TP d’Algorithmique

n°3

Table des matières

 Programme principal 2

Travail à faire 2

Algorithme 2

Programme 3

Jeux d’essai 5

 Initialiser une file d’attente 10

Travail à faire 10

Algorithme 10

Programme 10

 Désinitialiser une file d’attente 10

Travail à faire 10

Algorithme 10

Programme 11

 Ajouter une personne en queue de file d’attente 11

Travail à faire 11

Algorithme 11

Programme 12

 Retirer la personne en tête de file 12

Travail à faire 12

Algorithme 12

Programme 13

 Consulter les informations de la personne qui se trouve en tête 14

Travail à faire 14

Algorithme 14

Programme 14

 Calculer la longueur de la file d’attente 14

Travail à faire 14

Algorithme 14

Programme 15

 Fonctions 16

Contenu de file.cpp 16

Contenu de file.h 18

***Exercice : Module de gestion de file d’attente***

*Rappel de l’énoncé*

Nous nous proposons de gérer une file d’attente de personnes en utilisant un chaînage en mémoire. Nous créons un module de gestion FileAttente comprenant les opérations :

Initialiser une file d’attente

Désinitialiser une file d’attente

Ajouter une personne en queue de file d’attente

Retirer la personne en tête de la file

Consulter le numéro de la personne qui se trouve en tête

Calculer la longueur de la file d’attente

# Programme principal

## Travail à faire

Nous affichons une liste de choix proposés à l’utilisateur pour gérer une file d’attente. Selon le choix qu’il fait, nous appelons les différentes fonctions en demandant les paramètres à l’utilisateur.

## Algorithme

//Programme principal

**Utilise** FileDAttente

**Programme** fileAttente

**Début**

**Avec** choix : **Entier**

p\_maillon : **Pointeur** maillon

maillon\_temp : maillon

numero : **Entier**

nom : **Chaîne**

f : FileAttente

continuer : **Booléen**

init(f)

continuer <- **Vrai**

**Tant** **que** continuer **Faire**

**Afficher** "File d'attente"

**Afficher** "1. Ajouter une personne en queue"

**Afficher** "2. Retirer la personne en tête"

**Afficher** "3. Consulter la personne en tête de file"

**Afficher** "4. Calculer la longueur de la file d'attente"

**Afficher** "5. Désinitialiser la liste"

**Afficher** "6. QUITTER"

**Afficher** "Saisir choix : "

**Saisir** choix

**Selon** choix **Faire**

**Cas** 1:

**Afficher** "Entrer le numéro de la personne : "

**Saisir** numero

**Afficher** "Entrer le nom de la personne : "

**Saisir** nom

p\_maillon <- **Nouveau** maillon

(**Contenu** p\_maillon).numero <- numero

(**Contenu** p\_maillon).nom <- nom

ajoutEnQueue(f, p\_maillon)

**Cas** 2:

supprEnTete(f)

**Cas** 3:

**Si** calculerLongueur(f) > 0 **Alors**

maillon\_temp <- consulterTete(f)

**Afficher** "Le numéro de la personne est : "

**Afficher** maillon\_temp.numero

**Afficher** "Son nom est : "

**Afficher** maillon\_temp.nom

**Sinon**

**Afficher** "La file est vide, il n'y a pas de tête"

**Fin** **si**

**Cas** 4:

**Afficher** "Longueur de la file d'attente : "

**Afficher** calculerLongueur(f)

**Cas** 5:

desInit(f)

**Afficher** "Liste désinitialisée"

**Cas** 6:

continuer <- **Faux**

**Défaut**:

**Afficher** "Votre choix n'est pas valide"

**Fin** **faire**

**Fin** **faire**

**Fin** fileAttente

## Programme

|  |  |
| --- | --- |
| 1  2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77  78  79  80 | //  // main.cpp  // TP03  //  // Created by Oncle D. on 03/10/2014.  // Copyright (c) 2014 Oncle D. All rights reserved.  //  #include <iostream>  #include <string.h>  #include "file.h"  using namespace std;  int main(int argc, const char \* argv[]) {    int choix;  struct maillon \*p\_maillon;  struct maillon maillon\_temp;  int numero;  char nom[80];  struct FileAttente f;  init(f);  bool continuer=true;    while (continuer) {    cout << "FILE D'ATTENTE" << endl << endl;  cout << "1. Ajouter une personne en queue" << endl;  cout << "2. Retirer la personne en tête" << endl;  cout << "3. Consulter la personne en tête de file" << endl;  cout << "4. Calculer la longueur de la file d'attente" << endl;  cout << "5. Désinitialiser la liste" << endl << endl;    cout << "6. QUITTER" << endl;    cout << "Saisir choix : ";  cin >> choix;  switch (choix) {  case 1:  cout << "Entrez le numéro de la personne : ";  cin >> numero;  cout << "Entrez son nom : ";  cin >> nom;  p\_maillon = new maillon;  (\*p\_maillon).numero = numero;  strcpy((\*p\_maillon).nom, nom);  (\*p\_maillon).suivant = NULL;  ajoutEnQueue(f, p\_maillon);  break;  case 2:  supprEnTete(f);  break;  case 3:  if (calculerLongueur(f) > 0) {  maillon\_temp = consulterTete(f);  cout << "Le numéro de la personne est : " << maillon\_temp.numero << endl;  cout << "Son nom est : " << maillon\_temp.nom << endl;  }  else  cout << "La liste est vide, il n'y a pas de tête." << endl;  break;  case 4:  cout << "Longueur de la file d'attente : " << calculerLongueur(f) << endl;  break;  case 5:  desInit(f);  cout << "Liste désinitialisée." << endl;  break;  case 6:  continuer=false;  break;  default:  cout << "Votre choix n'est pas valide" << endl;  break;  }  }  return 0;  } |

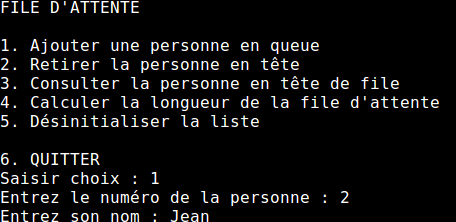
## Jeux d’essai

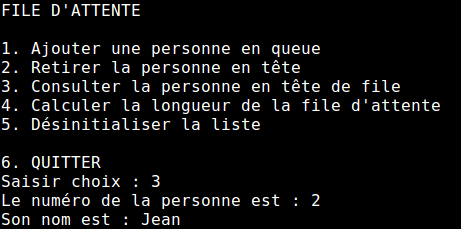
|  |  |  |
| --- | --- | --- |
| **Cas testé** | **Entrée** | **Résultat** |
| Ajout d’une personne en queue | 1 puis 2, Jean | Personne ajoutée |
| Voir la personne en tête | 3 | 3, Jean |
| Ajout d’une autre personne en queue | 1 puis 5, Marc | Personne ajoutée |
| Voir la personne en tête | 3 | 3, Jean |
| Supprimer la personne en tête | 2 | Personne supprimée |
| Voir la personne en tête | 3 | 5, Marc |
| Calculer la longueur de la file | 4 | 1 |
| Désinitialiser la file | 5 | File désinitialisée |
| Calculer la longueur de la file | 4 | 0 |
| Voir la personne en tête | 3 | Aucune personne en tête |
| Supprimer la personne en tête | 2 | Personne supprimée |

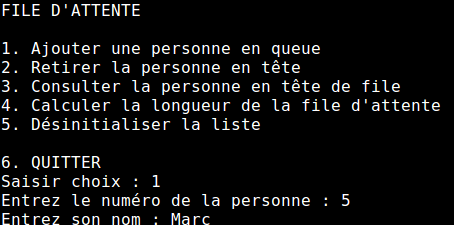
Les jeux d'essai sont exécutés en continuité.

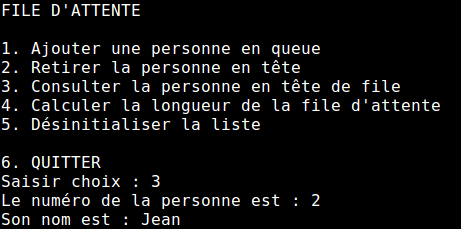
On remarque que le programme ne plante pas lorsqu'on essaie de supprimer une personne n'existant pas.

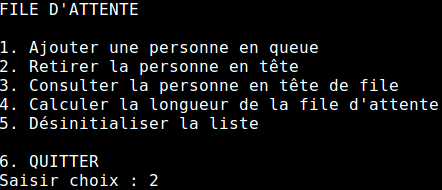
De même pour regarder la tête alors que la file est vide.

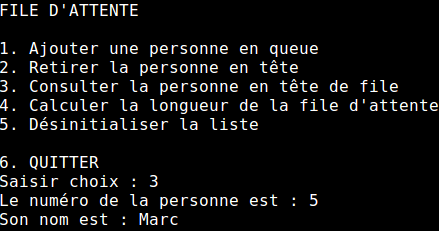


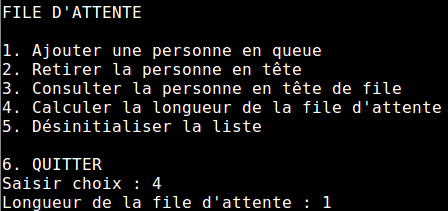


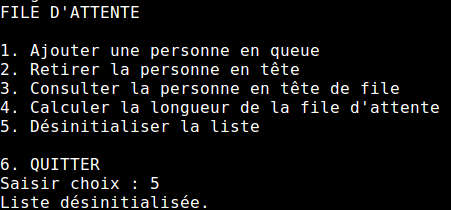


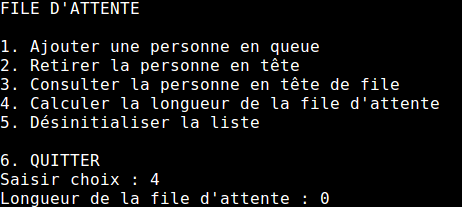


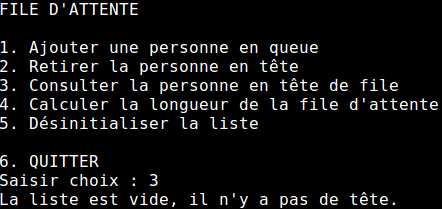


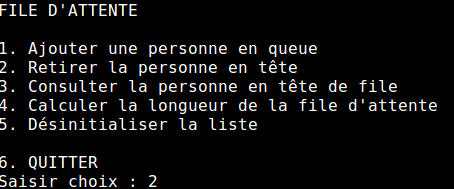












# Initialiser une file d’attente

## Travail à faire

Initialiser à NULL la tête et la queue d’une file d’attente.

## Algorithme

//Initialisation d'une file d'attente

//Paramètre de sortie : Un file d'attente

//Post-condition : La file d'attente est initialisée

**Procédure** init(; f : FileAttente)

**Début**

f.tete <- **NULL**

f.queue <- **NULL**

**Fin** init

## Programme

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | //Initialisation d'une file d'attente  //Paramètre de sortie : Un file d'attente  //Post-condition : La file d'attente est initialisée  void init(struct FileAttente &f)  {  f.debut=NULL;  f.fin=NULL;  } |

# Désinitialiser une file d’attente

## Travail à faire

Nous supprimons tous les éléments de la file d’attente puis réinitialisons la tête et la queue de la file à NULL.

## Algorithme

// Procédure permettant de désinitialiser une liste d'attente

// Paramètre d'entrée : une liste d'attente

// Paramètre de sortie : une liste d'attente

// Pré-condition : la liste est initialisée

// Post-condition : la liste est vidée et désinitialisée

**Procédure** desInit(f : FileAttente; f : FileAttente)

**Début**

**Avec** tmp : **Pointeur** maillon

tmp <- f.tete

**Tant** **que** tmp.suivant != **NULL** **Faire**

tmp <- tmp.suivant

supprEnTete(f ; f)

**Fin** **Faire**

f.tete <- **NULL**

f.queue <- **NULL**

**Fin** desInit

## Programme

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | // Procédure permettant de désinitialiser une liste d'attente  // Paramètre d'entrée : une liste d'attente  // Paramètre de sortie : une liste d'attente  // Pré-condition : la liste est initialisée  // Post-condition : la liste est vidée et désinitialisée  void desInit(struct FileAttente &f)  {  struct maillon \*tmp;  tmp = f.debut;    while ((\*tmp).suivant != NULL) {  tmp = (\*tmp).suivant;  supprEnTete(f);  }    f.debut = NULL;  f.fin = NULL;  } |

# Ajouter une personne en queue de file d’attente

## Travail à faire

Nous testons si si la file est vide ou non : si elle ne l’est pas, nous ajoutons un lien depuis le dernier maillon vers la nouvelle personne et nous remettons la queue de la file d’attente sur cette personne. Si elle l’est, nous mettons la tête et la queue de la file d’attente sur la nouvelle personne.

## Algorithme

//Procédure permettant l'ajout d'un élément en queue de file

//Paramètres d'entrée : Une file d'attente

// Un pointeur de maillon

//Paramètres de sortie : Une file d'attente ayant un élément en plus

//Pré-conditions : La file est initialisée

//Post-conditions : La fil a l'élément passé en paramètre en queue

**Procédure** ajoutenqueue( f: fileattente, p: **Pointeur** maillon ; f: fileattente)

**Début**

**Si** (f.queue) ≠ **NULL** **Alors**

(**Contenu** (f.queue)).suivant <- p

(f.queue) <- p

**Sinon**

(f.queue) <- p

(f.tete) <- p

**Finsi**

**Fin** ajoutenqueue

## Programme

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 9  10  11  12  13  14  15  16  17  18  19 | //Procédure permettant l'ajout d'un élément en queue de file  //Paramètres d'entrée : Une file d'attente  // Un pointeur de maillon  //Paramètres de sortie : Une file d'attente ayant un élément en plus  //Pré-conditions : La file est initialisée  //Post-conditions : La fil a l'élément passé en paramètre en queue  void ajoutEnQueue( struct FileAttente &f, struct maillon \*p)  {  if ((f.fin) != NULL)  {  (\*(f.fin)).suivant=p;  (f.fin) = p;  }  else  {  f.fin = p;  f.debut = p;  }  } |

# Retirer la personne en tête de file

## Travail à faire

Nous testons s’il y a au moins un élément. S’il n’y en a pas, nous ne faisons rien. Sinon, nous supprimons le maillon de tête et faisons pointer la tête de file vers le suivant.

## Algorithme

// Procédure permettant de retirer l'élément en tête de file

// Paramètres d'entrée : Une file d'attente

// Paramètre de sortie : Une file d'attente sans l'élément de tête

// Pré-condition : file initialisée

// Post-condition : L'élément de tête a été supprimé

**Procédure** supprEnTete( f : FileAttente ; f : FileAttente)

**Début**

**Avec** tmp : **Pointeur** maillon

tmp <- f.tete

**Si** tmp ≠ **NULL** **Alors**

tmp <- (\*tmp).suivant

**Détruire** f.tete

f.tete <- tmp

**Fin** **si**

**Fin** supprEnTete

## Programme

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 | // Procédure permettant de retirer l'élément en tête de file  // Paramètres d'entrée : Une file d'attente  // Paramètre de sortie : Une file d'attente sans l'élément de tête  // Pré-condition : file initialisée  // Post-condition : L'élément de tête a été supprimé  void supprEnTete(struct FileAttente &f)  {  struct maillon \*tmp;  tmp = f.debut;    if (tmp != NULL)  {  tmp = (\*tmp).suivant;  delete f.debut;  f.debut = tmp;  }  } |

# Consulter les informations de la personne qui se trouve en tête

## Travail à faire

Retourner les informations contenues dans la tête de la liste passée en paramètre.

## Algorithme

//Fonction permettant de consulter la personne en tête de file

//Paramètres d'entrée : Une file d'attente

//Valeur retournée : La personne en tête

//Pré-conditions : file initialisée

// file ayant une tête (l > 0)

**Fonction** consulterTete( f: FileAttente) **Retourne** maillon

**Début**

**Retourne** **Contenu**(f.tete)

**Fin** consulterTete

## Programme

|  |  |
| --- | --- |
| 1  2 3 4 5 6 7 8 9 | //Fonction permettant de consulter la personne en tête de file  //Paramètres d'entrée : Une file d'attente  //Valeur retournée : La personne en tête  //Pré-conditions : file initialisée  // file ayant une tête (l > 0)  struct maillon consulterTete( struct FileAttente f)  {  return \*(f.debut);  } |

# Calculer la longueur de la file d’attente

## Travail à faire

Nous parcourons la file d’attente de la tête jusqu’à la queue en incrémentant un compteur.

## Algorithme

//Fonction permettant de calculer la longueur de la file d'attente

//Paramètres d'entrée : une file d'attente

//Valeur retournée : la taille de la file

//Pré-conditions : la file est initialisée

**Fonction** calculerLongueur(f : FileAttente) **Retourne** **Entier**

**Début**

**Avec** n : **Entier**

p : **Pointeur** maillon

n <- 0

p <- f.tete

**TantQue** p != **NULL** **Faire**

n <- n + 1

p <- p.suivant

**Fin** **faire**

**Retourne** n

**Fin** calculerLongueur

## Programme

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9  10 11 12 13 14 15 16 17 18 19 20 | //Fonction permettant de calculer la longueur de la file d'attente  //Paramètres d'entrée : une file d'attente  //Valeur retournée : la taille de la file  //Pré-conditions : la file est initialisée  int calculerLongueur(struct FileAttente f)  {  int n;  struct maillon \*p;  n = 0;  p = f.debut;    while (p != NULL)  {  n++;  p = (\*p).suivant;  }    return n;  } |

# Fonctions

## Contenu de file.cpp

//

// file.cpp

// TP03

//

// Created by Oncle D. on 03/10/2014.

// Copyright (c) 2014 Oncle D. All rights reserved.

//

#include "file.h"

//Initialisation d'une file d'attente

//Paramètre de sortie : Un file d'attente

//Post-condition : La file d'attente est initialisée

void init(struct FileAttente &f)

{

f.debut=NULL;

f.fin=NULL;

}

// Procédure permettant de désinitialiser une liste d'attente

// Paramètre d'entrée : une liste d'attente

// Paramètre de sortie : une liste d'attente

// Pré-condition : la liste est initialisée

// Post-condition : la liste est vidée et désinitialisée

void desInit(struct FileAttente &f)

{

struct maillon \*tmp;

tmp = f.debut;

// while ((\*tmp).suivant != NULL) {

// tmp = (\*tmp).suivant;

// supprEnTete(f);

// }

while (f.debut != NULL) {

supprEnTete(f);

}

f.debut = NULL;

f.fin = NULL;

}

//Procédure permettant l'ajout d'un élément en queue de file

//Paramètres d'entrée : Une file d'attente

// Un pointeur de maillon

//Paramètres de sortie : Une file d'attente ayant un élément en plus

//Pré-conditions : La file est initialisée

//Post-conditions : La file a l'élément passé en paramètre en queue

void ajoutEnQueue( struct FileAttente &f, struct maillon \*p)

{

if (f.fin != NULL)

{

(\*(f.fin)).suivant=p;

(f.fin) = p;

}

else

{

f.fin = p;

f.debut = p;

}

}

// Procédure permettant de retirer l'élément en tête de file

// Paramètres d'entrée : Une file d'attente

// Paramètre de sortie : Une file d'attente sans l'élément de tête

// Pré-condition : file initialisée

// Post-condition : L'élément de tête a été supprimé

void supprEnTete(struct FileAttente &f)

{

struct maillon \*tmp;

tmp = f.debut;

if (tmp != NULL)

{

tmp = (\*tmp).suivant;

if (f.debut == f.fin) { // Il n'y a qu'un élément

delete f.debut;

f.debut = NULL;

f.fin = NULL;

}

else {

delete f.debut;

f.debut = tmp;

}

}

}

//Fonction permettant de consulter la personne en tête de file

//Paramètres d'entrée : Une file d'attente

//Valeur retournée : La personne en tête

//Pré-conditions : file initialisée

// file ayant une tête (l > 0)

struct maillon consulterTete( struct FileAttente f)

{

return \*(f.debut);

}

//Fonction permettant de calculer la longueur de la file d'attente

//Paramètres d'entrée : une file d'attente

//Valeur retournée : la taille de la file

//Pré-conditions : la file est initialisée

int calculerLongueur(struct FileAttente f)

{

int n;

struct maillon \*p;

n = 0;

p = f.debut;

while (p != NULL)

{

n++;

p = (\*p).suivant;

}

return n;

}

## Contenu de file.h

//

// file.h

// TP03

//

// Created by Oncle D. on 03/10/2014.

// Copyright (c) 2014 Oncle D. All rights reserved.

//

#ifndef \_\_TP03\_\_file\_\_

#define \_\_TP03\_\_file\_\_

#include <stdio.h>

struct maillon {

int numero;

struct maillon \*suivant;

char nom[80];

};

struct FileAttente {

struct maillon \*debut;

struct maillon \*fin;

};

void init(struct FileAttente &f);

void desInit(struct FileAttente &f);

void ajoutEnQueue( struct FileAttente &f, struct maillon \*p);

void supprEnTete( struct FileAttente &f);

struct maillon consulterTete( struct FileAttente f);

int calculerLongueur( struct FileAttente f);

#endif /\* defined(\_\_TP03\_\_file\_\_) \*/