Trimestre Été, 2008

Mohamed Lokbani

IFT1166 – Examen Intra – Solutionnaire -

Inscrivez tout de suite votre nom et le code permanent.

Nom: _____| Prénom(s): _____|

Signature: _____| Code perm: _____|

Date : mercredi 4 juin 2008

Durée : 2 heures (de 17h30 à 19h30)

Local : Z-330 ; Pavillon Claire McNicoll

Directives:

- Toute documentation permise.
- Calculatrice **non** permise.

- Répondre directement sur le questionnaire.
- Les réponses **doivent être brèves, précises, claires et nettement présentées.**

1._____/08 (1.1, 1.2, 1.3, 1.4)

2._____/20 (2.1, 2.2, 2.3, 2.4)

3._____/12 (3.1, 3.2, 3.3, 3.4)

4._____/20 (4.1, 4.2)

5._____/20 (5.1, 5.2)

6._____/20 (6.1)

Total:_____/100

Directives officielles

* Interdiction de toute communication verbale pendant l'examen.

* Interdiction de quitter la salle pendant la première heure.

* L'étudiant qui doit s'absenter après la première heure remettra sa carte d'étudiant au surveillant, l'absence ne devant pas dépasser 5 minutes. Un seul étudiant à la fois peut quitter la salle.

* Toute infraction relative à une fraude, un plagiat ou un copiage est signalée par le surveillant au directeur de département ou au professeur qui suspend l'évaluation.

F.A.S

Exercice 1 (08 points) Encerclez la bonne réponse et donnez une courte explication.

1.1 [Vrai | Faux] Le langage C++ est apparu après le langage Java ?

Le langage Java est apparu en 1995 alors que le langage C++ est apparu en 1982.

1.2 [Vrai | Faux] Pour utiliser le manipulateur de flux « setprecision() », il faudra inclure le fichier d'en-tête « iostream » ?

Pour utiliser « setprecision » il faudra inclure le fichier d'en-tête « iomanip ».

1.3 [Vrai | Faux] L'expression à évaluer dans l'instruction de contrôle « switch » peut avoir n'importe quel type ?

L'expression à évaluer doit retourner une valeur du type « int ».

1.4 [Vrai | Faux] Nous ne pouvons passer un paramètre à une fonction que de deux manières différentes ?

Il y a 3 manières pour passer une valeur à une fonction : par valeur, par pointeur et par référence.

Exercice 2 (20 points) Tout ce qui suit est syntaxiquement correct (compile et s'exécute correctement)

2.1 À l'aide d'un exemple, expliquez le rôle joué par la macro suivante :

```
01 | #define A(x) ((x==0)?0:1)
```

La macro permet de transformer une valeur entière en son équivalent logique « 0 » ou « 1 ».

Si « x=0 » la valeur retournée est « 0 », alors que pour tous les autres cas, la valeur retournée est « 1 ».

2.2 Tout en détaillant votre réponse, que va afficher en sortie le fragment suivant :

```
01      enum Semaine {Lu, Mar, Mer, Je, Ve, Sa, Di};
02      Semaine a[] = { Mer, Je, Mar, Di, Ve };
03      for (int i = 0; i < 5; i++)
04          a[i] = a[a[i]];
05      for (int i = 0; i < 5; i++)
06          cout << "a[" << i << "]= " << a[i] << endl;
```

Affichage en sortie :
a[0]= 1
a[1]= 6
a[2]= 6
a[3]= 2359160
a[4]= 4

Explications :
a[0]= a[a[0]] = « a[Mer] », mais comme « Mer = 2 », = a[2] = « Mar » = 1
a[1]= a[a[1]] = « a[Je] », mais comme « Je = 3 », = a[3] = « Di » = 6
a[2]= a[a[2]] = « a[Mar] », mais comme « Mar = 1 », = a[1], or a[1] vient d'être modifié et vaut maintenant « 6 »
a[3]= a[a[3]] = « a[Di] », mais comme « Di = 6 », = a[6] mais « a » ne contient que 5 éléments. Donc affichage d'une valeur arbitraire.
a[4]= a[a[4]] = « a[Ve] », mais comme « Ve = 4 », = a[4] = « Ve » = 4

2.3 À l'aide d'un exemple, quel est le rôle joué par cette macro ?

```
01  #define B(x,i) ((x)|(1<<(i)))
```

La macro « B » permet de mettre à « 1 » le bit de position « i ».

B(3,3) : x = 3 = 0011 ; (1<<3 ⇔ « 0001 » décalé 3 fois « 1000 » donc 8). Comme il y a l'opérateur « ou i.e. | » de bits « 0011 | 1000 » le résultat est « 1011 ». B(3,3) = 11 (sur des nombres non signés).

2.4 Tout en détaillant votre réponse, à quoi sert la fonction suivante :

```
01  int A(const char *s1, const char *s2) {
02      while ((*s1 != '\0') && (*s1 == *s2)) {
03          s1++; s2++;
04      }
05      return *s1 - *s2;
06  }
```

La fonction sert à comparer deux chaînes de caractères[1]. La comparaison débute avec le premier caractère de chaque chaîne et continue avec les caractères suivants. La fonction s'arrête dès que des caractères de même rang soient différents (le test *s1 == *s2) ou bien que la chaîne « s1 » soit terminée (le test *s1 != '\0').
Comme on compare des caractères, donc des valeurs entières (à cause de la conversion implicite vers le code ascii), la fonction retourne une valeur :
<0 si s1 est inférieure à s2 (par exemple s1 = "a" et arrive donc avant s2 = "b").
=0 si les eux chaînes sont identiques.
>0 si s1 est supérieure à s2 (par exemple s1 = "b" arrive après s2 = "a").

[1] Source: « Guide de référence Borland C++ ».

Exercice 3 (12 points) Soit « x » un nombre non signé initialisé avec la valeur « 5 ». Ce nombre est représenté comme suit sur 1 octet : « 00000101 ». Tout en développant votre réponse, dites ce que vont afficher en sortie les fragments de code suivants :

3.1
```
unsigned char c = 5;
cout << (c << 2) << endl;
```

Affichage : 20

c⇔ 00000101 ; (c << 2) décalage deux fois du nombre « c » ; la première fois « 00001010 » ; la seconde fois « 00010100 » et ce nombre correspond à 16+4 = 20.

3.2
```
unsigned char c = 5;
unsigned char x = ~c;
cout << (int) x << endl;
```

Affichage : 250

x est la négation bit à bit de c. c ⇔ 00000101 donc x ⇔ 11111010 et ce nombre vaut en décimal 128+64+32+16+8+2 = 250.

3.3
```
unsigned char c = 5;
cout << (c & 2) << endl;
```

Affichage : 0

Nous effectuons le « & » (et) logique mais au niveau des bits. Le « & » bit à bit entre « 00000101 » et « 00000010 » va donner « 00000000 » donc la valeur « 0 » car il n'y a aucuns bits communs à 1.

3.4
```
unsigned char c = 5;
cout << (c | 12) << endl;
```

Affichage : 13

Nous effectuons le « | » (ou) logique mais au niveau des bits. Le « | » bit à bit entre « 00000101 » et « 00001100 » va donner « 00001101 » donc la valeur « 13 » qui correspond à tous les bits communs qui sont à 1.

Exercice 4 (20 points) Tout en développant votre réponse, que vont afficher en sortie les fragments de code suivants qui compilent et s'exécutent correctement (dans un programme complet).

4.1

```
01 |    char s[]="CHAINE", *p=s+3;
02 |    cout << p << endl;
03 |    cout << p[-1] << endl;
04 |    cout << --*++p << endl;
```

Affichage :
INE
A
M

Explications :
« p » pointe le 4e caractère de la chaîne « s ». De ce fait, le premier caractère de « p » est la lettre « I ». Quand « p » est affiché en sortie, nous allons afficher tout le contenu de « p » jusqu'au caractère fin de chaîne, d'où l'affichage « INE ».
On affiche maintenant p[-1], comme p[0] contient s[3] donc p[-1] va contenir s[2] qui est la lettre « A ».
« ++p » permet d'incrémenter le pointeur « p ». « p » pointe maintenant la lettre « N ». « *p » permet de déréférencer le pointeur pour avoir son contenu qui est « N ». « --xyz » permet d'afficher la lettre qui précède « N » i.e. « M ».

4.2

```
01 |    int a[4]={10,20,30,40};
02 |    cout << a[1/2] << endl;
03 |    int* ptr=a+1;
04 |    cout << a-ptr << endl;
05 |    cout << ptr[1] << endl;
06 |    const int& r = *ptr;
07 |    cout << r+1 << endl;
08 |    cout << --*++ptr << endl;
```

Affichage :
10
-1
30
21
29

Explications :
« 1/2 », division entière donc le résultat est « 0 » et a[0] c'est « 10 ».
« ptr = a+1 » pointe maintenant le second élément du tableau « a », par ailleurs « a » pointe le premier élément « a[0] ». De ce fait « a-ptr » correspond au nombre d'éléments qui séparent « a » et « ptr ». Un calcul indirect peut-être formulé comme suit : « &a – (&a + 1) = -1) d'où l'affichage de « -1 » en sortie.
« la ligne 03 » a permis d'incrémenter le pointeur, il pointe maintenant la valeur « 20 ». La ligne « 05 » demande la valeur de p à la position « p+1 » donc « 30 ».
« r » est une référence initialisée avec le contenu du pointeur « 20 ». On affiche « r+1 » donc « 21 ».
« ++ptr » permet d'incrémenter le pointeur, il va pointer la valeur « 30 » du tableau a (donc a[2]). « *ptr » déréférence le pointeur, elle retourne la valeur « 30 ». « --xyz » on décrémente cette valeur, d'où l'affichage « 29 ».

Exercice 5 (20 points) Le programme suivant compile correctement. Mais lors de son exécution, il plante royalement et affiche en sortie le message suivant « Segmentation fault (core dumped) ». Ce message correspond à un incident de segmentation.

5.1 Expliquez brièvement l'une des causes d'une telle erreur.

C'est un débordement de la mémoire. Le programme a tenté d'accéder à une zone mémoire inexistante par exemple. Le fragment de code suivant permet de déréférencer un pointeur invalide car il pointe vers une adresse nulle.

```
int *p = NULL ;
p = 100 ;
```

5.2 Tout en développant votre réponse, corrigez les différentes erreurs conceptuelles pour que le programme puisse s'exécuter correctement et faire ce qu'il est supposé i.e. déclarer une structure, initialiser les éléments de la structure et afficher ces éléments en sortie. Vous pouvez inclure de nouvelles instructions si vous le jugez nécessaire. Dans ce cas, précisez entre quelles lignes.

```
01  #include <iostream>
02
03  using namespace std;
04
05  typedef struct {
06      char *nom;
07      char *prenom;
08      int age;
09  } type_fiche;
10
11  int main(){
12      type_fiche *fiche=NULL;
13
14      fiche->nom = new char;
15
16      strcpy(fiche->nom, "Sasseur");
17
18      strcpy(fiche->prenom, "Marlene");
19
20      fiche->age = 3;
21
22      cout << "Nom: " << fiche->nom << endl;
23      cout << "Prenom: " << fiche->prenom << endl;
24      cout << "Age: " << fiche->age << endl;
25
26      return 0;
27  }
```

La ligne « 14 » : nous avons utilisé une instance de « type_fiche » avant de lui avoir alloué un espace mémoire pour la contenir. Il faudra donc introduire à la ligne « 13 » le fragment de code suivant :

```
fiche = new type_fiche;
if (fiche == NULL){
    cerr << "prb allocation mémoire pour la structure fiche" << endl;
    exit(1);
}
```

La ligne « 16 » : nous avons copié dans la chaîne « nom », huit (8) caractères (« Sasseur » + '\0'), or la chaîne « nom » ne peut contenir qu'un seul caractère suite à l'instruction de la ligne « 14 ». Il faudra donc modifier la ligne « 14 » comme suit :

```
// « nom » ne peut contenir plus de 10 caractères y compris '\0'
fiche->nom = new char[10];
if (fiche->nom == NULL){
    cerr << "prb allocation mémoire pour la chaîne nom" << endl;
    exit(1);
}
```

La ligne « 18 » : nous avons copié dans la chaîne « prenom », huit (8) caractères (« Marlene » + '\0'), or l'espace pour contenir la chaîne « prenom » n'a pas été alloué. Il faudra donc introduire à la ligne « 17 » le fragment de code suivant :

```
// « prenom » ne peut contenir plus de 10 caractères y compris '\0'
fiche->prenom = new char[10];
if (fiche->prenom == NULL){
    cerr << "prb allocation mémoire pour la chaîne prenom" << endl;
    exit(1);
}
```

La ligne « 25 » : il faudra penser à libérer l'espace mémoire alloué. Il faudra donc introduire à la ligne « 25 » le fragment de code suivant :

```
delete [] fiche->nom;
delete [] fiche->prenom;
delete fiche;
```

Exercice 6 (20 points) Écrivez un programme C++ qui réalise ce qui suit :

1- Il inverse l'ordre des arguments entrés sur la ligne de commandes sauf l'argument « 0 » qui représente le nom du programme. Par exemple si la ligne de commandes contenait la suite [arg1, arg2, arg3, arg4], le résultat sera [arg4, arg3, arg2, arg1].
2- Il inverse par la suite l'ordre des caractères pour chaque argument. Par exemple si l'argument 3 contenait la chaîne « exemple », le résultat sera « elpmexe ».
3- Une fonction « main » pour tester ces fonctionnalités et afficher les arguments modifiés dans l'ordre spécifié.

Vous ne pouvez utiliser aucune fonction de la librairie « C » (par extension « C++ ») permettant de manipuler les chaînes de caractères. Vous devez réinventer la roue s'il le faut !

```
#include <iostream>

using namespace std;

/*  StrTaille sert à calculer la longueur de la chaîne s.
    StrTaille retourne un "int" représentant le nombre de caractères de s.
    Le caractère null d'arrêt ('\0') n'est pas compté.
*/

int StrTaille(char* s) {
    int Taille;
    for (Taille = 0 ; *s != '\0' ; s++) Taille++;
    return Taille;
}

/*  StrInverse sert à inverser la chaîne s.
    Nous utilisons pour cela la technique de "swap" (inter changer).
*/

void StrInverse(char* str){
    int i, j, len;
    char temp;
    i=j=len=temp=0;
    len=StrTaille(str);
    for (i=0, j=len-1; i<=j; i++, j--){
        temp=str[i];
        str[i]=str[j];
        str[j]=temp;
    }
}

int main (int argc, char * argv[]) {
    int nbreargs = argc-1;
    for (int i = nbreargs ; i > 0 ; i--){
        StrInverse(argv[i]);
        cout << argv[i];
        if (i!=1) cout << ' ';
    }
    return 0;
}

/*     exo6.exe elpmexe nu tse icec
       ceci est un exemple

ou bien
       exo6.exe ceci est un exemple
       elpmexe nu tse icec
*/
```