Algorithmique et langage C

Traveaux pratiques notés

Durée : 2h

Votre travail sera à rendre dans un fichier sous la forme **NOM_prenom_AP3_C_DS2.zip**.
Par exemple : GOSSWILLER_robin_AP3_C_DS2.zip .
Types de rendus possibles : .txt, .c, .h, .pdf, .png, .gif, .jpeg (et formats autres simples équivalents)

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. Pour chaque expression logique, donner son opposée. On simplifiera ces expressions autant que possible.
   - EXEMPLE : a < 10 || a > 10, son opposé est a == 10
   - a = 0
   - a < 3 || b > 5
   - (a = z && b < 4) || (a = z && b<8)
   - (a = z || b >20) && (a = z || b<12)

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.

On se place dans un système où sizeof(void *) = 8 (et non pas 4).

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

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

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.

Exercice 2

On considère le code suivant qui est incomplet :

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

void printOne(char c, int spot){
  printf("%c %d\n", c, spot);
}

void printLine(char c, int spot){
  for(int i =0; i<spot; i++) { printf("%c", c); }
  printf("\n");}
```

1. Soit une nouvelle fonction, printSquare, qui affiche un carré composé du caractère c, de taille de côté len. Rédiger la fonction en C.
2. On souhaite avoir une nouvelle fonction, printSomething, qui prends comme arguments une chaîne de caractères, sa taille, et un pointeur vers une des trois fonctions printOne, printLine et printSquare.
   Cette fonction affiche alors la chaîne de caractères en fonction de la position de la lettre dans la chaîne et en utilisant la fonction demandée.
   Exemples :
   printSomething("Junia", 5, &printOne);
   Affiche :
   J 0
   u 1
   n 2
   i 3
   a 4

   printSomething("ISEN", 4, &printLine);
   Affiche :
   I
   SS
   EEE
   NNNN

printSomething("AP3", 3, &printCube) ; Affiche :
A
PP
PP
333
333
333

3. Développer une fonction printSomething2 qui prends un nombre variable de pointeurs vers des fonctions plutôt qu'un seul, et cycle entre eux lors de l'affichage.
   Exemples :
   printSomething2("JUNIA", 5, &printOne, &printLine) ;
   Affiche :
   J 0
   UU
   N 2
   IIII
   A 4

   printSomething2("Hello", 3, &printOne, &printSquare, &printOne) ;
   Affiche :
   H 0
   ee
   ee
   l 2

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/sur un logiciel de dessin de votre choix. 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. Détailler le calcul de fonction1(7). On examinera pas à pas les différents appels récursifs.
2. Proposer une modification de fonction1 pour utiliser une boucle plutôt qu'un appel par récurrence.
3. 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 caractères d'un cran vers l'avant (a donne b, b donne c et ainsi de suite, et z deviens a).
   On pourra se servir du fait qu'une chaîne de caractères se termine par \0.