QROC RN Informatique – Décembre 2012

Répondre sur la feuille impérativement. Notes personnelles et polycopiés distribués seuls autorisés. Calculatrice autorisée.

NOM : *Corrigé*
Prénom :
Filière :

Compilation

- Chaque ligne du tableau ci-dessous contient une portion de code qui doit être considérée comme indépendante des autres. Indiquer (par une croix) si la portion de code est correcte ou provoque une erreur de compilation ou peut provoquer un effet indésirable à l'exécution (les en-têtes ont été omis volontairement)

Code	Correct	Pb Compilation	Pb possible à l'exécution

`int main() {`
`  double x = 7;`
`  y = x + 2;`
`  printf("x=%lg y=%lg\n", x, y);`
`  return 0;`
`}`
		X	

`int t[10];`
`int main() {`
`  int i;`
`  for (i = 0; i <=100; i++) {`
`  t[i] = i ;`
`  printf("%d ", t[i]);`
`  }`
`  return 0;`
`}`
			X

`int n,m,i;`
`int tab[5];`
`int * p;`
`int main() {`
`  for(i=0;i<5;i++) tab[i]=i*i;`
`  p=&tab[3];`
`  printf("tab[3]= %d\n",*p);`
`  return 0;`
`}`
	X		

Pointeurs

On considère la portion de code suivante :

```c
#include <stdio.h>
#include <stdlib.h>
int i,n,m;
int *p;
int tab[100];

int main(void) {
     // section 1
     for(i=0;i<100;i++) tab[i]=i*i;
     for(i=0;i<100;i=i+30) printf("tab[%d]=%d\n",i,tab[i]);
     // section 2
     n=tab[5];
     p=&tab[5];
     *p = *p +2;
     printf("n=%d   tab[5]=%d  *p=%d\n",n,tab[5],*p);
     // section 3
     p=&n;
     *p = *p +2;
     printf("n=%d   tab[5]=%d  *p=%d\n",n,tab[5],*p);
     return 0;
}
```

- Pour chaque section indiquer ce qui apparaîtra sur la console lors de l'exécution du programme

```
// section 1
tab[0]=0
tab[30]=900
tab[60]=3600
tab[90]=8100
// section 2
tab[5]=27 *p=27 n=25
// section 3
tab[5]=27 *p=27 n=27
```

Langage C

Le programme ci-dessous représente la version procédurale, telle que vue en cours, de l'algorithme de résolution d'une équation du premier degré.

```
void reso(int a,int b, double * px,int * pcode) {
    if (a==0) {
        if (b==0) *pcode=2;
        else *pcode=0;
    }
    else {
        *pcode=1;
        *px=-b/(double)a;
    }
}
int a,b,code;
double x;
int main() {
    printf("a ");
    scanf("%d",&a);
    printf("b ");
    scanf("%d",&b);
    // resolution
    reso(a,b,&x,&code);
    // affichage
    switch(code) {
        case 0:printf("0 sol.");break;
        case 1:printf("1 sol.=%lg",x);break;
        case 2:printf("inf. sol.");break;
    }
    return 0;
}
```

On souhaite utiliser une version mixte (souvent pratiquée en C) mélangeant l'aspect fonctionnel et l'aspect procédural. Dans cette version la valeur du code (0, 1 ou 2) est fournie à l'appelant en retour de fonction *reso*. La valeur de la solution est fournie, comme ci-dessus, au moyen de l'adresse passée par l'appelant.
Avac cette version le programme de test devient le suivant :

```
int a,b,code;
double x;
int main() {
    printf("a ");
    scanf("%d",&a);
    printf("b ");
    scanf("%d",&b);
    // resolution
    code=reso(a,b,&x);
    // affichage
    switch(code) {
        case 0:printf("0 sol.");break;
        case 1:printf("1 sol.=%lg",x);break;
        case 2:printf("inf. sol.");break;
    }
    return 0;
}
```

- Ecrite la signature de cette version de *reso* (sans paramètres formels)

```
int reso ( int , int , double * )
```

- Ecrire la définition complète de cette version de *reso* en y faisant bien sûr figurer les paramètres formels :

```
int reso ( int , int , double * ) {
   if (a==0) {
        if (b==0) return 2;
        else return 0;
    }
    else {
        *px=-b/(double)a;
        return 1 ;
    }
 }
```

Réingénierie

Le programme ci-dessous permet de tester le caractère premier d'un nombre entier.

```
#include <stdio.h>
#include <stdlib.h>
int n,d,fin=0;;
int main(void) {
     printf("Entrer un nombre ");
     scanf("%d",&n);
     d=2;
     while(d<n && !fin) {
          if (n % d == 0) {   // voir opérateur %[1] ci-dessous
               printf("%d n'est pas un nombre premier",n);
               fin=1;
          }
          d++;
     }
     if (! fin) printf("%d est un nombre premier",n);
     return 0;
}
```

- Écrire, en déduisant le code de cet exemple, la fonction *int isPrime(int n)* qui retourne *0* (c'est à dire *faux*) si le nombre *n* n'est pas premier et *1* (c'est à dire *vrai*) si *n* est premier.

```
int isPrime(int n) {
     int d=2;
     while(d<n) {
          if (n%d==0) return 0;
          d++;
     }
      if (d==n) return 1;
}
```

[1] *Rem : l'opérateur % est l'opérateur modulo du langage C (il correspond au reste de la division euclidienne des 2 opérandes).*
Exemples : 7 % 3 vaut 1
20 % 4 vaut 0
18 % 5 vaut 3

On appelle nombres premiers cousins des couples de nombres premiers dont la différence vaut 4.
Exemples des premiers nombres premiers cousins : (3, 7), (7, 11), (13, 17), (19, 23), (37, 41), (43, 47), (67, 71), (79, 83), (97, 101), (103, 107), (109, 113), (127, 131), (163, 167), (193, 197), (223, 227), (229, 233), (277, 281), (307, 311), (313, 317), (349, 353), (379, 383), etc...

- Écrire, en vous servant de la fonction *isPrime* ci-dessus (que vous l'ayez écrite ou non vous ferez l'hypothèse que vous en disposez) le programme qui affiche comme ci-dessus les couples de nombres premiers cousins inférieurs à une limite choisie par l'utilisateur.

```
#include <stdio.h>
#include <stdlib.h>
int isPrime(int n) {
    int d=2;
    while(d<n) {
        if (n%d==0) return 0;
        d++;
    }
    if (d==n) return 1;
}
int max;
int main(void) {
    int n;
    printf("Entrer une limite : ");
    scanf("%d",&max);
    for(n=2;n<max-4;n++) {
        if (isPrime(n) && isPrime(n+4))
                printf("(%d,%d), ",n,n+4);
    }
    return 0;
}
```