UFR d'Ingénierie

13 janvier 2020

L1 (2019-2020)

Éléments de programmation en C (LU1IN002)

TD

Semaines 1 à 6

Semaine 1 - TD

Objectifs

- Fonctions
- Fonction d'affichage (fonction `printf`)
- Alternatives
- Directive **`#define`**
- Assert

Exercices

Exercice 1 – Premier programme C

Soient la fonction et le jeu de tests suivants écrits en Python.

```
def surface(l,L):
    """int * int -> int
    hypothèse : (L >= l) et (l >= 0)

    renvoie la surface du rectangle défini par sa largeur l et sa longueur L."""

    return (l*L)

# jeu de tests :
assert surface(0,0) == 0
assert surface(0,20) == 0
assert surface(1,1) == 1
assert surface(5,10) == 50
```

Question 1

Écrivez la fonction `surface` en C.

En C, la fonction `assert` prend en paramètre une expression, si cette dernière est vraie, le programme poursuit son exécution, sinon il s'arrête en affichant un message indiquant l'expression qui n'est pas vérifiée. Pour utiliser cette fonction, il est nécessaire d'utiliser la directive **`#include`** `<assert.h>`.

Considérons le programme suivant (écrit dans le fichier `exemple_assert.c`) :

```
#include <assert.h>

int addition(int a, int b){
/*  renvoie la somme a+b */

  return a + b;
}

int main() {
  assert(addition(1,1) == 2);
  assert(addition(-1,1) == 1);
  assert(addition(20,0) == 20);

  return 0;
}
```

Lors de l'exécution, le programme s'arrête sur le deuxième `assert` car l'expression passée en paramètre est fausse. Les instructions qui suivent un `assert` non vérifié ne sont pas exécutées. Le message suivant est affiché :

```
Assertion failed: (addition(-1,1)== 1), function main, file exemple_assert.c, line 12.
Abort
```

Question 2

Écrivez le programme C complet permettant de tester la fonction `surface` de la question précédente en utilisant la fonction `assert`.

Exercice 2 – Fonction d'affichage

Question 1

Écrivez une fonction `multi` qui prend en paramètres un entier et un flottant et qui renvoie le résultat de la multiplication des deux valeurs.

La fonction `printf` permet d'afficher du texte à l'écran. Pour pouvoir l'utiliser il faut inclure la directive **`#include`** `<stdio.h>` en début de programme.

La fonction `printf` prend en paramètre une chaîne de caractères. L'exécution de la commande `printf("Ceci est du texte.\n");` affichera le texte `Ceci est du texte.` suivi d'un passage à la ligne représenté par le caractère `\n`.

Pour afficher la valeur de variables (ou la valeur de retour d'une fonction), il faut préciser dans le texte le format de la valeur que l'on veut afficher et faire suivre le chaîne de caractères de la liste des variables (ou appels de fonctions) dont on veut afficher la valeur (dans l'ordre d'apparition des formats).

- `%d` correspont à une valeur de type entier.
- `%f` correspont à une valeur de type flottant (affiche 6 chiffres après la virgule). La valeur affichée peut-être une valeur arrondie.
- `%c` correspont à une valeur de type caractère.
- `%s` correspont à une valeur de type chaîne de caractères.

Considérons le programme suivant :

```
#include <stdio.h>

int main() {
int i=2;
float a=1.27;
char k='h';

printf("entier : %d, flottant : %.10f, caractere : %c\n",i,a,k);
return 0;
}
```

Son exécution donnera l'affichage de la ligne :

```
entier : 2, flottant : 1.270000, caractere : h
```

Pour contrôler le nombre de chiffres affichés après la virgule lors de l'affichage d'un flottant, il suffit d'utiliser le format `%.xf` où `x` correspond au nombre de chiffres après la virgule souhaité.

Dans le programme précédent si on remplace le `%f` par `%.2f` on obtient l'affichage suivant :

```
entier : 2, flottant : 1.27, caractere : h
```

Si, dans le même programme, on remplace le `%f` par `%.1f` on obtient l'affichage suivant :

```
entier : 2, flottant : 1.3, caractere : h
```

Question 2

Nous souhaitons maintenant compléter la fonction `main` suivante pour qu'elle affiche le résultat de la multiplication de `op1` et `op2` sous la forme `3 * 2.70 = 8.10`. Votre code doit bien sûr être indépendant des valeurs des variables `op1` et `op2` et doit faire appel à la fonction `multi`.

```
int main() {
  int op1;
  float op2;

  op1=3;
  op2=2.7;

  /* instruction d'affichage a ajouter */

  return 0;
}
```

Exercice 3 – Le plus grand des trois

Question 1

Écrivez une fonction `plusGrand` qui prend en paramètre trois entiers et qui renvoie le plus grand des trois.

Question 2

Écrivez la fonction `main` permettant de tester le fonction `plusGrand`.

Question 3

Donnez une instruction, faisant appel à la fonction `plusGrand`, permettant d'afficher l'entier le plus grand parmi cinq.

Exercice 4 – Signe d'une somme

Question 1

Écrivez la fonction `signeSomme` qui prend en paramètre deux entiers et qui, sans calculer la somme, renvoie 0 si la somme est nulle, -1 si elle est négative et 1 sinon.

Question 2

Écrivez la fonction `main` qui permet de tester la fonction `signeSomme` en utilisant la fonction `assert`.

Exercice 5 – Prix d'une place de cinéma

Un cinéma pratique les tarifs suivants (tarifs en juin 2018) :

- tarif normal : 11,40 €
- séance débutant avant 11 heures (et après 8 heures) : 7,10 €
- moins de 14 ans : 4,5 €
- moins de 26 ans du lundi au vendredi : 4,90 €
- moins de 26 ans samedi et dimanche : 7,90 €.

Question 1

La directive **#define** permet d'associer une valeur à une chaîne de caractères. Lors de la compilation, chaque occurrence de la chaîne sera remplacée par la valeur.

La directive **#define** `TAILLE 3` associe la valeur 3 à la chaîne de de caractères `TAILLE`. Chaque apparition du mot `TAILLE` sera remplacée par la valeur 3. L'instruction `x=TAILLE;` revient donc à donner la valeur 3 à la variable `x`.

De cette façon, il est possible de nommer des « constantes » dont la valeur peut être modifiée en une seule écriture (dans la directive **#define**). Le code des fonctions sera donc indépendant de la valeur des constantes.

Nous utilisons cette directive pour définir les tarifs :

```
#define TNORMAL 11.4
#define TMOINS14 4.5
#define TMOINS26S 4.9
#define TMOINS26WE 7.90
#define TMATIN 7.10
```

Écrivez une fonction `prixPlace` qui :

- prend trois paramètres, deux entiers représentant un age et un jour (lundi=1, ..., dimanche=7) et un flottant correspondant à l'heure de début de la séance,
- renvoie le tarif appliqué.

Vous devez bien sûr calculer le tarif le plus avantageux pour le spectateur.

Question 2

Écrivez la fonction `main` permettant de tester la fonction `prixPlace`. Étant donné que cette fonction renvoie un flottant, il n'est pas possible de tester directement cette valeur. Nous souhaitons que la fonction `main` affiche les valeurs calculées lors du jeu de tests.

Semaine 2 - TD

Objectifs

— Boucles
— Pile
— Comparaison de réels

Exercices

Exercice 6 – Boucles en C

Soit la fonction et le jeu de tests suivants écrits en Python.

```
def somme_carres(m,n):
   """int * int -> int
   hypothèse : m <= n

   renvoie la somme des carrés des entiers dans l'intervalle [m;n]."""

   # s : int
   s = 0   # la somme des carrés

   # i : int
   i = m   # entier courant de l'intervalle

   while i <= n:
      s = s + i * i
      i = i + 1
   return s

# jeu de tests :
assert somme_carres(1,5) == 55
assert somme_carres(2,5) == 54
assert somme_carres(3,5) == 50
assert somme_carres(4,5) == 41
assert somme_carres(5,5) == 25
assert somme_carres(3,6) == 86
assert somme_carres(-4,0) == 30
assert somme_carres(0,4) == 30
```

Question 1

Écrivez, en C, une fonction `somme_carres` qui a le même comportement que la fonction Python et qui est écrite avec une boucle **`while`**.

Question 2

Écrivez la fonction `main` permettant de tester la fonction de la question précédente en utilisant la fonction `assert`.

Question 3

Nous considérons maintenant la même fonction mais écrite avec une boucle **`for`**.

```
def somme_carres(m,n):
   """int * int -> int
   hypothèse : m <= n

   renvoie la somme des carrés des entiers dans l'intervalle [m;n]."""

   # s : int
   s = 0   # la somme des carrés

   # i : int
   # entier courant de l'intervalle

   for i in range(m,n+1) :
      s = s + i * i
   return s
```

De la même façon, écrivez une fonction C équivalente en utilisant une boucle **for**.

Exercice 7 – Nombres premiers

Nous souhaitons afficher les nombres premiers inférieurs ou égaux à un entier donné (MAX).

Nous considérons le programme ayant la structure suivante dans lequel la primitive **#define** définit l'entier maximum :

```
#include <stdio.h>

#define MAX 5

... premier(...) {
  ...
}

... listeNombresPremiers(...) {
  ...
}

int main(){

  printf("liste des nombres premiers <= %d\n",...);
  ...
  return 0;
}
```

Question 1

La fonction `premier` prend en paramètre un entier (entier naturel par hypothèse) et renvoie 1 s'il est premier, 0 sinon. La fonction `listeNombresPremiers` prend en paramètre un entier et affiche les nombres premiers inférieurs ou égaux au paramètre.

Donnez la signature des fonctions `premier` et `listeNombresPremiers` et complétez la fonction `main`. Dans cette question il ne vous est pas demandé d'écrire les fonctions `premier` et `listeNombresPremiers`

Question 2

Comment faire pour tester le programme sur une autre valeur que 5 ?

Question 3

Écrivez une fonction `premier` qui prend en paramètre un entier (entier naturel par hypothèse) et qui renvoie 1 s'il

est premier, 0 sinon. Nous vous rappelons qu'un nombre premier est un entier naturel qui admet exactement deux diviseurs positifs, 1 et lui-même, 1 n'est donc pas premier. Nous pouvons déterminer qu'un nombre n'est pas premier dès qu'on a trouvé un de ses diviseurs (autre que 1 et lui-même).

Question 4

Écrivez une fonction `listeNombresPremiers` qui prend en paramètre un entier `n_max` et qui affiche les nombres premiers inférieurs ou égaux à `n_max`.

Question 5

Donnez les évolutions de la pile d'exécution lors de l'exécution du programme lorsque la valeur associée à `MAX` est 3.

Exercice 8 – Décomposition d'une somme en euros

Question 1

Écrivez une fonction `decomposition_somme` qui prend une somme entière en paramètre et qui affiche la décomposition de cette somme en un nombre minimal de billets-pièces de 5, 2 et 1 €.

Indice : il faut considérer le plus possible de billets de 5 €, puis le plus de pièces de 2 € et compléter par les pièces de 1 €. N'oubliez pas que la division entre deux entiers donne un résultat entier et que l'opérateur `%` permet d'obtenir le reste de cette division entière (`13/5 = 2` et `13%5 = 3`).

Question 2

Écrivez une fonction `decomposition_multiple_somme` qui prend une somme entière en paramètre et qui affiche toutes les décompositions possibles de cette somme en billets-pièces de 5, 2 et 1 €.

Exercice 9 – Nombres parfaits

Question 1

Ecrivez une fonction `sommeDiviseurs` qui prend en paramètre un entier naturel et qui renvoie la somme des ses diviseurs stricts.

Question 2

Un nombre est k-parfait si la somme de ses diviseurs est égale à k fois le nombre (vous remarquerez que cette fois on ne se limite pas aux diviseurs stricts). Ecrivez une fonction `k_parfait` qui prend deux entiers naturels en paramètre, `n` et `k`, et qui renvoie 1 si `n` est k-parfait, 0 sinon. Vous devez bien-sûr utiliser la fonction `sommeDiviseurs`.

Question 3

Nous souhaitons maintenant trouver, pour un entier donné, la plus petite valeur de k ($k \in [k_{min}, k_{max}]$) pour laquelle l'entier est k parfait. Les valeurs de k_{min} et k_{max} seront définies par des primitives **`#define`**. Ecrivez la fonction `trouver_k_parfait` qui prend en paramètre un entier naturel `n` et qui renvoie la plus petite valeur de k telle que `n` est k-parfait. Si on ne trouve pas de valeur, la fonction renvoie -1.

Exercice 10 – Comparaison de valeurs réelles

Question 1

Nous considérons une fonction `surface_float`, similaire à la fonction `surface` écrite la semaine dernière, dont les paramètres et la valeur de retour sont de type **`float`**. Pourquoi est-il fortement déconseillé d'utiliser la fonction `assert` pour tester directement la fonction `surface_float` ?

Pour tester l'égalité entre flottants, nous devons déterminer si les deux valeurs son égales à *epsilon près* de façon similaire à ce que vous avez fait en Python au premier semestre.

Question 2

Écrivez une fonction `valeur_absolue` qui prend un paramètre de type **`float`** et renvoie sa valeur absolue.

Question 3

Écrivez une fonction `egal_eps` qui prend en parmètres trois valeurs de type **`float`**, deux valeurs à comparer et *epsilon* et qui renvoie 1 si les deux valeurs sont égales à *epsilon près*, 0 sinon.

Question 4

Écrivez maintenant une fonction `main` permettant de tester la fonction `surface_float` en utilisant la fonction `assert`.

Semaine 3 - TD

Objectifs

— Pile
— Adresse
— Pointeur

Exercices

Exercice 11 – Appels de fonctions et pile

Question 1
Dites ce qu'affiche le programme suivant et donnez l'évolution de la pile d'exécution.

```
#include <stdio.h>

int diff(int a, int b) {
  int x = a -b;
  return x;
}

int calcul(int a, int b) {
   int x;

   if (a > b) {
      x = diff(a,b);
   }
   else {
      x = diff(b,a);
   }
   return x;
}

int main() {
   int res;

   res = calcul(7,2);
   printf("Le premier resultat du calcul est %d.\n",res);
   res = calcul(-15,3);
   printf("Le deuxieme resultat du calcul est %d.\n",res);
   return 0;
}
```

Exercice 12 – Qu'est-ce qu'un pointeur ?

Soit le programme suivant.

Question 1
Donnez l'évolution de la pile d'exécution lors de l'exécution du programme suivant. Déduisez-en l'affichage obtenu.

```
#include <stdio.h>

void ma_fonction (int a, int b) {
  int c1,c2;
  int *d;

  c1 = a + b;
  d = &a;
  *d = *d + 2;
  c2 = a + b;

  printf("a = %d, b = %d, c1 = %d, c2= %d, *d = %d\n",a,b,c1,c2,*d);
}

int main() {
  int a=7, b =10;

  printf("Avant appel : a=%d, b= %d\n",a,b);
  ma_fonction(a,b);
  printf("Apres appel : a=%d, b= %d\n",a,b);
  return 0;
}
```

Exercice 13 – Pointeur et paramètres

Soit le programme suivant :

```
#include <stdio.h>

void ma_fonction(int *param1, int param2) {
   int var_loc = 3;

   *param1 = var_loc * param2;
   param2 = var_loc + 1;
}

int main() {
   int v1, v2;

   v1 = 10;
   v2 = 3;
   ma_fonction(&v2,v1);
   return 0;
}
```

Question 1

Représentez l'évolution de la mémoire (pile d'exécution) lors de l'exécution du programme précédent.

Exercice 14 – Fonction mettant à jour plusieurs informations

Dans cet exercice nous supposons que nous avons à notre disposition la fonction `int valeur_aleatoire(int min, int max)` qui retourne un entier choisi aléatoirement entre les valeurs `min` et `max` comprises. Vous verrez en TP comment écrire cette fonction.

Question 1

Dites ce que fait le programme suivant :

```
#include <stdio.h>

#define NB_VALEURS 1000
#define VMIN 0
#define VMAX 32000

int valeur_aleatoire(int min, int max) {
  /* renvoie une valeur choisie aleatoirement entre min et max */
  ....
}

int minimum(int val, int valMin) {
  /* renvoie la plus petite valeur entre val et valMin */

  if (val < valMin) {
    return val;
  }
  return valMin;
}

int maximum(int val, int valMax) {
  /* renvoie la plus grande valeur entre val et valMax */

  if (val > valMax) {
    return val;
  }
  return valMax;
}

int main(){
  int i,val;
  int min=VMAX, max=VMIN;

  for (i=0; i < NB_VALEURS; i++) {
    val=valeur_aleatoire(VMIN,VMAX);
    min=minimum(val,min);
    max=maximum(val,max);
  }

  printf("MIN = %d, MAX = %d\n", min,max);
  return 0;
}
```

Question 2

Écrivez une fonction `minimum_maximum` qui permet en un seul appel de mettre à jour le minimum et le maximum. Voici la fonction `main` dans laquelle vous devez aussi compléter l'appel à `minimum_maximum` :

```
int main(){
  int i,val;
  int min=VMAX, max=VMIN;

  for (i=0; i < NB_VALEURS; i++) {
    val=valeur_aleatoire(VMIN,VMAX);
    minimum_maximum(........);
  }

  printf("MIN = %d, MAX = %d\n", min,max);
```

```
  return 0;
}
```

Exercice 15 – Permutation circulaire

Question 1

Ecrivez la fonction `permute` qui permet de permuter la valeur de deux variables. Si initialemenent `a=10` et `b=-20`, nous souhaitons que la valeur de ces variables après appel à la fonction `permute` soit `a=-20` et `b=10`.

Question 2

Écrivez la fonction `permute_circulaire` qui permet de faire une permutation circulaire entre les valeurs de trois variables. Votre fonction doit faire appel à la fonction `permute`.

Si initialemenent `a=10`, `b=20` et `c=30`, nous souhaitons que la valeur de ces variables après appel à la fonction `permute_circulaire` soit `a=20`, `b=30` et `c=10`.

Question 3

Soit la fonction `main` suivante. Complétez l'appel à la fonction `permute_circulaire` et donnez l'évolution de la pile d'exécution lors de l'exécution du programme.

```
int main(){
  int a, b, c;

  a=10; b=20; c=30;
  permute_circulaire(.....);
  return 0;
}
```

Semaine 4 - TD

Objectifs

— Tableaux : opérations de base
— Tableau en valeur de retour

Exercices

Exercice 16 – Affichage d'un tableau, échange du contenu de deux cases

Question 1
Écrivez une fonction qui affiche le contenu d'un tableau de flottants.

Question 2
Modifiez la fonction pour qu'elle affiche le contenu du tableau à raison de p éléments par ligne.

Question 3
Écrivez une fonction qui échange le contenu de la case d'indice i d'un tableau de flottants avec le contenu de la case d'indice j. Les valeurs de i et j sont supposées correctes (elles correspondent bien à des indices du tableau).

Écrivez une fonction `main` permettant de tester votre fonction.

Exercice 17 – Un tableau est-il un pointeur ?

Question 1
Représentez l'état de la mémoire à la fin de l'exécution du programme ci-dessous :

```
#include <stdio.h>
#include <stdlib.h>
#define N 4

int main() {
    int i;
    int tab[N];
    for (i = 0; i < N; i++) {
        tab[i] = 2 * i + 1;
        printf("%d\t", tab[i]);
    }
    printf("\n");
    return 0;
}
```

Question 2
On modifie maintenant la déclaration du tableau : `int tab[N];` est remplacé par :

```
    int *tab;
    tab = malloc(N * sizeof(int));
```

Le programme est-il toujours correct ? Si oui, représentez l'état de la mémoire à la fin de l'exécution du programme, sinon expliquez pourquoi.

Question 3

Le programme suivant est-il correct ? Si oui, qu'affiche-t-il ? Sinon, pourquoi ?

```
#include <stdio.h>
#define N 4

int main() {
    int i;
    int tab[N] = {1, 3, 5, 7};
    int *tab2;

    tab2 = tab;
    tab2[1] = 2;
    for (i = 0; i < N; i++) {
        printf("%d\t", tab[i]);
    }
    printf("\n");
    return 0;
}
```

Exercice 18 – Initialisation d'un tableau

Question 1

Écrivez un programme qui déclare un tableau statique de N entiers, l'initialise avec les valeurs de 1 à N et l'affiche.

Nous souhaitons maintenant pouvoir initialiser le tableau avec des valeurs tirées aléatoirement.

Question 2

Écrivez une fonction `init_alea` qui initialise un tableau d'entiers avec des valeurs tirées aléatoirement entre deux entiers `MIN` et `MAX`. Complétez le programme précédent pour tester la fonction.

Exercice 19 – Tableau en valeur de retour

Question 1

Quel est le résultat de l'exécution de ce programme ?

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

int *tirage(int max, int chance_max) {
/* renvoie le tirage de 5 numeros + numero chance */
   int i;
   int res[6];

   for (i = 0; i < 5; i++) {
      res[i] = 1 + (rand() % max);
   }
   res[5] = 1 + (rand() % chance_max);
   return res;
```

```
}

int main() {
   int i;
   int *loto;

   srand(time(NULL));
   loto = tirage(49, 10);
   printf("Tirage du jour : ");
   for (i = 0; i < 5; i++) {
      printf("%d\t", loto[i]);
   }
   printf("\nNumero chance : %d\n", loto[5]);
   return 0;
}
```

Semaine 5 - TD

Objectifs

— Chaînes de caractères
— Tableau à deux dimensions

Exercices

Exercice 20 – Comptage du nombre d'occurrences d'une valeur dans une chaîne

Nous considérons une chaîne de caractères dans laquelle certains caractères peuvent apparaître plusieurs fois. Nous vous rappelons qu'une chaîne de caractères est un tableau de caractères dont le dernier caractère a la valeur '`\0`'.

Question 1
Écrivez une fonction `compte` qui compte le nombre d'occurrences d'un caractère `cr` dans une chaîne. Écrivez une fonction `affiche_occur` qui utilise la fonction `compte` pour afficher le nombre d'occurrences de chaque caractère d'une chaîne.
Écrivez une fonction `main` permettant de tester vos fonctions.

À moins d'avoir été particulièrment astucieux dans l'écriture de la fonction `affiche_occur`, lorsqu'un caractère est présent plusieurs fois dans le tableau, son nombre d'occurrences est affiché à chaque fois.

Question 2
Modifiez la fonction `affiche_occur` pour n'afficher qu'une fois le nombre d'occurrences de chaque caractère.

Question 3
Écrivez une fonction qui renvoie la position de la valeur la plus fréquente dans le tableau. Complétez la fonction `main` pour tester votre fonction.

Question 4
Proposez une version optimisée de la fonction en utilisant le même mécanisme que pour afficher une seule fois chaque valeur.

Exercice 21 – Distances entre villes

Nous considérons un ensemble de `NB_VILLES` préfectures représentées par le code postal de leur département. Par exemple, le nombre `29`, code postal du Finistère, représente la ville de Brest. Les distances séparant ces villes sont regroupées dans un tableau dont la première colonne code la ville.

Nous supposons que nous disposons, dans la fonction `main`, d'un tableau donnant les distances entre Brest (`29`), Lille (`59`), Strasbourg (`67`), Paris (`75`) et Toulon (`83`).

L'ordre des colonnes étant identique à celui des lignes, les numéros des villes ne sont pas répétés en première ligne du tableau. On voit ici que la distance entre Brest et Lille est de `598` kilomètres.

La directive **`#define`** `NB_VILLES 5` fixe le nombre de villes.

```
int distances[NB_VILLES] [NB_VILLES + 1] = {
                 {29, 0, 598, 900, 504, 995},
                 {59, 598, 0, 407, 203, 861},
                 {67, 900, 407, 0, 397, 621},
                 {75, 504, 203, 397, 0, 694},
                 {83, 995, 861, 621, 694, 0},
};
```

Question 1

Écrivez une fonction `void affiche_distances(int dist[NB_VILLES][NB_VILLES + 1])` qui affiche les distances entre les villes sous la forme suivante (vous noterez l'ajout de la première ligne de l'affichage et la représentation de la valeur non significative `0`) :

```
km      29      59      67      75      83
29      -       598     900     504     995
59      598     -       407     203     861
67      900     407     -       397     621
75      504     203     397     -       694
83      995     861     621     694     -
```

Question 2

Écrivez une fonction `int plus_proche (int ville, int dist[NB_VILLES][NB_VILLES + 1])` qui utilise le tableau `distances` pour calculer la ville la plus proche de celle dont le code correspond au paramètre `ville`.

La fonction renvoie l'indice de ligne correspondant à la ville recherchée. Par exemple, si `ville` vaut `29`, la fonction renvoie `3` qui est l'indice de la ligne correspondant à Paris (distance : `504`). Si `ville` ne correspond pas à une ville du tableau, la fonction retourne `-1`.

Exercice 22 – Chaîne en valeur de retour

Nous voulons écrire une fonction `int_to_str` qui prend en paramètre un entier et qui construit et renvoie la chaîne de caractères représentant cet entier.

Question 1

Donnez le prototype de la fonction.

Puisque le contenu de la pile est perdu à la fin de la fonction, nous allons allouer dynamiquement (donc dans le tas) la zone mémoire destinée à stocker la chaîne de caractères.

La fonction doit donc commencer par compter le nombre de chiffres qui composent le nombre à transformer pour connaître la taille de la zone à allouer. Puis, en commençant par les unités, transformer chaque chiffre en caractère avant de le copier dans la chaîne. La règle qui permet de transformer un chiffre en caractère est simple : pour obtenir le code ASCII du caractère représentant un chiffre, il suffit d'ajouter `48` (ou `0x30`) à la valeur du chiffre. Par exemple, le code ASCII de `'3'` est `51` (ou `0x33`).

Question 2

Écrivez le corps de la fonction, ainsi qu'un programme permettant de la tester.

Exercice 23 – Minimum conditionnel d'un tableau

Question 1

Écrivez une fonction qui calcule et renvoie la position du minimum dans un tableau d'entiers `tab` contenant `taille` éléments. La fonction respectera le prototype suivant :

```
int indice_min(int tab[], int taille);
```

Le premier paramètre de la fonction est le tableau et le second la taille de celui-ci. La valeur renvoyée est la position dans le tableau (indice) du minimum trouvé par la fonction.

Question 2

Peut-on, en utilisant la fonction précédente, afficher la plus petite valeur contenue dans le tableau ?

Question 3

Écrivez une fonction qui renvoie la position de la plus petite valeur *positive ou nulle* contenue dans un tableau d'entiers quelconques. La fonction renvoie -1 si aucune valeur satisfaisant la condition n'est trouvée.

Écrivez une fonction `main` permettant de tester votre fonction.

Semaine 6 - TD

Objectifs

— Arithmétique des pointeurs
— Récursivité sur les tableaux

Exercices

Exercice 24 – Comparaison de chaînes de caractères

Question 1
Écrivez une fonction *itérative* qui prend deux chaînes de caractères en paramètre. La fonction renvoie 1 si les chaînes sont identiques, 0 sinon.

Écrivez une fonction `main` permettant de tester votre fonction.

Nous souhaitons maintenant effectuer la comparaison au moyen d'une fonction récursive.

Question 2
Quels sont les cas d'arrêt ?

Question 3
Quelle expression représente l'adresse en mémoire de la case d'indice `i` du tableau ? Comment peut-on accéder au contenu de cette case sans utiliser la notation `tab[i]` ?

Question 4
Écrivez une version *récursive* de la fonction de comparaison.

Exercice 25 – Recherche dichotomique d'un élément dans un tableau trié

La recherche dichotomique consiste à déterminer si un élément appartient à un tableau trié en divisant par deux l'intervalle de recherche à chaque itération. Les opérations à effectuer lors d'une itération sont donc :

- déterminer l'indice représentant le milieu de l'intervalle courant ; si élément à cette position est l'élément recherché, l'algorithme se termine avec une réponse positive ;
- sinon
  - si l'élément du milieu est supérieur à l'élément recherché, poursuivre la recherche dans l'intervalle à gauche de l'élément du milieu ;
  - si l'élément du milieu est inférieur à l'élément recherché, poursuivre la recherche dans l'intervalle à droite de l'élément du milieu ;

Nous voulons dans un premier temps réaliser cette recherche en utilisant une fonction *itérative*. Soient `g` et `d` les variables représentant les bornes gauche et droite respectivement de l'intervalle de recherche.

Question 1
Que valent :

- l'indice de l'élément au milieu de l'intervalle ?

- les bornes du nouvel intervalle de recherche si l'élément du milieu est supérieur à l'élément recherché ?
- les bornes du nouvel intervalle de recherche si l'élément du milieu est inférieur à l'élément recherché ?

Question 2
À quel moment peut-on déterminer que l'élément recherché ne figure pas dans le tableau ?

Question 3
Comment se traduit la condition précédente ?

Question 4
Écrivez une fonction *itérative* qui utilise le principe de la dichotomie pour tester la présence d'un élément dans un tableau. La fonction renvoie 1 si l'élément est présent, 0 sinon.

Écrivez une fonction `main` permettant de tester votre fonction.

Question 5
Écrivez une fonction *récursive* qui utilise le principe de la dichotomie pour tester la présence d'un élément dans un tableau. La fonction renvoie 1 si l'élément est présent, 0 sinon.

Question 6
Quelles sont les valeurs que vous choisiriez pour tester votre programme ?

Question 7
Lorsqu'on recherche un élément qui n'est pas présent dans le tableau, est-il plus intéressant d'utiliser une recherche dichotomique ou une recherche linéaire ?