Algorithmique et langage C

Devoir surveillé

Durée : 2h

Documents autorisés : une feuille recto-verso A4 **manuscrite**.

Calculatrice autorisée.

Questions de cours (4 points)

1. Donner un exemple de pointeur :
   - vers un entier
   - générique, vers une variable
   - vers une fonction qui à comme argument d'entrée deux entiers, et qui renvoie un entier
   - vers une fonction qui renvoie un float et aux arguments d'entrée génériques
2. A quelle étape de la compilation le mot-clé #include intervient-il ? Quel est son effet ?
3. Quel est la différence entre une structure de type pile et une structure de type file ?

Exercice 1

En suivant le code suivant, indiquer les valeurs demandées au fur et à mesure de l'execution du code. On supposera les conditions habituelles d'adressage connexe et croissant. L'adressage des variables commence à 0x1000 et sizeof(int) = sizeof(int*) = 4.

```c
int a[10] = { 0,10,20,30,40,50,60,70,80,90 };
int* pt = &a[3];
//Question 1
void swap(int* a, int* b) { // on suppose la fonction suivante
  int temp = *a;
  *a = *b;
  *b = temp;
}

swap(pt, pt + 4);
//Question 2
int* b = (int*)calloc(6, 4);
for (int i = 0; i < 3; i++)
{
  b[i] = a[2 * i];
}
int* pt2 = &b;
//Question 3

float f = 10.5;
double d = 25;
return;
//Question 4
```

1. Donner &a, &pt, pt, *pt.
2. Donner toutes les valeurs du tableau a qui ont changé.
3. Donner les valeurs des tableaux a et b (quand elles sont connues), pt2, &pt2.
4. On suppose une fonction similaire à swap, de la forme void swap2(void *a, void *b). Que se passe-t-il lorsque l'on appelle swap2(&f, &d) ? Si le programme s'execute, quelles sont les nouvelles valeurs de f et d ? S'il plante, pourquoi ?

Exercice 2

On considère le code suivant qui est incomplet :

```
1  #include <stdlib.h>
2  #include <stdio.h>
3
4  void printLine(char c, int len){
5    for(int i =0; i<len; i++) { printf("%c", c); }
6    printf("\n");}
7
8  void printSomething(char* s, int le){
9    int compte = 0;
10   //???
11   for(int i=1; i<=le; i++)
12   {
13     compte++;
14     method(s[i-1], i);
15   }
16   return;}
```

1. Compléter la ligne 10 qui définis un pointeur générique method auquel on affecte l'adresse de printLine.
2. Qu'affiche printSomething("ISEN", 3) ?
3. On suppose une fonction **int printSquare (char c, int len)**. Quelle(s) ligne(s) faut-il changer pour l'utiliser dans pritnSomething plutôt que printLine ?
4. printSquare affiche un carré composé du caractère c, de taille de côté len. Rédiger la fonction en C ou en pseudo-code.

Exercice 3

Soit le code suivant définissant une structure d'arbre :

```c
typedef struct node_ {
  int value;
  struct node_* gauche;
  struct node_* droite;
} node;
typedef node* pNode;

pNode newNode(int a)
{
  pNode n = (pNode)malloc(sizeof(struct node_));
  n->value = a;
  n->droite = NULL;
  n->gauche = NULL;
  return n;}

pNode addNode(int value, pNode previous, char direction) {
//direction : 'l' left, 'r' right
  // ???
}

void  main() {
  pNode root = newNode(5);
  pNode deux = addNode(2, root, 'l');
  pNode zero = addNode(0, deux, 'l');
  pNode sept = addNode(7, root, 'r');
  }
```

1. Représenter l'arbre que ce programme chercher à créer, sur votre feuille. Indiquer la valeur de chaque node et les liens entre racine et feuilles.
2. Rédiger le code de la fonction addNode. Cette fonction permet d'ajouter une nouvelle feuille à un arbre déjà existant.
   On supposera que la fonction est toujours 'bien utilisée' (direction est toujours 'l' ou 'r', previous est toujours un pointeur non-NULL vers une node).
3. Proposer un algorithme en pseudo-code (ou en C) qui permet de free() un arbre entier sans créer de fuites de mémoire.

Exercice 4

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

int fonction1(int a) {
  if (a<2) {return 1;}
  if (a%2!=0) {return a*fonction1(a-1)*fonction1(a-2);}
  else {return fonction1(a-1);}
}

void fonction2(int * pt) {
  if(*pt>0) {
    *pt = fonction1(*pt);
    fonction2(pt+1);
  }
  return;
}

void main() {
  int tableau[12] = {7,6,5,4,3,2,1,0,-1,-2,-3,-4};
  fonction2(tableau);
  printf("tableau :\n");
  for(int i =0; i<10; i++) {printf("%d : %d", i, tableau[i]);}
}
```

1. Donner le résultat de fonction1(n), pour n allant de -4 à 7 (pour 7, une décomposition sous la forme a * b * c ... pourra suffir).
2. Proposer une modification de fonction1 pour utiliser une boucle plutôt qu'un appel par récurrence.
3. Qu'affiche le programme lorsqu'il est executé ?
4. En se reprenant le principe de récurrence de fonction2, proposer une fonction récurrente **void fonction3(char * s)**.
   Cette fonction décale toutes les lettres d'une chaîne de caratères d'un cran vers l'avant (a donne b, b donne c et ainsi de suite, et z deviens {).
   On pourra se servir du fait qu'une chaîne de caractères se termine par \0.