Lebanese University
Faculty of Sciences 1

I2204 E – INFO216 E

Imperative Programming

Partial 2019-2020
Mon. Dec. 16th pm
Duration: 1 hour

Exercise I: Mishmash (20 *pts* = 10×2 *pts*)

For each of the following codes, indicate whether it

- **a)** compiles and runs successfully, **then give the output**,
- **b)** outputs some garbage value,
- **c)** produce a compile time error, or,
- **d)** can cause a segmentation fault.

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);
}
```

Exercise II: Strongly connected (10 pts)

Given a directed graph, we seek to check if it is strongly connected or not. A directed graph is said to be strongly connected if every vertex is reachable from every other vertex. For example, the below graph, fig. (a), is strongly connected as path exists between all pairs of vertices.

A simple solution would be to perform DFS starting from every vertex in the graph. If each DFS visits every other vertex in the graph, the graph is strongly connected.

We represent a graph by its adjacency list. It is an array of simply linked list nodes. Each node in the list represents a vertex. The fig. (b) shows the adjacency list of the directed graph in fig. (a). We give the DFSvisit algorithm in fig. (c) which visits all possible vertices starting from a given vertex u. At the beginning of each visit, the vertices colors are all initialized to white (not visited).

1. Write the function "DFSvisit" in C. Hint: you may need to use one additional parameter.
2. Write the function "check" which returns 1 if a given graph is strongly connected, or zero otherwise. This function must use "DFSvisit". The prototype of check must match the function call in checkTest().

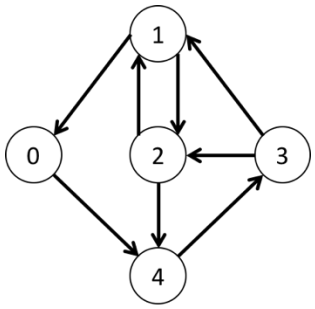

fig. (a)

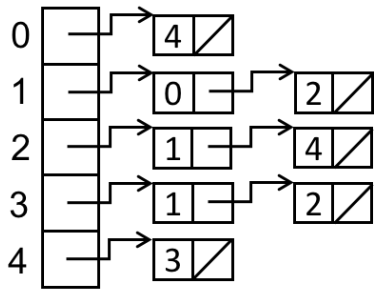

fig. (b)

```
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;
}
```