Durée : 1h30

AUCUN DOCUMENT AUTORISE -- PAS DE CALCULATRICE

Lisez attentivement l'énoncé et respectez les consignes de chaque exercice. Ecrivez lisiblement, indentez et commentez vos codes.
Vous trouverez en Annexes les headers des librairies *pile*, *file* et *tas_max* (file de priorité).
Vous ne devez pas coder les sous-programmes de ces librairies mais juste les appeler correctement quand cela est nécessaire.

Exercice 1. Questions de cours/TPs (files, piles) [6 points]

Pour répondre aux questions, cochez ou remplissez les encadrés.

1. (1 point) Pour chaque type de conteneur (Pile ou File), indiquer le principe sur lequel il repose (A, B, C ou D) :

   Pile : ☐
   File : ☐

   A. Premier entré, premier sorti
   B. Le plus petit sortira en premier
   C. Dernier entré, premier sorti
   D. Celui qui sortira est choisi aléatoirement

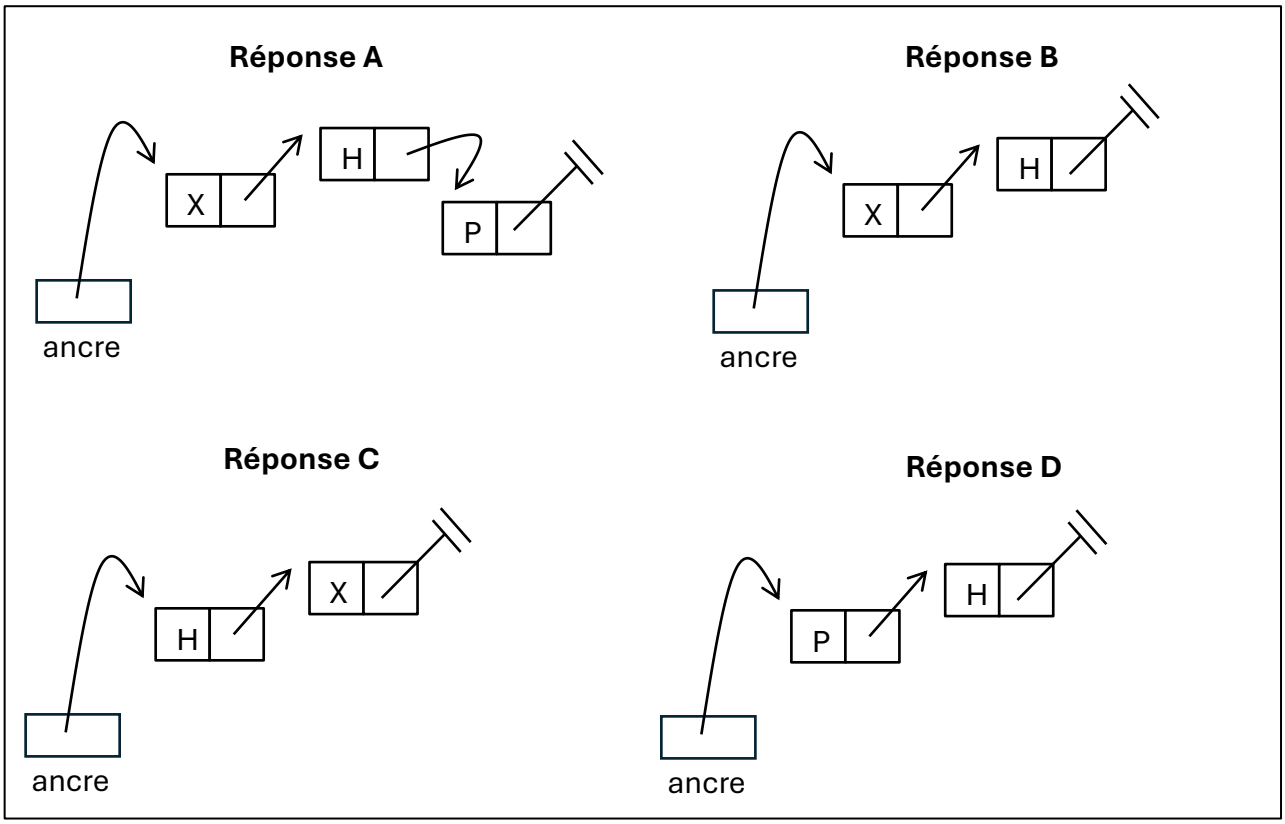
2. (1,5 points) Soit F une file d'entiers implémentée par un tableau automatique circulaire de taille 5. On a enfilé les entiers 12 puis 7 puis 23 ; puis défilé 2 éléments ; puis enfilé les entiers 14 puis 8 puis 35.
   a. Indiquer l'état de la file en complétant les cases du tableau ci-dessous (indiquer quelles valeurs elles contiennent)

indices	0	1	2	3	4
valeurs					

   b. Quel sera le prochain élément défilé ? ☐

3. (1 point) Pour implémenter une file par liste chainée, on utilise une liste simplement chainée avec deux ancres qui pointent sur le premier et le dernier maillon. Indiquer où seront enfilés et défilés les prochains éléments (cocher la bonne réponse) :
   - Le prochain élément sera enfilé en
     ☐ Début de liste (il deviendra le premier maillon)
     ☐ Fin de liste (il deviendra le dernier maillon)
   - Le prochain élément défilé sera celui en
     ☐ Début de liste ☐ Fin de liste

4. (1 point) Soit P une pile de caractères implémentée avec une liste simplement chainée à une seule ancre. On a empilé les caractères X puis H puis P ; puis dépilé un élément.
   Quel est le schéma correct de la pile (A, B, C ou D) ? ☐



5. (1,5 point) Quel est l'algorithme correct (A, B, C ou D) pour inverser une file ? ☐

exemple :
file de départ : 12 8 17 125 4 → file inversée : 4 125 17 8 12

Réponse A

- Déclarer et initialiser une pile vide P
- Tant que F n'est pas vide :
  Défiler le prochain élément de F et l'empiler dans P
- Tant que P n'est pas vide :
  Dépiler le prochain élément de P et l'enfiler dans F

Réponse B

- Déclarer et initialiser une file vide F2
- Tant que F n'est pas vide :
  Défiler le prochain élément de F et l'enfiler dans F2
- Tant que F2 n'est pas vide :
  Défiler le prochain élément de F2 et l'enfiler dans F

Réponse C

- Déclarer et initialiser une pile vide P
- Tant que F n'est pas vide :
  Défiler le prochain élément de F et l'empiler dans P
  Dépiler le prochain élément de P et l'enfiler dans F

Réponse D

Ce n'est pas possible, on ne peut pas inverser une file.

Exercice 2. Compréhension de code. Utilisation d'une pile. [3 points]

La fonction codée ci-dessous (dans l'encadré) utilise la librairie ***pile*** donnée en Annexe. Elle utilise également les deux sous-programmes décrits ci-dessous que vous ne devez pas coder :

1. Prototype : ***int estChiffre(char c);***
   Rôle : retourne 1 si le caractère reçu en paramètre est un chiffre (caractères de '0' à '9'), 0 sinon.
2. Prototype : ***int estOperateur(char c);***
   Rôle : retourne 1 si le caractère reçu en paramètre est un opérateur (caractères '+', '-', '*' ou '/'), 0 sinon.

```
char* toPost(char *expr) {
    char *exprP;
    int i;
    int j=0;
    int t=strlen(expr);
    t_pile p;
    //allocation d'une chaine dynamique
    //de deux fois la taille de celle reçue en paramètre
    exprP=(char*)malloc((t+1)*sizeof(char));
    //initialisation d'une pile vide
    init_pile(&p);
    //Parcours de la chaine reçue caractère par caractère
    for (i=0;i<t;i++) {
        //si le caractère courant est un chiffre
        if (estChiffre(expr[i])==1) {
            exprP[j]=expr[i];
            j++;
        }
        else
            //si le caractère courant est un opérateur (+, -, * ou /)
            if (estOperateur(expr[i])==1){
                    empiler(&p,&(expr[i]));
                    exprP[j]=' ';
                    j++;
            }
            else
                //si le caractère courant est une parenthèse fermante
                if (expr[i]==')') {
                    exprP[j]=' ';
                    exprP[j+1]=*((char*)depiler(&p));
                    j+=2;
                }
    }
    exprP[j]='\0';
    exprP=(char*)realloc(exprP,(strlen(exprP)+1)*sizeof(char));
    return exprP;
}
```

Quelle chaine retourne la fonction si un programme l'appelle en lui envoyant en paramètre la chaine «(14*((9-3)+(817/2)))» ? Indiquer la réponse en remplissant le tableau ci-dessous (1 caractère par case) :

indices	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
caractères																					\0

Exercice 3. Structures, files et files de priorité. [11 points]

Gestion des patients aux urgences.

Pour chaque patient on note son numéro de sécurité sociale (chaine de 13 caractères), son nom (chaine dynamique) ; et une priorité lui sera attribuée (entier). A leur arrivée, les patients sont enregistrés dans une file d'attente. Les patients sont ensuite vus par un médecin qui leur attribue une priorité. Ils sortent alors de la file d'attente et sont insérés dans une file de priorité.

1. **(1 pt) Définir un type structuré *t_patient* pour stocker les informations sur un patient.**
2. **(2 pts) Ecrire un sous-programme nommé *initPatient* qui remplit une structure de type *t_patient* dont l'adresse est reçue en paramètre. Le numéro de sécurité sociale et le nom sont saisis au clavier ; la priorité est initialisée à 0. Attention : le champ nom est dynamique et doit être alloué !**
3. **(1 pt) Ecrire un sous-programme nommé *saisirPriorité* qui permet à un médecin de saisir la priorité d'un patient. Le sous-programme reçoit en paramètre l'adresse d'une structure de type *t_patient* et demande de saisir la priorité qui doit être un entier compris entre 1 et 4 (à blinder avec une boucle).**
4. **(2 pts) Ecrire un sous-programme nommé *gererPatient* qui retire le prochain patient de la file d'attente, demande au médecin de saisir la priorité (en appelant le sous-programme *saisirPriorité*) et l'insère dans la file de priorité (un tas). Ce sous-programme reçoit en paramètres l'adresse de la file d'attente et l'adresse du tas. (3 pts)**
5. **(5 pts) Compléter le *main()* ci-dessous qui simule la gestion des patients aux urgences :**

   Les lignes à compléter sont les lignes numéros 88, 90, 97, 101, 107, 111, 113, 115 et 117. Ecrire le code manquant dans les encadrés en appelant si nécessaire les sous-programmes des questions ci-dessus ou les sous-programmes des différentes librairies fournies en Annexes.

```
82  int main()
83  {
84      //déclaration d'une file et d'un tas
85      t_file f;
86      t_tas t;
87      //initialiser la file
88      
89      //initialiser le tas avec un nombre de niveaux égal à 5
90      
91      //déclaration de variables
92      t_patient *p;
93      int i;
94      //insertion de 10 patients dans la file d'attente
95      for(i=0;i<10;i++){
96          //allouer une structure patient
97          p= ;
98          //saisie des informations sur le patient
99          saisirPatient(p);
100         //enfiler le patient dans la file d'attente
101         
102     }
103     printf("\n");
104     //tant que la file n'est pas vide
105     while(fileVide(&f)!=1){
106         //gérer un patient
107         gererPatient( , );
108     }
109     //libérer correctement la mémoire
110     //tant que le tas n'est pas vide
111     while( ){
112         //supprimer le prochain patient du tas
113         p= ;
114         //libérer la chaine pour le nom du patient
115         
116         //libérer le patient
117         
118     }
119     return 0;
120 }
```

Annexe 1 : librairie *file*

```c
#ifndef FILE_H_INCLUDED
#define FILE_H_INCLUDED
///type pour les maillons
typedef struct maillonF{
    void *data; //pointeur générique sur la donnée
    struct maillonF *suivant;  //pointeur sur le maillon suivant
}t_maillonF;

///type pour la file
typedef struct file{
    t_maillonF *tete;//ancre de tête : pointeur sur le prochain maillon à défiler
    t_maillonF *fin;//ancre de fin : pointeur sur le dernier maillon (après lequel enfiler)
}t_file;

///prototypes des sous-programmes

/* initialisation d'une file vide
paramètre : l'adresse de la file */
void init_file(t_file*f);

/* test si la file est vide
paramètre : l'adresse de la file
retour : 1 si la file est vide, 0 sinon */
int fileVide(t_file*f);

/* compter le nombre d'éléments
paramètre : l'adresse de la file
retour : taille de la file */
int tailleF(t_file*f);

/* enfiler une nouvelle donnée
paramètres : l'adresse de la file, l'adresse de la donnée */
void enfiler(t_file*f,void*data);

/* défiler le prochain élement
paramètre : l'adresse de la file
retour : l'adresse de la donnee*/
void* defiler(t_file*f);

/* acceder au prochain element (donnée en tête)
paramètre : l'adresse de la file
retour : adresse de la donnée en tête */
void* tete(t_file*f);

#endif // FILE_H_INCLUDED
```

Annexe 2 : librairie *pile*

```c
#ifndef PILE_H_INCLUDED
#define PILE_H_INCLUDED

///type pour les maillons
typedef struct maillonP{
    void *data; //pointeur générique sur la donnée
    struct maillonP *suivant;  //pointeur sur le maillon suivant
}t_maillonP;

///type pour la pile
typedef struct pile{
    t_maillonP *sommet;//ancre de tête : pointeur sur le sommet de la pile
    int taille; //taille de la pile
}t_pile;

///prototypes des sous-programmes
/* initialisation d'une pile vide
paramètre : l'adresse de la pile */
void init_pile(t_pile*p);

/* test si la pile est vide
paramètre : l'adresse de la pile
retour : 1 si la pile est vide, 0 sinon */
char pileVide(t_pile*f);

/* compter le nombre d'éléments
paramètre : l'adresse de la pile
retour : taille de la pile */
int tailleP(t_pile*f);

/* empiler une nouvelle donnée
paramètres : l'adresse de la pile, l'adresse de la donnée */
void empiler(t_pile*f,void*data);

/* dépiler le prochain élement
paramètre : l'adresse de la pile
retour : l'adresse de la donnée*/
void* depiler(t_pile*f);

/* acceder au prochain élément (donnée en tête)
paramètre : l'adresse de la pile
retour : adresse de la donnée au sommet de la pile */
void* sommet(t_pile*f);

#endif // FILE_H_INCLUDED
```

Annexe 3 : librairie *tas_max*

```c
#ifndef TAS_MAX_H_INCLUDED
#define TAS_MAX_H_INCLUDED

//type pour les noeuds
typedef struct noeud{
    int cle;
    void*data;
}t_noeud;
//type pour le tas
typedef struct tas{
    t_noeud*tab;
    int tailleM;
    int n;
} t_tas;

//prototypes des sous-programmes
/*initialisation d'un tas vide
parametres : l'adresse du tas, son nombre de niveau de départ (>=1)*/
void init_tas(t_tas*t,int tM);

/*test si le tas est vide
parametres : l'adresse du tas
retour : 1 si vide, 0 sinon */
int tasVide(t_tas*t);

/*ajouter un element
parametres : l'adresse du tas, le noeud à ajouter (le sous-programme en
fait une copie)*/
void ajouter_au_tas(t_tas*t, t_noeud nouveau);

/*supprimer la racine du tas (le noeud ayant la plus grande clé
parametres : l'adresse du tas
retour : l'adresse de la donnée du noeud supprimé*/
void* supprimer_du_tas(t_tas*t);

/*accéder à l'élément de plus haute priorité
parametres : l'adresse du tas
retour : l'adresse de la donnée du noeud racine*/
void* consulter(t_tas*t);

#endif // TAS_MAX_H_INCLUDED
```