TP d’algorithmique

n°5

Création du module de liste circulaire 2

Rappel de l’énoncé 2

Analyse du travail à faire 2

Initialiser la liste 2

Désinitialiser la liste 2

Supprimer un élément de la liste 2

Afficher la liste 2

Faire une rotation de la liste 2

Insérer un nouveau maillon 2

Compter ne nombre de maillon de la liste 2

Algorithme 3

Programme 6

Contenu de listeCirculaire.h 8

Algorithme principal 10

Analyse du travail à faire 10

Algorithme 10

Programme 12

Jeux d’essai 13

# Création du module de liste circulaire

## Rappel de l’énoncé

Programmer le module de gestion d’une liste circulaire constituée de maillons. Ce module devra contenir les procédures d’affichage de la liste, de rotation de la liste, d’insertion d’un maillon ainsi que la fonction retournant le nombre de maillon de la liste.

## Analyse du travail à faire

Démarrer un module listeCirculaire contenant le type maillon, les fonctions et procédures précisées dans l’énoncé et deux procédures d’initialisation et de désinitialisation.

### Initialiser la liste

Nous faisons pointer le pointeur sur la liste circulaire vers NULL.

### Désinitialiser la liste

Nous supprimons tous les éléments de la liste circulaire puis faisons pointer son pointeur vers NULL.

### Supprimer un élément de la liste

Nous testons si le pointeur de la liste est NULL, s’il l’est nous ne faisons rien.

Si la liste contient au moins deux éléments, nous copions le deuxième élément dans le premier puis détruisons le deuxième. Si la liste ne contient qu’un seul élément, on le supprime et faisons pointer le pointeur de la liste vers NULL.

### Afficher la liste

Nous créons un pointeur temporaire qui parcourt la liste en affichant le contenu de chaque maillon jusqu’à ce qu’il revienne à son point de départ : le pointeur de la liste circulaire.

### Faire une rotation de la liste

Nous testons si le pointeur de la liste est NULL, s’il l’est nous ne faisons rien.

Sinon, nous faisons pointer le pointeur de la liste vers l’élément suivant.

### Insérer un nouveau maillon

Nous créons un pointeur pointant vers le maillon à insérer.

Nous testons si le pointeur de la liste circulaire est NULL. S’il ne l’est pas, nous faisons pointer le nouveau maillon vers le deuxième maillon de la liste et le premier maillon vers le maillon à insérer. Nous échangeons ensuite les contenus du premier maillon et du maillon inséré.

Si le pointeur de la liste circulaire est NULL alors, nous le faisons pointer vers le nouveau maillon. Nous faisons ensuite pointer le maillon sur lui-même.

### Compter le nombre de maillon de la liste

Nous testons si le pointeur de la liste circulaire pointe vers NULL. Si ce n’est pas le cas, nous créons un pointeur parcourant la liste circulaire en incrémentant un compteur. Quand nous revenons au point de départ, nous retournons le compteur.

Si le pointeur de la liste circulaire est vide, nous retournons 0.

## Algorithme

**Module** listeCirculaire

**Type** maillon = {

numero : **Entier**

suivant : **Pointeur** maillon

}

//Procédure permettant d'initialiser la liste circulaire

//Paramètre de sortie : une liste circulaire

//Post-condition : la liste est initialisée

**Procédure** init(; p : **Pointeur** maillon)

**Début**

p <- **NULL**

**Fin** init

//Procédure permettant de désinitaliser la liste circulaire

//Paramètre d'entrée : une liste circulaire initialisée

//Paramètre de sortie : une liste circulaire vide et désinitialisée

//Pré-condition : la liste est initialisée

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

**Procédure** desinit(p : **Pointeur** maillon ; p : **Pointeur** maillon)

**Début**

**Tant** **Que** p ≠ **NULL** **Faire**

supp(p)

**Fin** **Faire**

**Fin** desinit

//Procédure permettant de supprimer le maillon pointé

//Paramètre d'entrée : une liste circulaire

//Paramètre de sortie : une liste circulaire avec l'élément pointé //en moins

//Pré-condition : la liste est initialisée

//Post-condition : l'élément pointé est supprimé

**Procédure** supp(p : **Pointeur** maillon ; p : **Pointeur** maillon)

**Début**

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

**Si** p ≠ **NULL** **Alors**

**Si** (**Contenu** p).suivant ≠ p **Alors**

tmp <- (**Contenu** p).suivant

(**Contenu** p).numero <- (**Contenu** tmp).numero

(**Contenu** p).suivant <- (**Contenu** tmp).suivant

**Détruire** tmp

**Sinon**

**Détruire** p

p <- **NULL**

**Fin** **Si**

**Fin** **Si**

**Fin** supp

//Procédure permettant d'afficher la liste circulaire

//Paramètres d'entrée : une liste circulaire

//Pré-condition : la liste est initialisée

**Procédure** affiche(p : **Pointeur** maillon)

**Début**

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

tmp <- p

**Si** p ≠ **NULL** **Alors**

**Répéter**

**Afficher** (**Contenu** tmp).numero

tmp <- (**Contenu** tmp).suivant

**Tant** **Que** (tmp ≠ p)

**Fin** **Si**

**Fin** affiche

//Procédure permettant de faire une rotation d'un cran

//Paramètre d'entrée : une liste circulaire

//Paramètre de sortie : une liste circulaire

//Pré-condition : la liste est initialisée

//Post-condition : le pointeur pointe vers le maillon suivant

**Procédure** rotation(p : **Pointeur** maillon ; p : **Pointeur** maillon)

**Début**

**Si** p ≠ **NULL** **Alors**

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

**Fin** **Si**

**Fin** rotation

//Procédure permettant d'insérer un nouveau maillon dans la liste //circulaire

//Paramètres d'entrée : une liste circulaire

// un maillon

//Paramètres de sortie : un liste circulaire avec un maillon en plus

//Pré-condition : la liste est initialisée

//Post-condition : le maillon a été ajouté

**Procédure** ajout(p : **Pointeur** maillon , nm : **Pointeur** maillon ; p : **Pointeur** maillon)

**Début**

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

**Si** p ≠ **NULL** **Alors**

(**Contenu** nm).suivant <- (**Contenu** p).suivant

(**Contenu** p).suivant <- nm

n <- (**Contenu** p).numero

(**Contenu** p).numero <- (**Contenu** nm).numero

(**Contenu** nm).numero <- n

**Sinon**

p <- nm

(**Contenu** nm).suivant <- nm

**Fin** **Si**

**Fin** ajout

//Fonction permettant d'obtenir le nombre de maillon d'une liste

//Paramètres d'entrée : une liste circulaire

//Valeur retournée : le nombre de maillon de cette liste

//Pré-condition : la liste est initialisée

**Fonction** nbMaillon(p : **Pointeur** maillon) **Retourne** **Entier**

**Début**

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

cpt : **Entier**

**Si** p ≠ **NULL** **Alors**

tmp <- p

cpt <- 0

**Répéter**

cpt <- cpt + 1

tmp <- (**Contenu** tmp).suivant

**Tant** **Que** tmp ≠ p

**Retourne** cpt

**Sinon**

**Retourne** 0

**Fin** **Si**

**Fin** nbMaillon

**Fin** listeCirculaire

## 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 81 82 83 84 85 86 87 88 89 90 91 92 | //  // listeCirculaire.cpp  // TP05  //  // Created by Oncle D. on 15/10/2014.  // Copyright (c) 2014 Oncle D. All rights reserved.  //  #include "listeCirculaire.h"  void init(struct maillon \*\*pp)  {  (\*pp) = NULL;  }  void desinit(struct maillon \*\*pp)  {  while ( (\*pp) != NULL)  {  supp(pp);  }  }  void supp(struct maillon \*\*pp)  {  struct maillon \*tmp;  if ((\*pp) != NULL) {  if ( (\*(\*pp)).suivant != (\*(\*pp)).suivant) {  tmp = (\*(\*pp)).suivant;  (\*(\*pp)).numero = (\*tmp).numero;  (\*(\*pp)).suivant = (\*tmp).suivant;  delete tmp;  }  else {  delete (\*pp);  (\*pp) = NULL;  }  }  }  void affiche(struct maillon \*p)  {  struct maillon \*tmp;  tmp = p;  if (p != NULL) {  do {  cout << (\*tmp).numero << endl;  tmp = (\*tmp).suivant;  } while (tmp != p);  }  }  void rotation(struct maillon \*\*pp)  {  if ((\*pp) != NULL) {  (\*pp) = (\*(\*pp)).suivant;  }  }  void ajout(struct maillon \*\*p, struct maillon \*nm)  {  int n;    if ((\*p) != NULL) {  (\*nm).suivant = (\*(\*p)).suivant;  (\*(\*p)).suivant = nm;  n = (\*(\*p)).numero;  (\*(\*p)).numero = (\*nm).numero;  (\*nm).numero = n;  }  else {  (\*p) = nm;  (\*nm).suivant = nm;  }  }  int nbMaillon(struct maillon \*p)  {  struct maillon \*tmp;  int cpt;    if (p != NULL) {  tmp = p;  cpt = 0;  do {  cpt++;  tmp = (\*tmp).suivant;  } while (tmp != p);  return cpt;  }  else  return 0;  } |

## Contenu de listeCirculaire.h

|  |  |
| --- | --- |
| 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 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 | //  // listeCirculaire.h  // TP05  //  // Created by Oncle D. on 15/10/2014.  // Copyright (c) 2014 Oncle D. All rights reserved.  //  #ifndef \_\_TP05\_\_listeCirculaire\_\_  #define \_\_TP05\_\_listeCirculaire\_\_  #include <stdio.h>  #include <iostream>  using namespace std;  struct maillon {  int numero;  struct maillon \*suivant;  };  //Procédure permettant d'initialiser la liste circulaire  //Paramètre de sortie : une liste circulaire  //Post-condition : la liste est initialisée  void init(struct maillon \*\*pp);  //Procédure permettant de désinitaliser la liste circulaire  //Paramètre d'entrée : une liste circulaire initialisée  //Paramètre de sortie : une liste circulaire vide et //désinitialisée  //Pré-condition : la liste est initialisée  //Post-condition : la liste est vide et désinitialisée  void desinit(struct maillon \*\*pp);  //Procédure permettant de supprimer le maillon pointé  //Paramètre d'entrée : une liste circulaire  //Paramètre de sortie : une liste circulaire avec l'élément //pointé en moins  //Pré-condition : la liste est initialisée  //Post-condition : l'élément pointé est supprimé  void supp(struct maillon \*\*pp);  //Procédure permettant d'afficher la liste circulaire  //Paramètres d'entrée : une liste circulaire  //Pré-condition : la liste est initialisée  void affiche(struct maillon \*p);  //Procédure permettant de faire une rotation d'un cran  //Paramètre d'entrée : une liste circulaire  //Paramètre de sortie : une liste circulaire  //Pré-condition : la liste est initialisée  //Post-condition : le pointeur pointe vers le maillon suivant  void rotation(struct maillon \*\*pp);  //Procédure permettant d'insérer un nouveau maillon dans la //liste circulaire  //Paramètres d'entrée : un liste circulaire  // un maillon  //Paramètres de sortie : un liste circulaire avec un maillon en //plus  //Pré-condition : la liste est initialisée  //Post-condition : le maillon a été ajouté  void ajout(struct maillon \*p, struct maillon \*nm);  //Fonction permettant d'obtenir le nombre de maillon d'une liste  //Paramètres d'entrée : une liste circulaire  //Valeur retournée : le nombre de maillon de cette liste  //Pré-condition : la liste est initialisée  int nbMaillon(struct maillon \*p);  #endif /\* defined(\_\_TP05\_\_listeCirculaire\_\_) \*/ |

# Algorithme principal

## Analyse du travail à faire

Nous initialisons une liste circulaire.

Puis nous affichons une liste de choix de gestion de cette liste circulaire pour l’utilisateur et lui demandons ce qu’il souhaite faire.

S’il choisit d’afficher la liste, nous utilisons la procédure d’affichage d’une liste circulaire.

S’il choisit d’ajouter un élément, nous lui demandons de saisir le numéro de cet élément puis l’ajoutons à la liste.

S’il choisit d’obtenir le nombre d’élément, nous lui affichons le nombre de maillons de la liste.

S’il choisit de faire une rotation ou de désinitialiser la liste, nous utilisons les procédures créées pour.

S’il choisit de quitter, nous désinitialisons la liste et arrêtons de lui proposer les choix.

S’il entre un choix incorrect, nous lui affichons que son choix est invalide.

## Algorithme

**Utilise** listeCirculaire

**Programme** liste

**Début**

**Avec** p : **Pointeur** maillon

tmp\_maillon : maillon

continuer : **Booléen**

choix : **Entier**

numero : **Entier**

nbr : **Entier**

continuer <- **Vrai**

init(p)

**Tant** **Que** continuer **Faire**

**Afficher** "LISTE CIRCULAIRE"

**Afficher** "1. Afficher la liste"

**Afficher** "2. Ajouter un élément"

**Afficher** "3. Compter le nombre de maillons"

**Afficher** "4. Faire une rotation"

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

**Afficher** "6. QUITTER"

**Afficher** "Saisir choix : "

**Saisir** choix

**Selon** choix **Faire**

**Cas** 1 :

affiche(p)

**Cas** 2 :

**Afficher** "Numéro de l'élément : "

**Saisir** numero

tmp\_maillon <- Nouveau maillon

(\*tmp\_maillon).numero <- numero

ajouter(p, tmp\_maillon)

**Cas** 3 :

nbr <- nbMaillon(p)

**Afficher** "Nombre de maillons : "

**Afficher** nbr

**Cas** 4 :

**Si** nbMaillon(p) > 0:

rotation(p)

**Sinon**

**Afficher** "Erreur, la liste est vide"

**Fin** **Si**

**Cas** 5 :

desinit(p)

**Cas** 6 :

desinit(p)

continuer <- **Faux**

**Défaut** :

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

**Fin** **Faire**

**Fin** **Faire**

**Fin** liste

## 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 | //  // main.cpp  // TP05  //  // Created by Oncle D. on 15/10/2014.  // Copyright (c) 2014 Oncle D. All rights reserved.  //  #include <iostream>  #include "listeCirculaire.h"  int main(int argc, const char \* argv[]) {  struct maillon \*p;  struct maillon \*tmp;  bool continuer;  int choix;  int nbr;  int numero;    continuer = true;  init(&p);    while (continuer) {  cout << "1. Afficher la liste" << endl;  cout << "2. Ajouter un élément" << endl;  cout << "3. Compter le nombre de maillon" << endl;  cout << "4. Faire une rotation" << endl;  cout << "5. Désinitialiser la liste" << endl;  cout << "6. QUITTER" << endl;    cout << "Saisir choix : ";  cin >> choix;    switch (choix) {  case 1:  affiche(p);  break;    case 2:  cout << "Numéro de l'élément : ";  cin >> numero;  tmp = new struct maillon;  (\*tmp).numero = numero;  ajout(&p, tmp);  break;    case 3:  nbr = nbMaillon(p);  cout << "Il y a " << nbr << " éléments dans la liste" << endl;  break;  case 4:  if (nbMaillon(p) > 0) {  rotation(&p);  cout << "Rotation effectuée." << endl;  }  else  cout << "La liste est vide, impossible de faire une rotation." << endl;  break;  case 5:  desinit(&p);  cout << "Liste désinitialisée." << endl;  break;  case 6:  continuer = false;  break;    default:  break;  }  }    return 0;  } |

## 

## Jeux d’essai

|  |  |  |
| --- | --- | --- |
| **Cas testé** | **Entrée** | **Résultat** |
| Désinitialiser une liste vide | 5 | Liste désinitialisée |
| Afficher une liste vide | 1 | (Rien) |
| Compter le nombre de maillons d’une liste vide | 3 | Il y a 0 élément dans cette liste |
| Faire une rotation d’une liste vide | 4 | Liste vide impossible de faire une rotation |
| Ajouter un élément | 2 puis 4 | Élément ajouté |
| Afficher la liste | 1 | 4 |
| Compter le nombre de maillons | 3 | Il y a 1 élément dans cette liste |
| Faire une rotation | 4 | Rotation effectuée |
| Désinitialiser la liste | 5 | Liste désinitialisée |
| Afficher la liste | 1 | (Rien) |
| Compter le nombre de maillons | 3 | Il y a 0 élément dans cette liste |
| Ajouter des éléments | 2 puis 15 |  |
|  | 2 puis 24 |  |
|  | 2 puis 33 |  |
|  | 2 puis 42 |  |
|  | 2 puis 51 | Éléments ajoutés |
| Afficher la liste | 1 | 51 42 33 15 |
| Compter le nombre de maillons | 3 | Il y a 5 éléments dans cette liste |
| Faire une rotation | 4 | Rotation effectuée |
| Afficher la liste | 1 | 42 33 15 51 |
| Quitter | 6 | Le programme se ferme |





























