Université de Nice-Sophia Antipolis
Licence Informatique - 2e année
U.E. Introduction au Langage C 2010–2011

Épreuve de contrôle continu
du Lundi 4 Avril 2011

Nom
Prénom
Né(e) à
Le

Durée : 2 heures
Tous documents autorisés

Note

Il est de votre responsabilité de rabattre le triangle grisé et de le cacheter au moyen de colle, agrafes ou papier adhésif. Si ne vous le faites pas, vous acceptez implicitement que votre copie ne soit pas anonyme.

Toutes les questions sont indépendantes.
Tous les codes sources devront être écrit en langage C.

1 Conversion (3 points)

Donner la déclaration et la définition d'une fonction `convertir` qui prend en paramètre une chaine de caractères représentant un entier, et qui retourne cet entier.
Par exemple si la chaine contient "1234" alors la fonction devra retourner l'entier 1234. Vous n'avez pas le droit d'utiliser la fonction atoi de la librairie ; c'est à vous d'écrire son équivalent. Les caractères correspondant aux chiffres sont encodés à partir du code ASCII 48 (pour '0') jusqu'à 58 (pour '9'). Les caractères qui ne sont pas des chiffres seront ignorés par la fonction lors de la conversion.

```
/* la gestion du signe est en bonus */
int conversion(char* ch){
  int res=0;
  while (*ch){
    if (*ch >= '0' && *ch <='9'){
     res += 10*res + (*ch - '0');
    }
    ch++;
  }
  return res;
}
```

2 Compréhension de code (3 points)

1. Expliquer ce que fait le programme suivant (rappel : le dernier argument du tableau des paramètres est le pointeur NULL) :

   ```
   while (*++argv) {printf("%s\n",*argv);}
   ```

   ```
   Ce code affiche la liste des parametres utilisés  sur la ligne de commande
   ```

2. Soit le code suivant :

   ```
   char ch1[]="123"; char ch2[]="456"
   char* chr = ch1 + ch2;
   ```

   Expliquer ce qui se passe.

   ```
   le pointeur chr est initialisé avec la somme de deux adresses: celle de ch1 et
    celle de ch2. Cela n'a pas beaucoup de sens et ne fait en aucun cas la
   concaténation des deux chaines de caractères.
   ```

Donner le code correct pour concatener les deux chaines ch1 et ch2 dans la chaine chr ;

```
void concat(char* chr, char* ch1, char* ch2){
  while (*dest++=*ch1++);
  res--;
  while (*dest++=*ch2++);
}
```

3 Réallocation de tableau (3 points)

On considère une fonction `reallouerTab` qui prend trois paramètres : un tableau d'entier, la taille de ce tableau exprimée en nombre d'éléments et la nouvelle taille du tableau (exprimée aussi en nombre d'éléments) et qui réalloue le tableau à la nouvelle taille et ne perd pas les éléments courants dans le tableau. La fonction ne retourne rien. Faites bien attention à bien gérer la mémoire (destruction du tableau courant et création du nouveau tableau). Donner la déclaration et la définition de la fonction `reallouerTab`.

```
void reallouerTab(int** tab, int n, int newn){
  if (newn <= n) return;
  int* newTab=(int*)malloc(sizeof(int)*newn);
  int i;
  for(i=0;i<n;i++){
   newTab[i]=*tab[i];
  }
  free(*tab);
  *tab=newTab;
}
```

4 Miroir (3 points)

Donner la déclaration et la définition de la fonction `miroir` qui prend en paramètre une chaine de caractères et qui retourne cette chaine renversée. Par exemple `miroir("bonjour")` retourne la chaine de caractères "ruojnob". N'oubliez pas d'allouer en mémoire la nouvelle chaine de caractères.

```
char* miroir(char* ch){
   int taille;
  for (taille=0;ch[taille];taille++);
  char* res=malloc(taille+1);
  int i;
  for(i=0;i<taille;i++){
    res[i]=ch[taille-i-1];
  }
```

```
 res[taille]='\0';
  return res;
}
```

5 Liste (3 points)

On considère les structures de données suivantes permettant de définir une liste :

```
struct Elt {
  struct Elt* _next;
  int _i;
}
struct Liste {
  struct Elt* _premier;
}
```

Le suivant du dernier élément de la liste est NULL. Donner la définition des fonctions suivantes (attention aux listes vides)

1. `estVide(struct Liste* liste)` : retourne 1 si la liste est vide et 0 sinon

   ```
   int estVide(struct Liste* liste){
     return (liste->_premier == NULL);}
   ```

2. `afficher(struct Liste* liste)` : affiche les entiers contenus dans les éléments

   ```
   void afficher(struct Liste* liste){
     struct Elt* elt=liste->_premier;
     while (elt != NULL){
       printf("%d ",elt->_i);
       elt=elt->_next;
     }
   }
   ```

3. `ajouter(struct Liste* liste, int j)` : ajoute la valeur contenue dans j dans la liste

   ```
   void ajouter(struct Liste* liste, int j){
     struct Elt* elt=malloc(sizeof(struct Elt));
     elt->_i=j;
     elt->_next=liste->_premier;
     liste->_premier=elt;
   }
   ```

6 Map (3 points)

Écrire la déclaration et la définition d'une fonction `map` qui prend en paramètre un tableau de double *td* et une fonction $fntab$ et qui applique sur chaque élément du tableau cette fonction $fntab$. La fonction `map` ne retourne rien.

```
void map(int n,double* td,double (*fntab)(double)){
  int i;
  for(i=0;i<n;i++){
    td[i]=fntab(td[i]); /* fntab(td[i]); est accepté */
  }
}
```

Donner le code correspondant à l'appel de la fonction `map` pour le tableau t et la fonction dont la déclaration est `double fois2(double)`.

```
map(n,t,fois2);
```

7 Recherche dichotomique (3 points)

La libraire standard du C contient la fonction dont la documentation est la suivante :

```
#include <stdlib.h>
void *bsearch(const void *key, const void *buffer, size_t count, size_t size,
                        int (*compare)(const void *, const void *) );
```

Cette fonction parcourt un tableau à la recherche d'un élément égal à *key*. Le tableau commence à l'adresse $buffer$ et contient *count* éléments, chacun ayant une taille *size* exprimée en nombre d'octets. La fonction pointée par *compare* est utilisé pour la comparaison. Elle doit avoir la signature suivante `int funcname(void *k, void *value)`. Cette fonction est invoquée pendant la recherche avec *key* comme valeur pour k et une valeur du tableau pour *value*. La fonction doit se comporter comme suit. Si $k < value$ elle doit retourner une valeur strictement plus petite que 0. Si $k = value$ elle doit retourner 0. Si $k > value$ elle doit retourner une valeur strictement plus grande que 0. On considérera que le tableau est trié.

On considérera que *tab* est un tableau de n entiers triés, avec $tab[7] = 9$ et $tab[8] = 10$. Écrire une fonction de comparaison (pour le paramètre *compare*) puis le code qui appelle cette fonction `bsearch` afin de rechercher si la valeur 10 est présente dans *tab*.

```
int fncompare(const void* key, const void* val){
   int* k=(int*)key;
   int* value = (int*) val;
   int ret=1;
   if (*k < * val) ret=-1;
   if (*k == *val) ret=0;
   return ret;
}
int key=10;
bsearch(&key,tab,n,sizeof(int),fncompare);
```

Que retourne la fonction `bsearch` dans ce cas ?

```
La fonction retourne l'adresse de tab[8], car c'est le premier élement du tableau égal à 10.
```