LOICH KAMDOUM DEAMENI
Mat-Nr: 506520

Uebungsblatt 3:

Übung 1: Pointer

- i + 1 -> korrekt (int)
- *p -> korrekt (int)
- *p + 3 -> korrekt (int)
- &i == p -> korrekt (bool)
- i == *p -> korrekt (bool)
- &p -> korrekt ()
- p + 1 -> nicht korrekt
- &p == i -> nicht korrekt
- **(&p) -> korrekt (int)
- *p + i > 1 -> korrekt (bool)

Übung 2: Destruktor

- alle Objekte der Klasse C wegzuräumen -> richtig
- Objekte der Klasse C im Heap wegzuräumen -> richtig
- alle Komponenten von Objekten der Klasse C wegzuräumen -> richtig
- Komponenten von Objekten der Klasse C , die im Heap liegen, wegzuräumen -> falsch

Übung 3: new & delete

1. Große Objekte oder Felder deren Größe erst zur Laufzeit bekannt sind, können mit Hilfe von new auf dem Heap alloziert werden. Hier ist der Wert von int noch nicht festgelegt.

2. nein

3.
a) Die beide haben keinen Wert
b) In der erste Aufruf das Objekt existiert nicht mehr, der Zeiger schon, deswegen bei Zugriff folgt ein segmentation fault. Das ist besonders gefährlich wenn mehrere Zeiger auf dasselbe Objekt existieren.

Übung 4: Vekettete Liste

1) der Pointer zeigt im Moment nicht auf eine Variable/Funktion und zeigt nicht auf irgendwelche Adresse in Heap.

2) Der Kode steht unten

5) Wenn mann die Liste Kopiert, wird die Liste richtig kopiert. Das ist möglich dank des Konstruktor und Zuweisungsoperator "=".

Übung 5: Verkettete Liste (min/max)

1.
- gute Testfälle: Die liste ist Aufsteigend geordnet.
- Seiteneffekte: Die liste ist leer.
- Sieht den Kode.

3. Wir müssen static Variable und Zeiger benutzen. Das führt dazu, dass die Methoden auch als static definiert werden müssen.

Node.h

```
#include <iostream>
#include <stdio.h>

#ifndef NODE_H_
#define NODE_H_

class Node
{
private:
   /* data */
   Node *next;

public:
   /* Konstruktoren */
   Node();
   Node(int);
   ~Node();

   /* data */
   friend class List;
   int value;

};

#endif /* NODE_H_ */
```

Node.cpp

```
#include "Node.h"

using namespace std;

Node::Node()
{
   value = 0;
   next = nullptr;
}

Node::Node(int value_)
{
   value = value_;
   next = nullptr;
}

Node::~Node()
{
   value = 0;
   next = nullptr;
}
```

List.h

```
#include <iostream>
#include <stdio.h>

#include "Node.h"

#ifndef LIST_H_
#define LIST_H_

class List
{
private:
   /* data */
   Node *first_;  //pointer to the first element
   Node *next_;  //pointer to the next element

public:
   /* Konstruktoren */
   List ();
   ~List();
   List (const List &);  //Copy-Konstruktor

   /* Methoden */
   Node* first() const;
   Node* next(const Node *n) const;
   void append (int i);
   void insert (Node *n, int i);
   void erase (Node *n);

   const Node* findMin() const;
   const Node* findMax() const;
   void testListMinMax();

};

#endif /* LIST_H_ */
```

List.cpp

```
#include "List.h"

using namespace std;

List::List()
{
   first_ = new Node();
   next_ = new Node();
}

List::List(const List &list_)
{
   first_ = list_.first_;
   next_ = list_.next_;
```

```
}

List::~List()
{
   Node *node_tmp = first_;
   Node *next_node = new Node();

   if (node_tmp != nullptr)
   {
      while(next(node_tmp) != nullptr)
      {
         next_node = next(node_tmp);
         delete node_tmp;
         node_tmp = next_node;
      }
      delete node_tmp;
      delete next_node;
   }
}

Node* List::first() const
{
   return first_;
}

Node* List::next(const Node *n) const
{
   return n->next;
}

void List::append (int i)
{
   Node *node_tmp = first_;
   Node *next_node = new Node();

   if (node_tmp->value != 0)
   {
      while(node_tmp->next != nullptr)
      {
         next_node = next(node_tmp);
         node_tmp = next_node;
      }
      node_tmp->next = new Node(i);
   }
   else
   {
      first_ = new Node(i);
   }
}

void List::insert (Node *n, int i)
{
```

```
    Node *current_node = first_;
    Node *next_node = first_;
    Node *prev_node = first_;
    Node *tmp;

    if (current_node != n)
    {
        while(current_node != n)
        {
            prev_node = current_node;
            next_node = next(current_node);
            current_node = next_node;
        }
        tmp = new Node(i);
        tmp->next = current_node;
        prev_node = tmp;
    }
    else
    {
        tmp = new Node(i);
        tmp->next = current_node;
        first_ = tmp;
    }
}

void List::erase (Node *n)
{
    Node *node_tmp = first_;
    Node *next_node = nullptr;

    if (node_tmp != n)
    {
        while(node_tmp != n)
        {
            next_node = next(node_tmp);
            node_tmp = next_node;
        }
        next_node = next(node_tmp);
        node_tmp->next = next(next(node_tmp));
        delete next_node;
    }
    else
    {
        node_tmp->next = nullptr;
    }
}

const Node* List::findMin() const
{
    Node *node_tmp = first_;
    Node *min_node = first_;
    Node *next_node = new Node();
```

```
    while(node_tmp->next != nullptr)
    {
        next_node = next(node_tmp);
        node_tmp = next_node;
        if(node_tmp->value < min_node->value)
            min_node = node_tmp;
    }
    return min_node;
}

const Node* List::findMax() const
{
    Node *node_tmp = first_;
    Node *max_node = first_;
    Node *next_node = new Node();

    while(node_tmp->next != nullptr)
    {
        next_node = next(node_tmp);
        node_tmp = next_node;
        if(node_tmp->value > max_node->value)
            max_node = node_tmp;
    }
    return max_node;
}

void List::testListMinMax()
{
    cout <<'\n'<<"Der min in der Liste ist: "<< this->findMin()->value;
    cout <<'\n'<<"Der max in der Liste ist: "<< this->findMax()->value;
    cout <<endl;
}
```

Main.cpp

```
#include <iostream>
#include <stdio.h>

#include "List.h"
#include "Node.h"

using namespace std;

int main ();

int main ()
{
    // cout << "lol"<<'\n';
    List list;
    list.append(2);
    list.append(3);
    list.insert(list.first(), 1);
```

```
    for (Node *n = list.first(); n != 0; n = list.next(n))
        std::cout << n->value << std::endl;

    cout <<'\n'<<"Nach der Kopie"<<'\n';
    // Kopie von der Liste
    List list2 = list;
    for (Node *n = list2.first(); n != 0; n = list2.next(n))
        std::cout << n->value << std::endl;

    //Neue Elemente werden in der Liste hinzugefügt zur Prüfung der Methode testMinMax()
    cout <<'\n'<<"Prüfung der Methode testMinMax()"<<'\n';
    list.append(10);
    list.testListMinMax();

    return 0;
}
```