communiquer avec vos collègues. Mais ne prenez pas le risque de rendre un code dont vous n'êtes pas l'auteur et que vous ne comprenez pas. La triche sera fortement sanctionnée. Le plagiat (récupération de solutions ou de contenu trouvés dans des livres ou sur le Web, même légèrement modifiés) est un acte grave, qui sera sanctionné comme il se doit. S'il n'est pas possible de faire la différence entre le copieur et le copié, l'étudiant transmettant son travail sera pénalisé de la même manière que l'étudiant s'en inspirant.

Variables, instructions de base, structures de contrôles, Opérateurs

Tous les programmes seront développés dans l'environnement Code::Blocks disponible en open source. Il fait partie des logiciels de type IDE (Environnement de Développement Intégré) que nous utiliserons dans le cadre de l'apprentissage du langage C. L'objectif des exercices proposés permettent de se familiariser à la syntaxe et à la structure de base de programmes en langage C. Seront étudiés les variables avec leur codage, l'utilisation des entrées/sorties standards permettant l'affichage et la saisie de données formatées, les opérateurs logiques et conditionnels ainsi que les structures de contrôles.

01 - Codage des entiers

Exécuter le programme suivant et analyser les résultats obtenus.

```
#include <stdio.h>

int main ( )
{
  char ch = 69, car = 'E';

  printf ("ch vaut %d  ou  %c \n", ch, ch);
  printf ("car vaut %c  ou  %d\n", car, car);
  return (0);
}
```

02 - Conversions implicites

Exécuter le programme suivant et analyser les résultats obtenus.

```
#include <stdio.h>

int main ( )
{
  int i = 0x1234, j;
  char d, e, f;
  float r = 12.789, x;

  j = r;
  x = i;
  d = i;
  e = r;
  f = i;
  printf("Conversion float -> int : r = %15.2f -> j = %d\n", r, j);
  printf("Conversion int -> float : i = %d -> x = %5.2f\n", i, x);
  printf("Conversion int -> char : i = %x -> d = %x\n", i, d);
  printf("Conversion float -> char : r = %5.2f -> e = %d\n", r, e);
  printf("Conversion int -> int : i = %d -> f = %d\n", i, f);
  return (0);
}
```

03 - Opérateurs de traitement binaire

Soit le programme suivant :

```
#include <stdio.h>

int main ( )
{
  short n = 45, p = 4, masque = 0;

  printf("Valeur de n avant modification:%x\n", n);
  masque = ~0;
  masque = masque <<  p;
  n = n & masque;
  printf("n modifie vaut : %x \n", n);
  return (0);
}
```

Mettre un point d'arrêt sur la ligne « masque = ~0; ». Exécuter le programme en pas à pas et visualiser le contenu des variables à chaque ligne exécutée. Interpréter le rôle de chaque opérateur. Que réalise ce programme ?

Proposer ensuite un masquage plus simple réalisant la même opération. La structure du nouveau programme est la suivante :

```
#include <stdio.h>

int main ( )
{
  short n = 45, masque = xxx;    // Valeur à déterminer

  printf("Valeur de n avant modification:%x\n", n);
  n = n & masque;
  printf("n modifie vaut : %x \n", n);
  return (0);
}
```

04 - Bit de signe.

Le programme élémentaire suivant permet de faire un test sur le signe de la variable « a ».

```
#include <stdio.h>

int main ( )
{
  char a = -56;      /* ou 56 */

  if (a < 0) printf("la variable a est negative.\n");
  else  printf("la variable a est positive.\n");

  return (0);
}
```

Réécrire ce programme de manière à ce que le test du signe de la donnée soit réalisé en utilisant le bit de poids fort de la variable « a ». Cela nécessite l'utilisation des opérateurs de traitement binaire.

05 - Codage des entiers de type char

Interpréter les résultats fournis par le programme suivant :

```
#include <stdio.h>

int main( )
{
 char a = 0x7F;
 unsigned char b = 0x7F;

 printf("a en décimal vaut: %d\n",a);
 printf("b en décimal vaut: %d\n",b);
 return (0);
}
```

Ensuite, initialiser ces deux variables a et b à la valeur 0x80. Conclusion.

Remarque importante :

Avant d'écrire les programmes suivants en langage C, vous devez, pour chaque exercice, définir en langage descriptif toutes les étapes permettant de répondre au problème exposé. Pour cela, utiliser un ensemble de mots clés et de structures permettant de décrire de manière complète et claire, l'ensemble des opérations à exécuter sur les données.

06 - Caractère

Ecrire un programme qui permet de faire la saisie d'un caractère en minuscule et qui indique à l'utilisateur s'il s'agit d'une voyelle ou d'une consonne.

07 - Degré Celsius - degré Fahrenheit.

Écrire un programme qui convertit les degrés Fahrenheit θ_f en degrés centigrades θ_c en utilisant la règle de conversion suivante : $\theta_c = \frac{5}{9}(\theta_f - 32)$. Le programme doit demander à l'utilisateur de saisir θ_f et doit afficher θ_c. Vérifier que le résultat soit correct ? Dans la négative, modifier le programme pour que celui-ci fonctionne correctement.

08 - Volume d'une sphère.

Écrire un programme qui calcule le volume d'une sphère dont le rayon R est donné. On rappelle que le volume d'une sphère est : $V = \frac{4}{3}\pi R^3$. Il faut noter que la valeur « π » est présente dans la librairie « **math.h** » qui se trouve dans le répertoire : **c:\Programmes (x86)\CodeBlocks\MinGW\include**. Il est donc nécessaire d'ouvrir ce fichier pour connaitre la déclaration de « π » et de l'inclure pour pouvoir l'utiliser.

09 - Suite de Fibonacci.

Le mathématicien italien Fibonacci s'est posé le problème de savoir combien de couples de lapins seraient engendrés au bout de "n" périodes de reproduction. Il supposa, pour cela, que chaque couple peut engendrer un nouveau couple à partir de la deuxième génération. Autre hypothèse : aucun animal n'est supposé mourir pendant la période étudiée. La formule de récurrence, qui permet de déterminer le nombre de couples représentant la $n^{ième}$ génération est donnée par :

$$U_n = U_{n-1} + U_{n-2}$$

En effet, les nombres correspondant à chaque génération peuvent être obtenus en considérant que seuls les couples de la $(n-2)^{ième}$ génération peuvent être féconds. Soit U_{n-2} ce nombre. Pour obtenir le nombre de la $n^{ième}$ génération, il faut donc y ajouter le nombre déjà existant de couples de la $(n-1)^{ième}$ génération.

Ecrire un programme calculant les différents termes de la suite de Fibonacci et la limite $V_n=U_n/U_{n-1}$. Dans ce programme, les variables représentant U_n, U_{n-1} et U_{n-2} devront être déclarées en **int** et V_n en **double**. Il faut penser à initialiser U_{n-1} à 1 et U_{n-2} à 0. Après exécution du programme, l'affichage doit être semblable à celui-ci-dessous.

```
un = 1 , vn = 1.0000000000
un = 2 , vn = 2.0000000000
un = 3 , vn = 1.5000000000
un = 5 , vn = 1.6666666667
un = 8 , vn = 1.6000000000
un = 13 , vn = 1.6250000000
un = 21 , vn = 1.6153846154
un = 34 , vn = 1.6190476190
```

10 - Calcul d'une moyenne et de l'écart type.

Ecrire un programme permettant le calcul de la moyenne et de l'écart type de notes fournies au clavier avec un dialogue de ce type :

```
Note 1 : 12
Note 2 : 15.5
Note 3 : 13.5
Note 4 : -1

Moyenne : 13,67      Ecart type : 1,43
```

Ce programme n'utilise pas de tableau. Le nombre de notes n'est pas connu a priori et l'utilisateur peut en fournir autant qu'il le désire. Pour signaler qu'il a terminé, on convient qu'il fournira une note fictive négative. Celle-ci ne devra naturellement pas être prise en compte dans le calcul de la moyenne. Le test d'arrêt de la boucle doit se faire sur le signe de la valeur saisie au clavier. Le programme doit traiter le cas où aucune note ne serait saisie.

On rappelle que si l'on a "n" valeurs, la moyenne (notée $\bar{x}$) et l'écart type (noté e) sont données par les relations suivantes :

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad e = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2 - \bar{x}^2}$$

Le calcul de la racine carrée d'un nombre peut être obtenu simplement en utilisant la fonction « **sqrt** » de la librairie « **math.h** » qui se trouve dans le répertoire suivant : **C:\Programmes (x86)\ CodeBlocks\MinGW\include**.

Les résultats des calculs de la moyenne et de l'écart type seront stockés respectivement dans les variables d'identificateur "**moyenne**" et "**ecart_type**" déclarées dans le programme.

11 - Factorielle.

Ecrire un programme permettant la saisie d'un entier "n" et le calcul de n! (factorielle de "n"). Le programme doit afficher un message d'erreur dans le cas où n < 0. Utiliser tout d'abord une boucle **for**. On rappelle que la factorielle d'un nombre entier est définie de la manière suivante :

$$n! = \begin{cases} \text{non défini} & \text{si } n < 0 \\ 1 & \text{si } n = 0 \\ 1 \times 2 \times 3 \times \cdots \ (n-1) \times n & \text{si } n > 0 \end{cases}$$

Nous utiliserons l'algorithme suivant :

```
Déclarer i ;     // compteur de boucle
Déclarer n ;     // variable dont on doit calculer n!
Déclarer fact ;          // variable qui contient le résultat

Initialiser les variables
Pour i allant de 1 à n faire
        fact = fact * i
Finpour
Afficher le résultat
```

Quelle est la plus grande valeur possible pour la variable "**n**" si "**n**" est déclaré en **int** ? Justifier cette valeur.

Modifier ensuite le programme de manière à effectuer le calcul par les instructions de bouclage **while** puis **do while**.

Tableaux statiques - Pointeurs - Allocation dynamique

Les exercices proposés concernent les tableaux uni et bidimensionnels (déclaration, utilisation, lecture et affichage des éléments d'un tableau, manipulation des données) en les déclarant tout d'abord en statique puis en dynamique. Seront abordés aussi les pointeurs, l'allocation dynamique en mémoire. Le principal intérêt de la déclaration dynamique est d'allouer ou de libérer de la mémoire en fonction des besoins de l'utilisateur et ceci en cours d'exécution d'un programme.

12 – Tableau statique à une dimension

Ecrire un programme déclarant un tableau unidimensionnel de réels (noté **tab**) dont la dimension par défaut est définie par une déclaration de type **#define**. Après avoir demandé à l'utilisateur le nombre souhaité de valeurs, initialiser le tableau par saisie des valeurs au clavier et afficher son contenu ainsi que les adresses correspondantes. Prévoir le cas où le nombre souhaité de valeurs est supérieur à la dimension par défaut.

L'affichage des valeurs doit être semblable à :

```
tab[0] = 1.000000        adresse = 22fab0
tab[1] = 2.000000        adresse = 22fab4
tab[2] = 3.000000        adresse = 22fab8
```

Conclure sur le codage d'un float.

Compléter ce programme de manière à calculer la moyenne ainsi que l'écart type des éléments.

13 – Tableau statique à une dimension : détermination du minimum (version 1)

Reprendre le programme de l'exercice 12 et le compléter de manière à déterminer le minimum des valeurs du tableau en se basant sur l'algorithme suivant :

```
constante max = 12 ;
variables tab[0 ... max-1], min : flottants ;
variables i, n : entiers ;

début
        Ecrire ( Nombre d'éléments ? ) ;
        Lire(n) ; Lire(tab) ;

        min = tab[0] ;
        pour i = 1 à (n-1) faire
            si tab [i] < min alors min = tab [i]
        finpour
fin

Ecrire (Le minimum des éléments du tableau est : ) ;
```

14 – Tableau statique à une dimension : détermination du minimum (version 2)

Reprendre le programme de l'exercice 12 et le compléter de manière à déterminer le minimum des valeurs du tableau en s'affranchissant de la variable "**min**". L'algorithme correspondant est donné comme suit :

```
constante max = 12 ;
variable tab[0 ... max-1] : flottant ;
variables indiceMin, i, n : entiers ;

début
        Ecrire ( Nombre d'éléments ? ) ;
        Lire(n) ; Lire(tab) ;

        indiceMin = 0 ;
        pour i = 1 à (n-1) faire
            si tab[i] < tab[indiceMin] alors indiceMin = i ;
        finpour
fin

Ecrire (Le minimum des éléments du tableau est : ) ;
```

15 – Tableau statique à une dimension : recherche d'un élément dans un tableau

Reprendre le programme de l'exercice 12 et le compléter de manière à déterminer si un élément se trouve ou ne se trouve pas dans le tableau non ordonné (non trié). L'algorithme correspondant est donné comme suit :

```
constante max = 12 ;
variable tab[0 ... max-1] : flottant ;
variables i, n : entiers ;
variable trouve : booleen ;

début
        … / …
        trouve = Faux
        pour i = 1 à (n-1) faire
            si tab [i] = element alors trouve = Vrai
        finpour
fin

si trouve alors Ecrire (L'élément xxx est dans le tableau)
sinon Ecrire (L'élément xxx n'est pas dans le tableau)
```

Modifier ensuite le programme de telle manière à éviter de continuer à explorer le tableau une fois que l'on a trouvé l'élément.

16 – Tableau statique à une dimension : recherche dichotomique d'un élément dans un tableau (OBLIGATOIREMENT trié)

La recherche dichotomique est une manière efficace et rapide de rechercher un élément dans une structure de données triée. Le principe est le suivant : comparer l'élément recherché avec l'élément se situant au milieu du tableau. Si les valeurs sont égales, la tâche est accomplie, sinon on recommence dans la moitié du tableau pertinente. Le nombre d'itérations de la procédure, c'est-à-dire le nombre de comparaisons, est logarithmique en la taille du tableau. Réaliser le programme correspondant à l'algorithme suivant :

```
constante dim = 100 ;
variable tab[0 ... dim-1] : Tableau de flottants classés ;
variable val : nombre flottant recherché ;
variables i : entier ;
variables inf, sup, milieu : entiers ;
variable flag = vrai : booleen ;


//initialisation
Créer le tableau
Saisir val

inf = 0
sup = dim-1


/*Boucle de recherche*/

  Tant que flag = vrai
  milieu = partie entière( (inf + sup) / 2 )

  Si val < tab[milieu] alors  sup = milieu -1
        Sinon inf = milieu+1
  FinSi

  Si (inf > sup) ou (val = tab[milieu]) alors flag = faux
  FinSi

//Affichage du résultat
  Si (val = tab[milieu]) alors Ecrire (La valeur val se trouve à la position milieu+1)
  Sinon Ecrire (La valeur val ne se trouve pas dans le tableau)
```

17 – Tableau à deux dimensions

Ecrire un programme déclarant un tableau bidimensionnel d'entiers (noté **tab2**) de 10 lignes et de 10 colonnes et le remplir de valeurs aléatoires comprises entre 1 et 100. Pour générer des nombres aléatoires, on dispose de la fonction standard "**rand()**" (déclarée dans **stdlib.h**) qui retourne une valeur aléatoire comprise entre 0 et une certaine valeur **RAND_MAX** (constante prédéfinie dont la valeur est fonction de l'environnement de développement).

Afficher le contenu du tableau tel que l'affichage soit semblable à ce qui suit.

80	41	79	41	4	24	1	43	87	97
40	47	96	61	87	96	1	60	72	56
59	52	76	68	13	11	1	36	56	53
86	75	95	99	19	51	43	67	75	30
8	20	33	60	67	71	42	5	63	19
69	74	19	40	61	3	32	42	95	67
79	15	26	23	89	50	66	40	5	6
68	70	92	42	21	5	91	35	62	5
97	19	70	95	98	8	93	23	58	7
66	55	10	38	39	77	72	33	6	94

Remarque :

En fait, il s'agit d'un tirage pseudo-aléatoire c'est-à-dire que si on exécute une seconde fois le programme, on obtient exactement la même série de valeurs. Cela est dû au fait que pour générer ces valeurs aléatoires, la fonction "**rand**" utilise une formule de récurrence de la forme $U_{n+1} = f(U_n)$ avec une valeur initiale U_0 initialisée à 1 dans la bibliothèque standard du C, ce qui explique la répétition. Il existe une fonction qui permet de changer cette valeur initiale : "**srand**". "**srand**" permet donc de réinitialiser le générateur de nombres aléatoires. Il faut l'utiliser avec un paramètre susceptible d'être modifié d'un appel du programme à l'autre : on utilise souvent le temps externe, mesuré en secondes (et qui change donc toutes les secondes), disponible grâce à la fonction "**time**" déclarée dans **time.h**. Introduire **srand(time(0))** dans votre programme ce qui permet d'obtenir des séquences différentes.

18 – Pointeurs

Ecrire un programme qui :

- déclare 2 deux variables flottantes **a** et **b** initialisées respectivement à 12,5 et –123,4.
- déclare 2 pointeurs **ptra** et **ptrb** destinés à pointer des flottants.
- Initialiser ces pointeurs tels que **ptra** et **ptrb** pointent respectivement **a** et **b**.
- Afficher l'adresse et le contenu des pointeurs **ptra**, **ptrb**, ainsi que les valeurs pointées par les pointeurs en utilisant uniquement le formalisme pointeur.

19 – Pointeur et tableau dynamique unidimensionnel

Reprendre l'exercice de base n°12 et modifier-le de la manière suivante :

- Saisir la dimension du tableau au clavier.
- Déclarer le tableau unidimensionnel par allocation dynamique en utilisant l'instruction **calloc** qui est définie dans **stdlib.h**.
- Remplacer tous les termes **tab[ ]** en introduisant le formalisme pointeur.

20 – Pointeur et tableau dynamique bidimensionnel

Reprendre l'exercice de base n°17 et modifier-le de la manière suivante :

- Saisir les dimensions du tableau au clavier (ligne – colonne)
- Déclarer le tableau bidimensionnel par allocation dynamique en utilisant l'instruction **calloc** qui est définie dans **stdlib.h**.
- Remplacer tous les termes **tab2[ ] [ ]** en introduisant le formalisme pointeur.

Fonctions, programmation modulaire

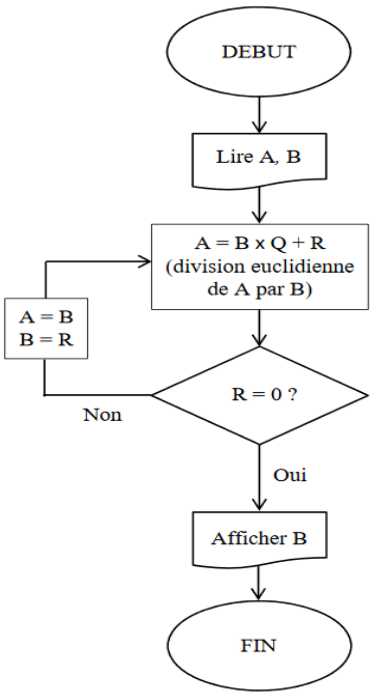
Les fonctions permettent de décomposer un programme en entités plus limitées et donc d'en simplifier à la fois la réalisation et la mise au point. Les exercices proposés permettent d'illustrer la notion de fonction, sa déclaration, son développement et son utilisation. On s'intéressera plus particulièrement au passage d'arguments d'entrée. Sera abordé également la programmation avec des fonctions situées dans des fichiers autres que celui contenant le programme principal.

21 – PGCD - PPCM (1ère version)

Le programme suivant a pour but de calculer le PGCD (plus grand diviseur commun) de 2 entiers saisis par l'utilisateur via une fonction d'identificateur PGCD. Le calcul du PGCD est basé sur l'algorithme d'Euclide qui est présenté par l'organigramme ci-dessous.

```
#include <stdio.h>

 xxx PGCD (xxx) ;

 main()
 {
  int resultat, a, b;

  printf("Entrer deux nombres entiers : ");

  scanf("%d", &a);
  scanf("%d", &b);

  xxx PGCD (xxx);
  printf("PGCD = %d\n", resultat);

  return (0);
 }

 xxx PGCD (xxx)
 {

 }
```

Algorithme d'Euclide



1 - Ecrire une fonction d'identificateur **PGCD** qui retourne le plus grand commun diviseur à partir de 2 nombres entiers positifs (penser à utiliser judicieusement l'opérateur modulo %). Compléter après le programme principal et vérifier son fonctionnement.

2 - Ecrire ensuite une fonction d'identificateur **PPCM** qui calcule le plus petit commun multiple de deux entiers non nuls. On rappelle que le PPCM est le plus petit entier strictement positif qui soit à la fois multiple de ces deux nombres. L'idée est ici de réutiliser la fonction PGCD en utilisant la propriété : PGCD(a, b) * PPCM(a, b) = a*b et en prenant bien soin de tester si l'un des deux entiers n'est pas nul. Tester le programme.

3 - Ecrire une fonction d'identificateur **premier** qui permet de dire si 2 entiers sont premiers entre eux ou pas. Dire que 2 entiers sont premiers entre eux c'est dire que leur PGCD est égal à 1.

4 - Modifier le programme tel que le PGCD soit affiché uniquement lorsque les entiers ne sont pas premiers entre eux.

5 - Ecrire une fonction **PPCM2** qui calcule le PPCM deux entiers non nuls mais contrairement à la fonction **PPCM** précédente, celle-ci ne retourne pas de résultat spécifique (déclaration **void** en argument de sortie) mais le résultat doit toujours être accessible et affiché dans le programme principal.

6 - Les 4 fonctions précédentes doivent maintenant être intégrées dans un autre fichier nommé **bibli.c**. Effectuer les modifications nécessaires à apporter afin que les opérations de compilation et de linkage se déroulent correctement. Tester le programme dans cette configuration.

22 – Passage d'un tableau à une dimension en argument d'entrée

1 - Reprendre le programme de l'exercice n°19 et le modifier de la manière à ce que la moyenne et l'écart type des éléments du tableau soient effectués par l'intermédiaire de 2 fonctions d'identificateur respectif **moyenne** et **ecart**.

2 - Ecrire une fonction d'identificateur **maxmin** qui renvoie le minimum et le maximum des valeurs du tableau. Compléter le programme principal pour que celui-ci appelle la fonction **maxmin**, l'affichage du minimum et du maximum doit se faire dans le programme principal.

23 – Passage d'un tableau bidimensionnel en argument d'entrée

1 - Reprendre le programme de l'exercice n°20 et le modifier tel que l'initialisation des éléments du tableau par des nombres aléatoires et l'affichage se fasse via deux fonctions d'identificateur respectif **remplissage** et **affichage**.

2 - Modifier le programme tel que l'allocation dynamique en mémoire s'effectue par l'intermédiaire d'une fonction d'identificateur **allocation**.

24 – Conversion d'une chaîne de caractères en un nombre entier

Ecrire un programme qui réalise la conversion d'une chaîne de caractères en un nombre entier. La saisie des caractères doit s'effectuer en utilisant la fonction **gets** et la conversion doit être réalisée par une fonction d'identificateur **conv_int** qui prend en argument d'entrée la chaîne de caractères saisie.

On suppose que la chaîne est correctement saisie (utilisation du clavier numérique) pour qu'elle représente bien un entier en décimal (exemple : la chaîne "741" donne l'entier équivalent 741). Pour calculer l'entier équivalent, il faut, pour chaque chiffre, multiplier par 10 la valeur accumulée puis ajouter la valeur du dernier chiffre ((7 * 10 + 4) * 10 + 1 = 741).

Retrouver le résultat en utilisant la fonction C « **atoi** » (Ascii TO Int) dont le prototype est : **int atoi (const char *)**.

25 – Crible d'Ératosthène

Le crible d'Ératosthène est un procédé qui permet de trouver tous les nombres premiers compris entre 2 et un certain entier naturel donné "**n**". Le pseudo-code suivant permet de trouver ces nombres premiers. La valeur "1" n'est pas considérée comme premier.

```
constante max entier 1000 ;
variable n, i, d : entiers ;
premier[0..max] : tableau d'entiers ;

début
…/…
premier[0] = Faux; premier[1] := Faux;

pour i = 2 à n faire premier[i] := Vrai ;  // tout le monde est candidat

pour i = 2 à n faire
    si premier[i] = Vrai    // → si premier[i] alors … car c'est un booléen
        alors
        pour d = i+1 à n faire
                    si (d mod i = 0) alors premier[d] = Faux ;
        finpour

pour i = 2 à n faire
    si premier[i] alors écrire(i)
fin
```

Ecrire une fonction d'identificateur **Eratosthene** qui permet de déterminer les nombres premiers compris entre 2 et "**n**" et une fonction d'identificateur **Affichage** qui permet de les afficher.

Ecrire le programme principal.

26 – Récursivité

Ecrire un programme qui appelle une fonction d'identificateur **factorielle** et qui, comme son nom l'indique, calcule la factorielle d'un entier mais de manière récursive. L'algorithme correspondant est défini comme suit :

```
u_n = n * u_(n-1)  et  u_0 = 1

Entrée : un entier positif n
Sortie : factorielle de n

Fact(n)
si n = 0 retourner 1
sinon retourner n x Fact(n-1)
```

Structures, listes chaînées, Arbres

Les structures consistent à regrouper dans une seule variable plusieurs éléments de types différents caractérisant la variable. Les exercices suivants permettent de se familiariser avec ce type de variables (définition, maniement avec ou sans fonction). Sera également traité l'utilisation de pointeur pour la gestion de listes simplement chaînées et les arbres binaires.

27 – Déclaration, initialisation et affichage

1 - Définir une structure nommée "**struct point**" possédant 2 champs correspondant respectivement à l'abscisse et à l'ordonnée d'un point : le 1er étant un champ de type **int** et le 2ème de type **float**.

2 - Compléter le programme en déclarant une variable de type **struct point** puis l'initialiser et l'afficher. Afficher ensuite la taille mémoire d'un point à l'aide de l'opérateur **sizeof**.

28 – Structures et fonctions

1 - Ecrire les fonctions permettant de saisir un point (identificateur "**saisie_point**") et d'afficher un point (identificateur "**affiche_point**"). Dans cette version, la fonction " **saisie_point**" retourne le point saisi. Modifier ensuite le programme principal de manière à ce qu'il appelle ces 2 fonctions.

2 - Créer une fonction d'identificateur "**symetrique**" qui renvoie en résultat un point de coordonnées opposées. Modifier ensuite le programme principal de manière à ce qu'il appelle cette fonction.

3 - Créer ensuite une fonction d'identificateur "**symetrique2**" qui réalise la même fonction que "**symetrique**" mais qui ne renvoie pas de résultat spécifique (déclaration **void** en argument de sortie). Cependant, le résultat doit toujours être accessible et affiché dans le programme principal. Modifier ensuite le programme principal de manière à ce qu'il appelle cette fonction.

29 – Utilisation de la déclaration typedef

Faire une 2ème version du programme précédent en introduisant la déclaration **typedef**.

30 – Tableaux de structure

1 - Définir un tableau statique permettant de stocker un ensemble de points (déclaration **typedef**). Demander le nombre de points à l'utilisateur, les initialiser dans le programme principal puis afficher tous les éléments du tableau par une fonction d'identificateur "**affichage_courbe**".

2 - Ecrire une fonction permettant de créer le tableau par allocation dynamique. Tester si l'allocation mémoire a réussi. Modifier le programme principal afin de tester cette fonction.

31 – Liste simplement chaînée

L'objectif du programme à développer est de gérer une liste chaînée constituée de nombres entiers par l'intermédiaire de 3 fonctions : **Creer_Liste** qui permet de créer la liste, **Afficher_Liste** qui permet d'afficher le contenu de chaque élément entier de la liste un à un en partant du premier élément et **Detruire_Liste** qui permet de supprimer la liste correctement en libérant la mémoire allouée.

Chaque élément de la liste aura la forme de la structure suivante :

```
struct liste {
        int Data;
        liste *Suivant;
};
```

Cette liste contient :

- Un entier qui correspond à la donnée que l'on veut stocker.
- Un pointeur vers un élément du même type appelé **Suivant** ce qui permet de lier les éléments les uns aux autres.

Le programme principal est le suivant :

```
typedef struct liste liste;

xxx Creer_Liste (xxx);
xxx Afficher_Liste (xxx *);
xxx Detruire_Liste (xxx);

int main()
{
 liste *Chaine;

 Chaine = Creer_Liste (5);          // La liste contient 5 éléments
 Afficher_Liste (Chaine);
 Detruire_Liste (Chaine);
 return (0);
}
```

Compte tenu des informations précédentes :

1 - Définir le prototype des fonctions **Creer_Liste**, **Afficher_Liste** et **Detruire_Liste**.

2 Ecrire le contenu de la fonction **Creer_Liste** tel que l'ajout d'un élément s'effectue en début de liste. Vérifier la création de la liste en exécutant le programme en pas à pas.

3 - Ecrire le contenu de la fonction **Afficher_Liste** et vérifier son fonctionnement en exécutant le code de la fonction en pas à pas.

4 - Ecrire le contenu de la fonction **Detruire_Liste** et vérifier son fonctionnement en exécutant le code de la fonction en pas à pas.

32 – Arbre binaire

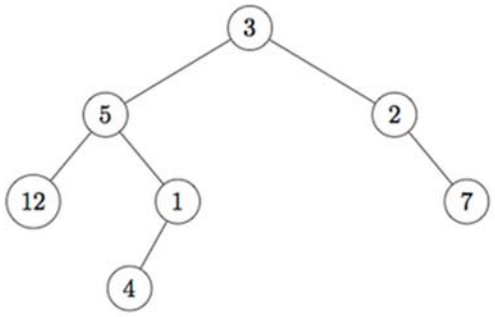
Rappel : en langage C, un nœud est représenté par une structure contenant des données ainsi que deux pointeurs vers les éventuels cellules filles. Un arbre possède donc une structure analogue à une liste chainée, à l'exception prêt que chaque cellule possède ici deux "suivants". Par convention, on appellera fils gauche et fils droit les deux cellules filles d'un nœud (quand elle existe ...).

La structure de donnée utilisée pour représenter un arbre binaire peut-être définie comme suit :

```
typedef int TypeDonnee;       // Type des datas (ici int)

typedef struct Nœud
{
   TypeDonnee donnee;         // donnee = valeur du nœud
    struct Nœud *fg;          // *fg = pointeur vers son fils gauche
    struct Nœud *fd;          // *fd = pointeur vers son fils droit
} TypeNoeud;
```

On définit également le pointer suivant : typedef TypeNoeud *Arbre;

1 - Ecrire une fonction **Arbre CreerArbre (TypeDonne val, Arbre fg, Arbre fd)** qui prend en argument une donnée de type **TypeDonnee** ainsi que deux arbres et qui renvoie un arbre dont la racine contient la valeur et les deux sous-arbres passés en paramètre. Pour créer l'arbre ci-après :



La déclaration sera :

```
Arbre arbre = CreerArbre(3, CreerArbre(5, CreerArbre(12, NULL, NULL), CreerArbre(1, CreerArbre(4, NULL,NULL), \
NULL)), CreerArbre(2, NULL, CreerArbre(7, NULL, NULL)));
```

2 - Parcours en profondeur d'un arbre binaire

Rappel : il existe différentes façons de parcourir un arbre binaire et notamment le parcours en profondeur. Pour parcourir un arbre non vide « **a** », on le parcourt récursivement son sous-arbre gauche, puis son sous-arbre droit, la racine de l'arbre pouvant être traitée au début, entre les deux parcours ou à la fin. Dans le premier cas, on dit que les nœuds sont traités dans un ordre préfixe, dans le second cas, dans un ordre infixe, et dans le troisième cas, selon un ordre postfixe.

Ecrire 3 fonctions d'affichage permettant un affichage des valeurs des nœuds d'un arbre binaire « a » selon les différents modes de parcours :

```
void AffichagePrefixe(Arbre a) ;
void AffichageInfixe (Arbre a);
void AffichagePostfixe (Arbre a);
```

Les différents parcours donnent pour l'arbre précédent :

Préfixe : 3,5,12,1,4,2,7
Infixe : 12,5,4,1,3,2,7
Postfixe : 12,4,1,5,7,2,3

3 - Affichage de la structure d'un arbre

Ecrire une fonction **void AfficheArbre (Arbre a)** permettant d'afficher les valeurs des nœuds d'un arbre binaire de manière à lire la structure de l'arbre. Un nœud sera affiché sous la forme : **{g,v,d}** où « **g** » est le sous-arbre gauche, « **v** » la valeur du nœud et « **d** » le sous-arbre droit.

L'arbre donné en exemple sera affiché sous la forme : **{{{_,12,_},5,{{_,4,_},1,_}},3,{_,2,{_,7,_}}}**. Les '_' indiquent des sous-arbres vides.

4 - Libération de la mémoire

Ecrire une fonction **void DetruitArbre (Arbre a)** qui libère la mémoire occupée par tous les nœuds d'un arbre binaire.

5 - Nombre de Nœuds

Ecrire une fonction **int NombreNoeuds (Arbre a)** qui calcule le nombre de nœuds d'un arbre binaire.

33 – Arbre Binaire de Recherche (ABR)

Un ABR est une structure de données qui permet de représenter un ensemble de valeurs si l'on dispose d'une relation d'ordre sur ces valeurs. La propriété fondamentale de cet arbre est que toutes les valeurs situées dans le sous-arbre gauche d'un ABR sont toutes inférieures à la valeur du nœud de l'arbre, et celle situées dans le sous arbre droit lui sont supérieures.

Trier un tableau de valeurs entières composé de « **n** » éléments (ce nombre sera choisi par l'utilisateur ...) dont les valeurs seront choisies de manière aléatoire. Pour classer les éléments composant le tableau, il suffit de construire un ABR correspondant aux éléments à classer. En utilisant, un parcours infixé de cet ABR, les valeurs de nœuds seront ensuite stockées dans un tableau de dimension adéquate dont vous afficherez en suite les valeurs.

Ajouter toutes les fonctions (que vous jugerez nécessaire) afin de permettre la suppression d'un nœud donné de l'ABR.

Gestion de fichiers

Il existe deux types de fichiers, les fichiers binaires et les fichiers textes. Les exercices proposés sont consacrés à la gestion de ces deux types de fichier et donc de se familiariser avec les fonctions de base de gestion de fichiers telles que "**fopen**", "**fscanf**", "**fprintf**", "**fread**", "**fwrite**", "**fclose**" ….

34 – Fichier binaire – fichier texte

L'objectif est de réaliser un transfert de données, initialement contenues dans un fichier binaire, vers un fichier texte. Pour vérifier si le transfert est réussi c'est-à-dire vérifier le contenu du fichier texte, vous pouvez utiliser par exemple l'éditeur de texte de l'environnement Code Blocks.

Tout d'abord, il est nécessaire de créer un fichier binaire contenant des données saisies au clavier.

- Déclarer un pointeur de fichier **pt1**, une chaîne de caractères **nom1** contenant le nom du fichier binaire ainsi qu'un pointeur sur entier **x**. Ce pointeur **x** doit pointer une zone mémoire contenant les données qui doivent être introduites dans le fichier binaire (opération d'écriture).

- Initialiser le pointeur **x** par allocation dynamique (**calloc**) et saisir des données. Saisir également le nom du fichier binaire.

- Ouvrir ce fichier binaire en écriture en utilisant la fonction **fopen.**

- Ecrire les données du tableau **x** dans le fichier binaire (**fwrite**) et fermer le fichier (**fclose**).

- Vérifier si le fichier est bien présent dans votre répertoire de travail.

Pour effectuer un transfert des données contenues dans le fichier binaire vers un fichier texte, procéder de la manière suivante :

- Déclarer un pointeur de fichier **pt2**, une chaîne de caractères **nom2** contenant le nom du fichier texte ainsi qu'un pointeur sur entier **y**. Ce pointeur **y** doit pointer une zone mémoire, différente de celle pointée par **x**.

- Initialiser le pointeur **y** par allocation dynamique (**calloc**). Saisir le nom du fichier texte.

- Ouvrir le fichier binaire en lecture et le fichier texte en écriture.

- Tant que la fin du fichier binaire n'est pas atteinte alors lire ce fichier en utilisant **fread**. Chaque donnée lue doit être introduite dans la zone mémoire pointée par **y**. Ensuite écrire les données lues dans le fichier texte en utilisant **fprinf.**

- Fermer les deux fichiers (**fclose**)

- Editer le fichier binaire puis le fichier texte. Conclusion.

35 – Fichier – Liste simplement chaînée

Reprendre le programme de l'exercice 31 sur la liste simplement chainée et ajoutez-y les fonctions suivantes :

1 - **void SauvegardeListe (... , ...)** : Fonction qui enregistre le contenu d'une liste chainée dans un fichier

2 - **void RecupereListe (... , ...)** : Fonction qui permet la récupération d'une liste chainée à partir de sa sauvegarde.

N.B. 1 : Vous prendrez garde de choisir un mode de lecture/écriture adéquate (i.e. texte ou binaire) pour les fichiers.

N.B. 2 : Vous pourrez également réfléchir sur l'opportunité d'inclure une partie "entête" dans le fichier de sauvegarde avant la partie "données".

N.B. 3 : Vous pourrez modifier le prototype des fonctions (void -> int) afin d'inclure la gestion de code(s) d'erreur.

Programmation d'algorithmes de tri de données

Le problème de tri et de recherche de valeurs dans une liste constitue un problème classique et utile de l'algorithmique. Dans ce qui suit, on considère un ensemble d'éléments aléatoires de type entier et on cherche à les ordonner dans l'ordre croissant par différentes méthodes basiques.

36 – Tri par sélection

Principe de la méthode :

Il s'agit d'un algorithme de tri par comparaison. Sur un tableau de n éléments (indice 0 à n-1), le principe du tri par sélection est le suivant :

- rechercher le plus petit élément du tableau, et l'échanger avec l'élément d'indice 0 ;
- rechercher le second plus petit élément du tableau, et l'échanger avec l'élément d'indice 1 ;
- continuer de cette façon jusqu'à ce que le tableau soit entièrement trié.

Si on suppose que les m premiers éléments (0 à m-1) sont triés, il faut chercher la position du plus petit élément parmi les éléments m à n-1. On le permute avec l'élément m. L'ensemble se trouve donc trié de l'indice 0 à l'indice m.

Illustration :

En gras, les éléments déjà triés et en italique, les éléments à permuter.

18	10	3	25	9	*2*
2	*10*	*3*	25	9	18
2	**3**	*10*	25	*9*	18
2	**3**	**9**	*25*	*10*	18
2	**3**	**9**	**10**	*25*	*18*
2	**3**	**9**	**10**	**18**	**25**

Analyse :

Cet algorithme est particulièrement simple, mais inefficace sur de grandes entrées, car il s'exécute en temps quadratique en le nombre d'éléments à trier. Dans tous les cas, pour trier n éléments, le tri par sélection effectue n(n-1)/2 comparaisons. Sa complexité est donc $\Theta(n^2)$ (où n est la taille du tableau).

De ce point de vue, il est inefficace puisque les meilleurs algorithmes s'exécutent en temps $\Theta(n \ln(n))$ mais le tri par sélection n'effectue que peu d'échanges :

- n-1 échanges dans le pire cas, qui est atteint par exemple lorsqu'on trie la séquence 2,3,…,n,1 ;
- $n-(1/2+\cdots+1/n)\approx n-\ln(n)$ en moyenne c'est-à-dire si les éléments sont deux à deux distincts et que toutes leurs permutations sont équiprobables ;
- aucun si l'entrée est déjà triée.

Travail à effectuer :

Reprendre l'exercice n°19 et le modifier de manière à ce que le tableau soit initialisé par des valeurs aléatoires comprises entre 1 et 100 (fonction standard "**rand()**" déclarée dans **stdlib.h**).

Reprendre le pseudo-code du cours d'algorithmique puis écrire en langage C la fonction correspondante d'identificateur "**tri_selection**" qui ordonne dans l'ordre croissant les éléments du tableau par cette méthode. La tester ensuite.

37 – Tri par insertion

Principe de la méthode :

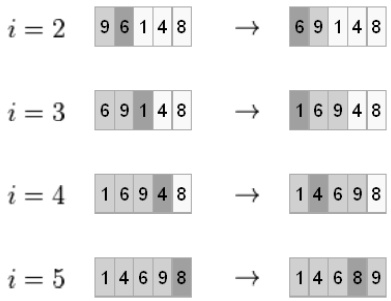
Le tri par insertion est un algorithme de tri classique, que la plupart des personnes utilisent naturellement pour trier des cartes : prendre les cartes mélangées une à une sur la table, et former une main en insérant chaque carte à sa place.

On prend 2 éléments et on les met dans l'ordre. On prend un troisième élément qu'on insère dans les 2 éléments déjà triés, etc. Un élément m va être inséré dans l'ensemble déjà trié des éléments 0 à m-1. Ce qui donnera m+1 éléments triés (0 à m).

L'insertion consiste à chercher l'élément de valeur immédiatement supérieure ou égale à celle de l'élément à insérer. Soit k l'indice de cet élément, on décale les éléments k à m-1 vers k+1 à m et l'on place l'élément à insérer en position k.

Illustration :

Voici les étapes de l'exécution du tri par insertion sur le tableau T = {9, 6, 1, 4, 8}. Le tableau est représenté au début et à la fin de chaque itération.



Analyse :

Dans le pire des cas, le nombre de comparaisons et de $n^2/2$. Dans le meilleur des cas, il est de n. L'algorithme est de complexité $\Theta(n^2)$ mais il est plus efficace que les deux précédents si le tableau est en partie trié.

Travail à effectuer :

Reprendre le pseudo-code du cours d'algorithmique puis compléter le programme précédent et le modifier de manière à introduire la fonction d'identificateur "**tri_a_insertion**" qui ordonne dans l'ordre croissant les éléments du tableau par cette méthode. La tester ensuite.

38 – Tri à bulles

Principe de la méthode :

Le **tri à bulles** ou **tri par propagation** est un algorithme de tri qui consiste à faire remonter progressivement les plus grands éléments d'un tableau, comme les bulles d'air remontent à la surface d'un liquide. L'algorithme parcourt le tableau et compare les couples d'éléments successifs. Lorsque deux éléments successifs ne sont pas dans l'ordre croissant, ils sont échangés. Après chaque parcours complet du tableau, l'algorithme recommence l'opération. Lorsqu'aucun échange n'a lieu pendant un parcours, cela signifie que le tableau est trié. On arrête alors l'algorithme.

Illustration :

Prenons la liste de chiffres {5 1 4 2 8} et trions-la de manière croissante en utilisant l'algorithme de tri à bulles. Pour chaque étape, les éléments comparés sont écrits en gras.

Première étape:
(**5 1** 4 2 8) → (**1 5** 4 2 8), les éléments 5 et 1 sont comparés, et comme 5 > 1, l'algorithme les intervertit.
(1 **5 4** 2 8) → (1 **4 5** 2 8) Interversion car 5 > 4.
(1 4 **5 2** 8) → (1 4 **2 5** 8) Interversion car 5 > 2.
(1 4 2 **5 8**) → (1 4 2 **5 8**) Comme 5 < 8, les éléments ne sont pas échangés. Les éléments les plus grands se déplacent ainsi comme des bulles vers la droite du tableau.

Deuxième étape:
(**1 4** 2 5 8) → (**1 4** 2 5 8) Même principe qu'à l'étape 1.
(1 **4 2** 5 8) → (1 **2 4** 5 8)
(1 2 **4 5** 8) → (1 2 **4 5** 8)

À ce stade, la liste est triée, mais pour le détecter, l'algorithme doit effectuer un dernier parcours.

Troisième étape:
(**1 2** 4 5 8) → (**1 2** 4 5 8)
(1 **2 4** 5 8) → (1 **2 4** 5 8)

Comme la liste est triée, aucune interversion n'a lieu à cette étape, ce qui provoque l'arrêt de l'algorithme.

Analyse :

Le principe du tri à bulles est simple mais sa complexité est de l'ordre de n^2 en moyenne (où n est la taille du tableau). Dans le pire des cas, le nombre de comparaisons et le nombre de permutations à effectuer sont de $n^2/2$. Dans le meilleur des cas (ensemble déjà trié), le nombre de comparaisons est de n - 1 et l'algorithme est donc de complexité linéaire.

Travail à effectuer :

Ecrire le pseudo-code de cet algorithme. Compléter le programme précédent et le modifier de manière à introduire la fonction d'identificateur "**tri_a_bulles**" correspondant à cet algorithme. La tester ensuite.

Approfondissement

Les exercices suivants doivent être traités à condition que tous les programmes précédents ont été développés.

39 – Arbre binaire

Rependre le programme de l'exercice 32 et le compléter de la manière suivante :

1 - Ecrire deux fonctions **Arbre FilsGauche (Arbre a)** et **Arbre FilsDroit (Arbre a)** qui retourne respectivement les fils gauche et droit d'un arbre non vide.

2- Ecrire une fonction **TypeDonnee Valeur (Arbre a)** qui retourne la valeur de la racine d'un arbre non vide.

3- Ecrire une fonction : **void ChangeValeur (TypeDonnee val, Arbre a)** qui remplace la valeur de la racine de l'arbre « **a** » par val si « **a** » est non vide, et qui ne fait rien dans le cas contraire.

4 - Ecrire deux fonctions **void ChangeFG (arbre a, arbre fg)** et **void ChangeFD (arbre a, arbre fd)** qui remplacent le fils gauche (resp. le fils droit) de l'arbre « **a** » par « **fg** » (resp. « **fd** ») si « **a** » est non vide, et ne font rien dans le cas contraire.

Attention : Penser à libérer la mémoire lorsque cela est opportun ...

40 – Programmation d'algorithmes

L'objectif est de programmer les algorithmes suivants qui ont été traités dans le cours d'algorithmique :

- Jeu de NIM
- Jeu du crêpier : problème consistant à mettre dans un ordre croissant une pile de crêpes de différentes tailles (initialement en désordre) en retournant uniquement les crêpes avec une spatule.
- Jeu de MARIENBAD
- Coefficients binomiaux - Triangle de Pascal
- Tours de Hanoi
- Conversion d'un nombre entier en base b
- Se sortir d'un labyrinthe la nuit : algorithme de PLEDGE
- Carré magique

Vous programmez le (ou les algorithmes) que vous souhaitez. Cependant, les programmes développés doivent répondre aux consignes indiquées page 1. La notation tient compte de l'ensemble des points explicités.