INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES CENTRE VAL DE LOIRE	3ème Année FISE Programmation C	2023/2024
		Partiel

ATTENTION: Je me réserve le droit de pénaliser les mauvaises réponses avec des points négatifs, et aussi les réponses ambiguës et celles bonnes mais contenant des justifications farfelues.

Partie A :10 pts

1) Pendant la compilation du **module** errdef.c, le compilateur signale une erreur de syntaxe :

// gcc –W –Wall –std=c99 –c errdef.c

```
#define max 100

int err( int v[]) {
  int max ;
  for (int i= 0 ;v[i]>0 ; i++) {
   if (v[i]> max)
     max=v[i];
  return max;
}
```

```
main.c: In function 'err':
main.c:11:13: error: expected identifier or '(' before numeric constant
   11 | #define max 100
      |             ^~~
main.c:13:8: note: in expansion of macro 'max'
   13 |    int max ;
      |        ^~~
main.c:16:12: error: lvalue required as left operand of assignment
   16 |         max=v[i];
      |            ^
```

Identifiez précisément l'origine de cette erreur. Proposez une correction

2) Donnez la déclaration qui correspond à l'affirmation suivante : *La variable **area** est un pointeur constant sur une fonction qui prend deux arguments muets de même type int et retourne une valeur de type int.*

3) Après la séquence
int t[2]; t[0]=1; t[1]=2;
L'expression *(*t +1) a pour type et pour valeur (ou si autre chose, précisez cette autre chose) :

4) Quel est le type de l'identificateur ***t*** dans la déclaration suivante : int *t[3](int a,int b)

5) Qu'affiche la séquence de code suivante ? Justifiez votre réponse !

```
int v[2][3] = {{1, 2, 3}, {4, 5, 6}};
printf("%d", ((int *)v)[5]);
```

6) Soit la séquence de code suivante

```
…
struct pixel {
  short x; short y;
  struct {
    unsigned short rouge; unsigned short vert; unsigned short bleu;
  } couleur;
};
struct pixel *px;
```

Donnez l'instruction qui permet d'accéder au champ « bleu » en partant de la variable « px ». Donnez l'instruction qui permet de calculer la taille, en octet, de la structure pointée par « px ».

7) Décrire de manière concise ce qu'elle ***réalise*** la fonction allouer_chaine définie ainsi

```
char * allouer_chaine(char * source) {
   char *copie= malloc(strlen(source) +1) ) ;
    return copie ;
}
```

8) Considérons le programme suivant :

```
#include <stdio.h>
#include <string.h>
int main() {
  const char *adc= "salut" ; char *ad= = "hello" ;
  strcpy(adc,  "bonjour" );
  return 0 ;
}
```

Pendant la compilation, le compilateur émet juste un message d'avertissement. Pendant l'exécution, le programme se termine avec le message « segmentation fault ». Donnez une explication à ce comportement

9) Supposons que nous ayons un tableau « 2D », int t[2][3], dans lequel 6 entiers sont stockés. Donnez une expression équivalente à l'expression suivante *(*(t+1)+2)

10) Si ***f*** est une fonction renvoyant un pointeur générique et ayant comme paramètres un entier et un pointeur sur un caractère, on pourra utiliser comme prototype de f :

Partie B : 6 pts

11) La fonction ***strrchr*** est une fonction de la librairie standard :
char * **strrchr**(const char * **string**, int **c**) // la letter r du milieu= reverse

Cette fonction recherche la dernière occurrence du caractère ***c*** passé en second paramètre dans la chaîne de caractères ***string*** spécifiée via le premier paramètre. Soit le caractère **c** recherché est présent dans la chaîne **string** et, dans ce cas, un pointeur sur la dernière occurrence du caractère vous sera retourné. Soit le caractère n'est pas présent dans la chaîne et dans ce cas, le pointeur NULL vous sera renvoyé.

Considérons cette définition de la fonction ***nomDeBase*** :

```
#include ….
char *nomDeBase(char *chemin) {
  int len= strlen(chemin) ;
  char *p=(char *)strrchr(chemin,'/') ;
  if (p==NULL)
   return strcpy(malloc(len+1), chemin);
  else
  return strcpy(malloc(len -(p-chemin)), p+1);
}

int main()
{
   printf("%s\n",nomDeBase("/usr/bin/emacs"));
   //affiche?
   printf("%s",nomDeBase("emacs"));
   //affiche?
   return 0;
}
```

12) Soit le programme suivant :

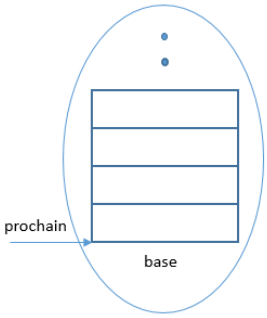
```
#include <stdio.h>

int main()
{
   int i=0;
   for (i=0;i<20;i++) {
      switch(i) {
         case 0: i+=5;
         case 1: i+=2;
         case 5: i+=5;
         default : i+=4;
         break;
      }
      printf("%d " ,i);
   }
}
```

Son exécution provoque l'affichage suivant :

13) Nous allons définir une pile d'entiers comme un couple <base, prochain> où ***base*** est tableau d'entiers et **prochain** un pointeur référençant le premier élément libre de la pile (dans ***base***). Lorque la pile est vide, le champ prochain pointe sur la première case de ***base***. Voici, sur ce modèle, la définition d'une pile d'entiers :

```
typedef struct {
   int base[TAILLE_PILE] ;
   int *prochain ;
} pile_d_entiers ;
```



Les opérations définies sur cette pile sont classiquement: empiler un entier, dépiler, consulter le sommet de la pile, etc…

Complétez la définition de la fonction ***empiler*** (aucun contrôle de débordement n'est demandé). Avant l'appel à la fonction ***empiler***, vous partez de l'hypothèse que la pile a été correctement instanciée ailleurs et initialisée avec cette instruction :

laPile.prochain=base ;

```
void empiler( pile_d_entiers pe, int e) {




}
```

PARTIE C : 6 points

14) On considère le module ***Shape*** suivant :

shape.h :

```
1   #ifndef __SHAPE_H
2   #define __SHAPE_H
3   #include <stdint.h>
4   /* Shape's attributes... */
5   typedef struct
6   {
7       int16_t x; /* x-coordinate of Shape's position */
8       int16_t y; /* y-coordinate of Shape's position */
9   } Shape;
10
11  /* Shape's operations (Shape's interface)... */
12  void Shape_ctor(Shape *const me, int16_t x, int16_t y); //constructor
13  void Shape_moveBy(Shape *const me, int16_t dx, int16_t dy);
14  #endif
```

shape.c :

```
1   #include "shape.h"
2
3   /* constructor */
4   void Shape_ctor(Shape *const me, int16_t x, int16_t y)
5   {
6       me->x = x;
7       me->y = y;
8   }
9   /* move-by operation */
10  void Shape_moveBy(Shape *const me, int16_t dx, int16_t dy)
11  {
12      me->x += dx;
13      me->y += dy;
14  }
```

Dévinez et complétez le code des fonctions suivantes « d'après leurs noms » qui se trouvent dans shape.c :

```
18  /* "getter" operations implementation */
19  int16_t Shape_getX(const Shape *const me)
20  {
21
22
23  }
24  int16_t Shape_getY(const Shape *const me)
25  {
26
27
28  }
```

Un exemple d'utlisation du module ***shape*** se trouve dans **app.c** :

app.c

```
#include "shape.h"
int main() {
  // Déclarez deux variables s1 et s2 de type Shape
```

```
 // Avec Shape_ctor, donnez à sl la position initiale (0,1) et à s2 la position (-1,2)
```

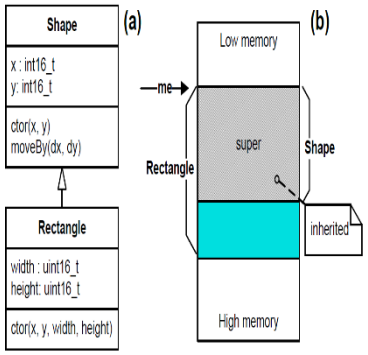
```
 // Déplacez s1 de deux unités (+2) sur l'axe des abscisses et de 4 unités (-4) sur l'autre axe
```

```
}
```

15) On considère maintenant le module **Rectangle** (Rectangle est un cas particulier de Shape). La structure **Rectangle** est définie avec 3 champs : le champ *super* de type **Shape** et le deux champs *width* (largeur) et *height* (hauteur) du rectangle.

rectangle.h

```
1  #ifndef __RECT_H
2  #define __RECT_H
3  #include "shape.h"
4
5  typedef struct
6  {
7      Shape super; /* <== inherits Shape */
8
9      /* attributes added by this subclass... */
10     uint16_t width;
11     uint16_t height;
12 } Rectangle;
13
14 /* constructor */
15 void Rectangle_ctor(Rectangle *const me, int16_t x, int16_t y, uint16_t width, uint16_t height);
16
17 #endif
```



rectangle.c

```
#include "rectangle.h"
void Rectangle_ctor(Rectangle * const me, int16_t x, int16_t y, uint16_t width, uint16_t height)
{
/* Initialisez correctement le champ super (qui est une structure) de la structure avec les paramètres x et y */
```

/* **Initialisez** correctement les champs ***width*** et ***height*** de la structure */

}

16) Complétez le code l'application app

app.c

```
#include "rectangle.h"
int main() {
  // Déclarez deux variables r1 et r2 de type Rectangle

 // Avec Rectangle_ctor, construisez le rectangle r1 (x=0, y=2, width=10, height=15)
// et le rectangle r2 (x=-1, y=3, width=5, height=8)

// Déplacez r1 de deux unités (+2) sur l'axe des abscisses et de 4 unités (-4) sur l'autre axe

}
```

17) On considère maintenant le module **Circle** (Circle est un cas particulier de Shape comme Rectangle) : Fournissez la structure adéquate à ce module dans circle.h et le prototype de la fonction de construction Circle_ctor (en suivant la même démarche utlisée pour le module Rectangle)

```
#ifndef CIRCLE_H
#define CIRCLE_H

#endif
```