## Linked List

**LinkedList.h**

|  |
| --- |
| #include "stdafx.h"  #include <stdio.h>  #include <stdlib.h>  typedef struct LinkedListNode {  //Definieër datatypes  int data;  //Einde data van struct  LinkedListNode\* next;  };  typedef struct LinkedListHeader  {  int count;  LinkedListNode\* head;  };  void LinkedListContruct(LinkedListHeader\*);  bool LinkedListDestruct(LinkedListHeader\*);  int LinkedListCount(LinkedListHeader\*);  void LinkedListAddRear(LinkedListHeader\*, LinkedListNode\*);  void LinkedListAddFront(LinkedListHeader\*, LinkedListNode\*);  LinkedListNode\* LinkedListItem(LinkedListHeader\*, int);  bool LinkedListDeepCopy(LinkedListHeader\*, LinkedListHeader\*); |

**LinkedList.cpp**

|  |
| --- |
| #include "LinkedList.h"  void LinkedListContruct(LinkedListHeader \*header)  {  header->count = 0;  header->head = NULL;  }  bool LinkedListDestruct(LinkedListHeader \*header)  {  if(header->head == NULL)  return false;  LinkedListNode \*node = header->head;  LinkedListNode \*next = node->next;  for(int i = 0; i < header->count; i++)  {  free(node);  node = next;  }  return true;  }  int LinkedListCount(LinkedListHeader \*header)  {  int cnt = 0;  if(header->head == NULL)  return 0;  LinkedListNode\* node = header->head;  while(1)  {  cnt++;  if(node->next == NULL)  break;  else  node = node->next;  }  header->count = cnt;  return cnt;  }  void LinkedListAddFront(LinkedListHeader \*header, LinkedListNode \*item)  {  if(header->head == NULL)  {  item->next = NULL;  header->head = item;  header->count++;  return;  }  item->next = header->head;  header->head = item;  header->count++;  }  void LinkedListAddRear(LinkedListHeader \*header, LinkedListNode \*item)  {  LinkedListNode \*last = header->head;  //Als er niks in de lijst zit, als eerste object toevoegen  if(header->head == NULL)  {  item->next = NULL;  header->head = item;  header->count++;  return;  }  //Laatste object in de lijst vinden  while(1)  {  if(last->next == NULL)  break;  else  last = last->next;  }  //Laatste blokje laten wijzen naar nieuw blokje  last->next = item;  //Volgende pointer in nieuw blokje naar null laten verwijzen  item->next = NULL;  header->count++;  }  LinkedListNode\* LinkedListItem(LinkedListHeader \*header, int itemNumber)  {  //Eerst kijken of index binnen lijst valt  if(itemNumber > header->count)  return NULL;  LinkedListNode \*node = header->head;  //aftellen totdat je itemNumber keer door de lijst bent geitereerd  while(1)  {  if(itemNumber == 1)  break;  else  {  itemNumber--;  node = node->next;  }  }  return node;  }  //Alleen herbruikbaar als functie wordt aangepast  bool LinkedListDeepCopy(LinkedListHeader \*original, LinkedListHeader \*copy)  {  //Kopie opnieuw initialiseren  LinkedListContruct(copy);  //Deep copy  int cnt = 1;  while(original->count != copy->count)  {  LinkedListNode \*node = (LinkedListNode\*)malloc(sizeof(LinkedListNode));  node->data = LinkedListItem(original, cnt)->data;  LinkedListAddRear(copy, node);  cnt++;  }    return true;  } |

## Queue

Een queue gedraagt zoals een een wachtrij. Nieuwe objecten sluiten op het einde aan. Degene die er als eerste in staat zal er ook weer als eerste uit komen.

**Queue.h**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  typedef struct QueueNode {  //Queue data goes here  int number;  //End of queue data. do not touch below  bool isEmpty;  QueueNode \*next;  };  typedef struct QueueHeader {  int count;  QueueNode \*head;  QueueNode \*tail;  };  void QueueInit(QueueHeader\*);  void QueueDestruct(QueueHeader\*);  int QueueEnqueue(QueueHeader\*, QueueNode\*);  QueueNode QueueDequeue(QueueHeader\*);  QueueNode QueuePeek(QueueHeader\*);  bool QueueIsEmpty(QueueHeader\*);  bool QueueCount(QueueHeader\*, int\*); |

**Queue.cpp**

|  |
| --- |
| #include "Queue.h"  void QueueInit(QueueHeader \*header)  {  if(header->count != 0)  {  while(header->head != NULL)  {  QueueDequeue(header);  }  }  header->count = 0;  header->head = NULL;  header->tail = NULL;  }  void QueueDestruct(QueueHeader \*header)  {  while(header->head != NULL)  QueueDequeue(header);  }  int QueueEnqueue(QueueHeader \*header, QueueNode \*node)  {  //Als count null is moet moeten head en tail naar een ding wijzen  if(header->count == 0)  {  node->next = NULL;  node->isEmpty = false;  header->head = node;  header->tail = node;  header->count++;  return 1;  }  else  {  node->next = NULL;  node->isEmpty = false;  header->tail->next = node;  header->tail = node;  header->count++;  return 1;  }  return 0;  }  QueueNode QueueDequeue(QueueHeader \*header)  {  if(header->count == 0)  {  //dummy queue terug geven om aan te geven dat er geen is  QueueNode dummy = {0};  dummy.isEmpty = true;  return dummy;  }  else  {  //pointer houden voor reference  QueueNode \*postagePaid = header->head;  //Andere node aanmaken om gegeheugen te kunnen wissen  QueueNode package = \*postagePaid;  //Als er wel nog iets in de rij staat de pointers veranderen  if(header->head->next != NULL)  {  header->head = header->head->next;  header->count--;  }  //Anders is de head en tail nul  else  {  header->head = NULL;  header->tail = NULL;  header->count--;  }  //Geheugen van pointer vrijmaken  free(postagePaid);  //Pakketje met data terug geven. Dit wordt opgehaald door de garbage collector  return package;  }  }  //Geeft het eerste object van de queue zonder deze te verwijderen  QueueNode QueuePeek(QueueHeader \*header)  {  if(header->count == 0)  {  //dummy queue terug geven om aan te geven dat er geen is  QueueNode dummy = {0};  dummy.isEmpty = true;  return dummy;  }  else  return \*header->head;  }  bool QueueIsEmpty(QueueHeader \*header)  {  if(header->count == 0)  return true;  else  return false;  }  bool QueueCount(QueueHeader \*header, int \*dst)  {  if(header->count == 0)  {  \*dst = -1;  return false;  }  else  \*dst = header->count;  return true;  } |

## Testprogramma

Voor dit geheel te testen heb ik het volgende test-programma gemaakt

**Main.cpp**

|  |
| --- |
| #include "LinkedList.h"  #include "Queue.h"  #include <stdio.h>  #include <stdlib.h>  #include <conio.h>  void main(void)  {  while(1)  {  char chc;  printf("(1)Linked List, (2)Queue: ");  scanf("%c", &chc);  if(chc == '1')  {  LinkedListHeader list;  LinkedListContruct(&list);  bool quit = false;  char keuze;  while(1)  {  fflush(stdin);  printf("1 = count, 2 = add front, 3 = add rear, 4 = find index, 5 = print all, 6 = quit: , 7 = Deep Copy");  scanf("%c", &keuze);  switch(keuze)  {  case '1':  printf("%d", LinkedListCount(&list));  break;  case '2':  {  LinkedListNode \*eerst = (LinkedListNode\*)malloc(sizeof(LinkedListNode));  printf("Waarde die je wilt invoegen: ");  scanf("%d", &eerst->data);  LinkedListAddFront(&list, eerst);  }  break;  case '3':  {  LinkedListNode \*laatst = (LinkedListNode\*)malloc(sizeof(LinkedListNode));  printf("Waarde die je wilt invoegen: ");  scanf("%d", &laatst->data);  LinkedListAddRear(&list, laatst);  }  break;  case '4':  {  int index = 0;  printf("index: ");  scanf("%d", &index);  LinkedListNode \*node = LinkedListItem(&list, index);  if(node != NULL)  printf("Index %d heeft als waarde %d", index, node->data);  }  break;  case '5':  {  int cnt = 1;  while(1)  {  LinkedListNode \*node = LinkedListItem(&list, cnt);  if(node != NULL)  {  printf("Op index %d is het getal %d\n", cnt, node->data);  cnt++;  }  else  break;  }  }  break;  case '6':  quit = true;  break;  case '7':  {  LinkedListHeader copiedHeader = {0};  LinkedListContruct(&copiedHeader);  LinkedListDeepCopy(&list, &copiedHeader);  //Gekopieerd. \_getch() wordt aangeroepen om te kijken of het ook werkt  \_getch();  }  break;  default:  break;  }  printf("\n");  if(quit)  {  LinkedListDestruct(&list);  break;  }  }  }  if(chc == '2')  {  QueueHeader header = {0};  QueueInit(&header);  bool quit = false;  char keuze;  while(1)  {  fflush(stdin);  printf("1=enqueue, 2=dequeue, 3=peek, 4=isempty, 5=count,6=quit: ");  scanf("%c", &keuze);  switch(keuze)  {  case '1':  {  QueueNode \*node = (QueueNode\*)malloc(sizeof(QueueNode));  int num = 0;  printf("Getal om te queuen: ");  scanf("%d", &num);  node->number = num;  QueueEnqueue(&header, node);  }  break;  case '2':  {  QueueNode node = QueueDequeue(&header);  if(!node.isEmpty)  printf("Het volgende deel van de queue is %d" , node.number);  else  printf("De queue is leeg!");  }  break;  case '3':  {  QueueNode node = QueuePeek(&header);  if(!node.isEmpty)  printf("Na het spieken blijkt dat het volgend nummer %d is" , node.number);  else  printf("de queue is leeg!");  }  break;  case '4':  if(QueueIsEmpty(&header))  printf("ja");  else  printf("nee");  break;  case '5':  {  int num = 0;  if(QueueCount(&header, &num))  printf("Er staan %d nummers in de rij", num);  else  printf("Er staan geen nummers meer in de rij");  }  break;  case '6':  quit = true;  break;  default:  break;  }  printf("\n");  if(quit)  {  QueueDestruct(&header);  break;  }  }  }  }  } |