Lebanese University
Faculty of Sciences 1

I2204 F – INFO216 F

Programmation Impérative

Partiel 2019-2020
Lun. 16 Déc. pm
Durée : 1 heure

Exercice I: Méli-mélo (20 *pts* = 10×2 *pts*)

Pour chacun des codes suivants, indiquer s'il

- **a)** compile et s'exécute avec succès, **et donner la sortie**,
- **b)** génère une valeur arbitraire,
- **c)** produit une erreur de compilation, ou bien,
- **d)** provoque une erreur de segmentation.

1

```c
#include <stdio.h>
void foo(int*);
int main(){
    int i = 10, *p = &i;
    foo(p++);}
void foo(int *p){
    printf("%d\n", *p);}
```

2

```c
#include<stdio.h>
void main(){
    int i;
    for(i=0; i<10;printf("%d",i%4),i++);
}
```

3

```c
#include<stdio.h>
typedef struct point{int x,y;} Point;
Point * t(){
    Point p={1,2};
    return &p;
}
void v(){
    Point *p=t();
    printf("%d %d", p->x,p->y);
}
void main(){
    v();}
```

4

```c
#include<stdio.h>
int minmax(int *t, int size){
    int max=*t, min=*t;
    for(++t;size>=0;size--){
        if(min>*t) min=*t;
        else if(max<*t) max=*t;
        t++;
    }
    return min;
    return max;
}
void main(){
    int t[]={30,40,20,50,15};
    int min, max=minmax(t, 5);
    printf("min=%d, max=%d", min,max);}
```

5

```c
#include<stdio.h>
void swap(int x, int y){
    x+=y;
    y=x-y;
    x-=y;}
void main(){
    int x=78,y=29;
    swap(x,y);
    printf("x=%d, y=%d", x,y);}
```

6

```c
#include <stdio.h>
void main(){
    char *str = "hello, world\n";
    char *strc = "good morning\n";
    strcpy(strc, str);
    printf("%s\n", strc);
}
```

7

```c
#include<stdio.h>
void f3(int *p){
    printf("%d in f3, ", (*p)--);}
void f2(int *p){
    printf("%d in f2, ", ++*p);
    f3(p);}
void f1(int *p){
    f2(p);
    *p*=2;
    printf("%d in f1, ", *p);}
void main(){
    int x=3;
    printf("%d in main, ", x);
    f1(&x);}
```

8

```c
#include <stdio.h>
void f(char *k){
    int i;
    k+=2;
    k[2] = '\0';
    for(i=0;i<2;i++)
        printf("%c", *k--);
    printf(" %s", k--);
}
void main(){
    char s[] = "hello";
    f(s);
}
```

9

```c
#include<stdio.h>
int series(int n){
    if(n>0)
        return n + series(n-1);
}
void main(){
    printf("%d",series(3));
}
```

10

```c
#include <stdio.h>
void main(){
    float arr[5] = {12.5, 10.0, 13.5, 90.5, 0.5};
    float *ptr1 = &arr[0];
    float *ptr2 = ptr1 + 3;
    printf("%.1f ", *ptr2);
    printf("%ld", ptr2-ptr1);
}
```

Exercice II: Fortement connecté (10 pts)

Étant donné un graphe orienté, nous cherchons à vérifier s'il est fortement connecté ou non. On dit qu'un graphe orienté est fortement connecté si chaque sommet est accessible à partir de chaque autre sommet. Par exemple, le graphe ci-dessous, fig. (a), est fortement connecté car un chemin existe entre toutes les paires de sommets.

Une solution simple serait d'exécuter DFS à partir de chaque sommet du graphe. Si chaque DFS visite tous les autres sommets du graphe, le graphe est fortement connecté.

Nous représentons un graphe par sa liste d'adjacence. Il s'agit d'un tableau de nœuds de liste simplement chaînées. Chaque nœud de la liste représente un sommet. La fig. (b) montre la liste d'adjacence du graphe orienté de la fig. (a). Nous donnons l'algorithme DFSvisit à la fig. (c) qui visite tous les sommets possibles à partir d'un sommet donné u. Au début de chaque visite, les couleurs des sommets sont toutes initialisées en blanc (non visité).

1. Écrire la fonction "DFSvisit" en C. Astuce: vous devrez peut-être utiliser un paramètre supplémentaire.
2. Ecrire la fonction "check" qui retourne 1 si un graphe donné est fortement connecté, ou zéro sinon. Cette fonction doit utiliser "DFSvisit". Le prototype de check doit correspondre à l'appel de fonction dans checkTest().

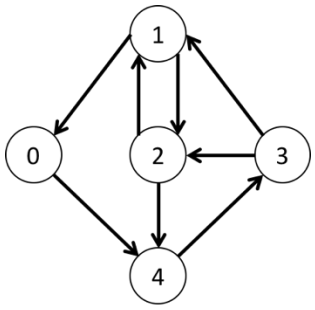

fig. (a)

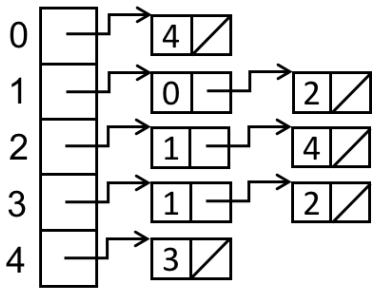

fig. (b)

```
DFSvisit(G, u)
   color[u]=gray;
   for each v in Adj(u)  do
      if (color[v] = white)
         DFSvisit(G,v);
      endif
   endfor
color[u] = black;
end
```
fig. (c)

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

#define nb    5

#define white 0
#define gray  1
#define black 2

typedef struct node{
    int vertex;
    struct node * next;
}node;

void Push(node **headRef, int v){
    //alloc and fill
    node *newNode =
         (node *) malloc(sizeof(node));
    newNode->vertex = v;
    //link next
    newNode->next = *headRef;
    //link head
    *headRef = newNode;
}
```

```c
void fillGraph(node** graph, int size){
    //create the adjacency list of the example
    Push(&graph[0],4);
    Push(&graph[1],2);
    Push(&graph[1],0);
    Push(&graph[2],4);
    Push(&graph[2],1);
    Push(&graph[3],2);
    Push(&graph[3],1);
    Push(&graph[4],3);
}

void checkTest(){
    node *graph[nb];
    int i;
    for(i=0;i<nb;i++)
        graph[i]=NULL;
    fillGraph(graph, nb);
    char *s=(check(graph,nb)==1)?"":"not";
    printf("Graph %s strongly connected ",s);
}

int main(int argc, const char * argv[]) {
    checkTest();
    return 0;
}
```