Examen de SELC/INF104

1h30 – Sans document

Questions de cours (14 points)

Bases du langage C (7 points)

Question 1 (2 points) : En langage C, quel opérateur permet de connaître la taille (en octets) de l'espace mémoire occupé par une variable? Donnez un exemple d'utilisation. Le résultat fourni par l'utilisation de cet opérateur sera-t-il déterminé au moment de la compilation ou de l'exécution du programme ?

Question 2 (1,5 point) : Nommez la zone mémoire dans laquelle sont stockées les variables locales d'un programme. Quelle fonction permet de faire de l'allocation dynamique de mémoire en C ? Quelle fonction permet de libérer un espace mémoire alloué dynamiquement ?

Question 3 : On considère le code suivant, qui a servi à la production d'un exécutable appelé `count_char_in_word`.

```
1   #include <stdio.h>
2
3   int main(int argc, char * argv[]) {
4     if(argc!=3) {
5       printf("Wrong usage: expected ./count_char_in_word char word\n");
6       return -1;
7     }
8
9     int i=0, res=0;
10
11    char found=0;
12    while(argv[2][i]!='\0') {
13      if(argv[2][i]==argv[1][0]) {
14        found = 1;
15        break;
16      }
17      i++;
18    }
19    if(!found) {
20      printf("No occurence of char %c in %s\n",argv[1][0],argv[2]);
21      return 0;
22    }
23
24    i=0;
25    while(argv[2][i]!='\0') {
26      if(argv[2][i]==argv[1][0])
27        res++;
28      i++;
29    }
30    printf("Found %d occurence(s) of %c in %s\n", res, argv[1][0], argv[2]);
31    return 0;
32  }
```

Question 3.1 (1 point) : Qu'affiche le terminal après exécution du programme avec la ligne de commande suivante :

```
./count_char_in_word
```

Justifiez votre réponse.

Question 3.2 (1,5 point) : Qu'affiche le terminal après exécution du programme avec la ligne de commande suivante :

```
./count_char_in_word c cacao
```

Même question avec la ligne de commande suivante :

```
./count_char_in_word cacao cacao
```

Même question avec la ligne de commande suivante :

```
./count_char_in_word b cacao
```

Question 3.3 (1 point) : Quel est l'effet de l'instruction `break` ligne 15 ? Quel est l'effet de l'instruction `return` ligne 21 ?

Chaine de production (7 points)

Question 4 (2 points) : `gcc` permet de décomposer la production d'un exécutable à partir de fichiers sources en différentes étapes. Nommez et décrivez les entrées/sorties d'au moins quatre étapes réalisée par `gcc` lors de l'exécution de la commande `gcc fic.c -o appli`

On souhaite produire un exécutable à partir des fichiers suivants :

Contenu du fichier main.c	*Contenu du fichier globs.c*	*Contenu du fichier globs.h*

`int main(int argc, char* argv[]){`
`  print_state();`
`}`
	`#include <stdio.h>`
	`#include <globs.h>`
	
	`char state=4;`
	`void print_state()`
	`{`
	`  printf("State=%d",state);`
	`}`
		`void print_state();`

On exécute la commande suivante :

```
gcc –Wall –o appli main.c globs.c
```

Question 5 (1 point) : le terminal affiche le message suivant :

main.c:2:3: warning: implicit declaration of function 'print_state'

Expliquez pourquoi le compilateur produit ce warning, et que proposeriez-vous pour y remédiez ?

Question 6 (1 point) : dans la ligne de commande exécutée, à quoi sert l'option `–Wall` ? A quoi correspond l'option `–o` ?

Question 7 (2 points) : Comment réaliser la compilation séparée des fichiers `main.c` et `globs.c` ? Comment produire l'exécutable à partir des fichiers objets produits par la compilation séparée de `main.c` et `globs.c` ? Quelle est l'avantage de cette méthode par rapport à celle proposée dans l'énoncé ? Rappel de la méthode proposée : `gcc –Wall –o appli main.c globs.c`

Question 8 (1 point) : On souhaite utiliser la variable `state` dans la fonction `main` du fichier `main.c`. Modifiez le fichier `main.c` de sorte à pouvoir faire référence dans `main.c` à la variable state définie dans `globs.c` sans déclencher d'erreur de compilation.

Problème (6 points)

Il vous est ici demandé d'implémenter les fonctions de base d'un pokedex (vous savez probablement mieux que nous de quoi il s'agit, disons un index de pokemon). Nous vous fournissons le code de la fonction main, qui initialise le pokedex à partir d'informations données sur la ligne de commande. Voici quelques informations sur ce code :

Un pokemon est représenté par une structure de données `monster_t` constituée d'un identifiant unique et d'une catégorie (feu, glace, roche, dragon, etc.), tous deux encodés par un entier non signé. L'état du pockemon (vue, attrapé) est encodé en utilisant le type char.

Le programme fourni s'utilise en passant sur la ligne de commande le nombre de pokemons à insérer dans le pokedex, suivi d'une suite d'entiers correspondants aux identifiants des différents pokemon. La catégorie du pokemon et son état sont tirés aléatoirement (voir les lignes 32 et 34 du code fourni).

Nous donnons ici un exemple d'usage du programme pokedex_init obtenu après compilation du code ci-dessous : `./pokedex_init 5 21 48 12 125 4`

Après exécution du programme, nous souhaitons avoir en mémoire un pokedex dans lequel les pokemons sont triés par ordre croissant d'identifiants.

Puisque le nombre de pokemons est connu à l'exécution du programme, nous utiliserons une liste chainée pour implémenter le pokedex.

Question 9 (1,5 point) : proposez une implémentation des fonctions `create_monster` et `create_link`. La signature de ces fonctions est donnée dans le code C ci-dessous.

Question 10 (3 points) : proposez une implémentation de la fonction `insert_link`. La signature de ces fonctions est donnée dans le code C ci-dessous. Vous ferez attention aux cas particuliers suivants : (i) la liste est vide, cas où `beg` vaut `NULL`, (ii) le maillon à insérer, l, devient le nouveau début, (iii) le maillon l est à insérer entre deux maillons, et (iv) le maillon l est à insérer en fin de chaine. Nous rappelons ici que la liste chainée doit être triée par ordre croissant d'identifiant des monstres.

Question 11 (1,5 point) : proposez un nouveau type de donnée pour encoder la catégorie d'un pokemon, sachant que la catégorie d'un pokemon appartient à un ensemble d'identifiants prédéfini (e.g. water, fire, ice, plant, rock, psi, dragon). En plus de la définition du type de donnée, proposez une modification des lignes 7 et 32 du code ci-dessous. On s'assurera que la solution apportée n'introduit pas de consommation mémoire supplémentaire par rapport à l'occupation mémoire du programme initial.

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct monster_t
{
  unsigned int id;
  unsigned int category;
  char status;
} monster_t;

typedef struct link
{
  monster_t * monster;
  struct link * next;
} link_t;

monster_t * create_monster();
link_t * create_link(monster_t * m);
link_t * insert_monster(link_t * beg, link_t * l);

int main(int argc, char* argv[])
{
  unsigned int i=0, id=0, cat=0;
  int monster_nb = atoi(argv[1]);
  printf("Monster number %d\n", monster_nb);
  link_t * beg = NULL;
  for(i=0; i < monster_nb; i++) {
    monster_t * m = create_monster();
    printf("Create monster with ");
    m->id = atoi(argv[i+2]);
    printf("id=%d ", m->id);
    m->category = rand()%8;
    printf("category=%d ", m->category);
    m->status = rand()%2;
    printf("status=%d\n", m->status);
    link_t * l = create_link(m);
    beg = insert_monster(beg, l);
  }

  link_t * cur = beg;
  while(cur!=NULL) {
    printf("Monster %d\n", cur->monster->id);
    cur = cur->next;
  }
}
```