Bases de la Programmation en C – Travaux dirigés

1 Expressions et représentation des nombres en C

Exercice 1. [Identificateurs] Indiquer si les identificateurs suivants sont valides :

foo-1	_foo_bar	3cavaliers	foo.	tête
___A_	a	_	goto	

Exercice 2. [Les expressions et leurs évaluations] Les principaux opérateurs et leurs priorités sont :

16	`()` `[]` `->` `.`	G
15	`++` `--` (postfixé)	D
14	`!` `~` `++` `--` (préfixé) `-` (unaire)	D
	`*` (indirection) `&` (adresse) `sizeof`	D
13	`*` (multiplication) `/` `%`	G
12	`+` `-`	G
11	`<<` `>>`	G
10	`<` `<=` `>` `>=`	G
9	`==` `!=`	G
8	`&` (et bit à bit)	G
7	`^`	G
6	`|`	G
5	`&&`	G
4	`||`	G
3	`?:`	D
2	`=` `+=` `-=` `*=` `/=` `%=` `>>=` `<<=` `&=` `^=` `|=`	D
1	`,`	G

En supposant que $A = 20 \quad B = 5 \quad C = -10 \quad D = 2 \quad X = 12 \quad Y = 15$, évaluer les expressions valides parmi les expressions suivantes:

$5 * X + 2 * 3 * B/4$	$A == B = 5$	$A+ = X + 2$	$A! = C* = -D$
$A\% = D + +$	$A\% = + + D$	$(X + +) * A + C$	$A = 6 * D == X?B : Y$
$!(X - D + C)\| \| + +D$	$A\&\&B\|\|!0\&\&C\&\&!D$	$C = 12 - -$	`(1<<(2<<1))==((1<<2)<<1)`

En utilisant le parenthésage, corriger les expressions invalides.

Exercice 3. [Quelques fonctions / programmes]

Quel est l'erreur dans ce code ?

```
int square(int x);
  { return x*x; }
```

Que se passe t-il si l'on appelle la fonction `show` avec le paramètre 4 ?

```
int show(int x){
  printf("%d %d\n", x, x*x);
  return x*x;
  printf("%d %d\n", x, x+x);
  return x+x;
}
```

Quelle est la valeur de retour des deux programmes suivants ?

```
int tmp(int x, int y){
  x = 2*x + y;
  return x;
}

int main(void){
  int x = 2, y = 5;
  y = tmp(y, x);
  x = tmp(y, x);
  return x*y;
}
```

```
int main(){
  int a=5;
  if (1 <= a <= 2)
    return 0;
  else
    return 1;
}
```

Exercice 4. [Représentations des entiers] Quel et le plus grand et le plus petit entier parmi `a`, `b` et `c` ?

```
int a = 132;
int b = 0x101;
int c = 0241;
```

Exercice 5. [Afficher un chiffre hexadécimal] Écrivez la fonction `int put_hdigit(int h);` qui affiche un nombre `h` sous la forme d'un caractère hexadécimal. `h` est donc une valeur entre 0 et 15 et le code ASCII produit est donc:

- entre `'0'` et `'9'` si $0 \leq h \leq 9$; ou
- entre `'A'` et `'F'` si $10 \leq h \leq 15$.

Votre fonction renverra -1 si h n'a pas une valeur correcte, et le code ASCII du caractère affiché sinon. La seule fonction autorisée est la fonction `putchar`.

Exercice 6. [Afficher un nombre décimal] Donnez la définition de la fonction `int putdec(int d);` qui affiche les chiffres de l'écriture décimale de la valeur `d`, au besoin, précédé d'un signe négatif, et sans afficher de `'0'` inutiles à gauche du nombre.
Cette fonction retourne normalement 0, -1 en cas d'erreur.

2 Préprocesseur et compilation séparée

Exercice 7. [Macros]

1. Définissez deux macros `TRUE` et `FALSE`
2. Définissez une macro `SQR` qui "calcule" le carré d'un élément. Explicitez son utilisation avec:

   ```
   int a=3;
   int b;
   b=SQR(a+1);
   ```

   Quid de l'utilisation suivante ?

   ```
   b=SQR(a++);
   ```

3. Écrivez une macro `CAPITAL` qui renvoie un booleen (vrai si la lettre est majuscule, faux sinon). Explicitez son utilisation sur ce code:

   ```
   while (CAPITAL(c=getchar())){
     putchar(c);
   }
   ```

Exercice 8. [Header file et gardes.] Vous avez construit lors du TP2 un fichier `numbers.c` contenant le code, entre autre, des fonctions `putdec`, `puthex` et `putbin`.

1. Écrire le fichier de prototype donnant la déclaration de ces 3 fonctions.
2. Comment s'assurer que ce fichier ne soit pas inclus plusieurs fois ?
3. Quelle ligne doit contenir notre fichier principal (où se trouvera notre fonction `main`, et utilisant certains des fonctions nommées ci-dessus) pour permettre une bonne compilation ?
4. Le fichier `numbers.c` contient d'autres fonctions (affichage de chiffres). Si nous ne souhaitons pas que ces fonctions additionnelles soient utilisés en dehors de notre fichier, comment s'assurer que ce ne soit pas le cas ?

Exercice 9. [Localité des variables.] Considérons les fichiers suivants:

main.c

```
extern int x;
static int y;

void proc2(void)
{
  printf("proc2: %d %d\n", x, y);
}

int main(void){
  choice(0);
  affectation();
  choice(1);
  proc1();
  proc2();
  return 0;
}
```

aux.c

```
int x;
static int y;
void affectation(void) {x=2; y=3;}

void choice(int a) {
  int x = 1;
  static int y=5;
  if (a == 0)
    y = 0;
  else y += x;
  printf("choice: %d %d ; ", x, y);
}

void proc1(void) {
  printf("proc1: %d %d ; ", x, y);
}
```

1. Que manque t-il dans le fichier `main.c` pour qu'il puisse compiler ?
2. Comment créer un exécutable à partir de ces fichiers en utilisant une *compilation séparée* ?
3. Écrire un Makefile minimal permettant d'assurer une telle compilation séparée.
4. Quel affichage ce code génère t-il ?

Exercice 10. [Les erreurs classiques de compilation et/ou d'éditions de liens] Dans cet exercice, vous avez à examiner plusieurs scenarios indépendants, dans lesquels on vous donne le contenu d'un Makefile, et d'un ou plusieurs fichiers source.

Scénario 1

Fichier `foo.h`

```
void foo();
```

Fichier `foo.c`

```
void foo(){
}
```

Fichier `main.c`

```
#include "foo.h"
int main(){
  foo();
}
```

Fichier `Makefile`

```
all: main

main: main.o
	gcc -o main main.o

main.o: main.c
	gcc -c main.c

foo.o: foo.c
	gcc -c foo.c
```

Scénario 2

Fichier `foo.h`

```
void foo() {
}
```

Fichier `foo.c`

```
#include "foo.h"
```

Fichier `main.c`

```
#include "foo.h"
int main(){
  foo();
}
```

Fichier `Makefile`

```
all: main

a.out: main.o
	gcc main.o foo.o

main.o: main.c
	gcc -c main.c

foo.o: foo.c
	gcc -c foo.c
```

Scénario 3

Fichier `foo.h`

```
void foo();
```

Fichier `foo.c`

```
void foo() {
}
```

Fichier `main.c`

```
int main() {
   foo();
}
```

Fichier `Makefile`

```
all: main

a.out: main.o
	gcc main.o foo.o

main.o: main.c
	gcc -c main.c

foo.o: foo.c
	gcc -c foo.c
```

Pour chaque scénario, vous devez réaliser les actions suivantes:

- Donnez la liste des unités de compilation, et pour chacune, donnez la liste des fichiers `.h` inclus.
- Pour chaque fonction, donnez le/les unité(s) de compilation dans laquelle elle est définie / déclarée.
- Vérifiez que chaque fonction utilisée dans une unité de compilation est bien déclarée au préalable.
- Pour chaque fichier exécutable créé par le Makefile, vérifiez que chaque fonction utilisée est définie exactement une fois, et qu'il existe exactement une définition de `main`.

3 Manipulation élémentaire de tableaux

Exercice 11. [initialisation de tableaux et affichage]
Quel affichage est produit par le programme suivant :

```c
#include <stdio.h>
#define TABSIZE 5
void print_tab_tabsize(int t[]){
  int i;
  for (i = 0; i < TABSIZE; i++)
    printf("%d, ", t[i]);
}
int te [TABSIZE];
int tf [TABSIZE] = {63, 64};
int main(void){
  int ta [TABSIZE] = {17,15,13,12,14};
  int tb [] = {27,25,23,22,24};
  int tc [TABSIZE] = {37,39};
  int td [TABSIZE];
  printf("ta : ");
  print_tab_tabsize(ta);
  printf("\ntb : ");
  print_tab_tabsize(tb);
  printf("\ntc : ");
  print_tab_tabsize(tc);
  printf("\ntd : ");
  print_tab_tabsize(td);
  printf("\nte : ");
  print_tab_tabsize(te);
  printf("\ntf : ");
  print_tab_tabsize(tf);
  putchar('\n');
  return 0;
}
```

Exercice 12. [Copies de tableau]
Soient les deux tableaux définis par :

```c
#define SIZE 12
int tsrc[SIZE], tdest[SIZE];
```

1. Donnez le code C permettant de copier `tsrc` dans `tdest`.
2. Donnez le code C permettant d'affecter à `tdest` les seuls éléments strictement positifs de `tsrc` en complétant la fin du premier tableau par des valeurs nulles.

Exercice 13. [Compteur de chiffres et de blancs]
On veut, au sein d'une fonction, compter les occurrences de chaque chiffre ('0' à '9') et des blancs (' ' ou `\t`) d'un texte lu sur stdin. Le résultat sera stocké dans un tableau de 10 entiers pour les chiffres et dans une variable entière pour les blancs.

Exercice 14. [Calcul de maximum absolu d'un tableau de double]

1. Donnez la déclaration d'une fonction d'identificateur `max_val` qui retourne la plus grande valeur, en valeur absolue, d'un tableau de valeurs double passé en paramètre.
2. Donnez la définition de la fonction `max_val`.
3. Soient les déclarations :

```c
#define SIZE 200
double m, t[SIZE];
```

Donnez le code C qui range dans la variable `m` la plus grande valeur absolue des éléments du tableau `t`.

Exercice 15. [Recherche dichotomique] Donnez la définition de la fonction de prototype :

```
int search(float val, float tab[], int size);
```

qui retourne un indice de cellules contenant la valeur `val` dans le tableau `tab` de taille `size`. Ce tableau est supposé trié et votre implantation doit faire une recherche dichotomique. De plus, cette fonction retourne -1 si la valeur `val` ne se trouve pas dans le tableau.

Exercice 16. [Affichage formatté] Donnez la définition d'une fonction de prototype :

```
void affiche_ligne_xxd(char tab[]);
```

qui affiche la chaîne de caractères codée par le tableau `tab` en respectant le format suivant :

- la première colonne de 8 caractères représente l'offset du premier caractère du paquet de 16 caractères considérés du fichier ;
- la seconde colonne comporte 32 caractères groupés par parquet de 4. Elle indique le codage hexadécimal des caractères considérés ;
- la dernière colonnes de 16 caractères correspond à l'affichage de ces derniers (les caractères non imprimables sont remplacés par un point). Pour rappel, les caractères imprimables ont une valeur ASCII entre 32 et 127.

Par exemple, appliquée à la chaîne `"La vie est belle.\n"`, la fonction affiche :

```
00000000: 4C61 2076 6965 2065 7374 2062 656C 6C65 La vie est belle
00000010: 2E0A                                    ..
```

4 Manipulation élémentaire de types composés

Exercice 17. [Comparaison de structures] Considérons la structure suivante:

```
struct date_s {
  int jour, mois, annee;
};
```

Donnez le code d'une fonction de prototype

```
int cmp_date(struct date_s, struct date_s);
```

qui compare si une date est plus récente qu'une autre. Cette fonction renverra 1 si la première date est plus récente, -1 si elle est plus ancienne, et 0 si les deux dates sont identiques. Pour éviter une répétition de code, il est suggéré d'écrire une macro adaptée.

Exercice 18. [Manipulations de formules] On désire manipuler des formules caractérisées par :

- un nom de variable est un identificateur auquel est associée une valeur flottante ;
- un terme est soit une valeur flottante immédiate, soit un nom de variable ;

Une formule peut être un terme, la somme de deux termes, le produit (ou le quotient) d'un terme et d'un coefficient entier ;

- l'évaluation d'un terme est la valeur flottante *associé* au terme.
- l'évaluation d'une formule est une valeur flottante *produite* par la formule.

Ainsi, les formules suivantes sont valides :

$$a, \qquad a + 4.0, \qquad a/4, \qquad a + b, \qquad b \times 3, \quad 5.0.$$

On propose les définitions et déclarations suivantes pour les variables et les termes:

```
/* Les variables */
#define NAME_LG 16 /* longueur d'un id de variable */
#define NVARS 32 /* nombre maximal de variables */

struct var_s {
  char v_name[NAME_LG]; /* nom de la variable */
  float v_val; /* valeur de cette variable */
};

static struct var_s vars[NVARS]; /* "table des symboles" */
static unsigned int nvars=0;     /* nombre de variables d\'efinies */

/* Les termes */
enum term_type_e {VAL_T, ID_T}; /* 2 types de termes~: flottant et identificateur */

struct term_s {
  enum term_type_e t_type;
  union { /* union anonyme */
    float t_val;
    char t_var[NAME_LG];
  } t_term;
};
```

1. Proposez une fonction d'identificateur `value_of_term` qui évalue un terme (i.e. qui retourne sa valeur). Vous aurez probablement besoin de deux fonctions intermédiaires: une pour évaluer une variable, et au sein de celle-ci, une fonction pour comparer deux chaînes de caractères (pour rappel, une chaîne de caractères est un tableau de caractères qui contient le caractère `\'0'` marquant la fin de la chaîne).

2. Proposez une définition de type pour représenter une formule.

3. Proposez une fonction d'identificateur `value_of_formula` qui évalue une formule (i.e. qui retourne la valeur d'une formule passée en paramètre).

Exercice 19. [Grands entiers positifs] On propose de manipuler de grands entiers positifs. Un grand entier positif est représenté par un vecteur de `GEP_SIZE` chiffres qui sont des `unsigned long`.

1. Proposez la définition d'un type de données pour mémoriser un grand entier positif avec la contrainte que l'en puisse utiliser l'opérateur d'affectation avec les grands entiers.

2. En proposant un nouveau type `gep` permettant de coder des grands entiers et une retenue, donnez le prototype d'une fonction qui retourne la somme de deux `gep` et l'éventuelle retenue.

3. Proposez une définition de cette fonction.