Mechatronic Engineering

Object Oriented Programing and Software Engineering
Laboratory instruction 12
C++ introduction

AGH Kraków, 2020

Contents

1 Linked lists

Linked list is a dynamic structure of objects (data). This allows one to freely change its size while the program is running. The only limitation is computer memory. List is made up of connected elements. There are two types of lists:

- Singly linked lists,
- Doubly linked lists.

1.1 Singly linked lists

Each element of this list is connected only to the next element of this list. This type of list can be navigated only in one direction from the head (first element) to the tail (last element).The operation diagram of a sinly linked list is shown in the figure 1.

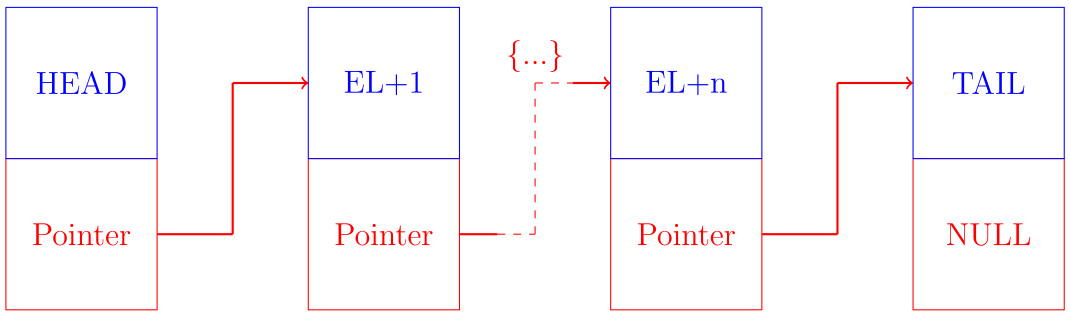

Figure 1: Singly linked list schematic

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

using namespace std;

//Element declaration
struct lst_el{
  lst_el * next;
  int key;
  string name;

};

```

```

//definition of sinly listed list class
class TsingleList{
    lst_el * head, * tail;
    int cnt;

  public:
    TsingleList(){
      head = tail = NULL;
      cnt  = 0;
    }

    ~TsingleList(){
      lst_el * el;

      while(head){
        el = head->next;
        delete head;
        head = el;
      }
    }

//Method returning list size
    unsigned size(){
      return cnt;
    }

//Method adding an element at the front of the list
    lst_el * push_head(lst_el * el){
      el->next = head;
      head = el;
      if(!tail) tail = head;
      cnt++;
      return head;
    }

//Method adding an element at the end of the list
    lst_el * push_tail(lst_el * el){
      if(tail) tail->next = el;
      el->next = NULL;
      tail = el;
      if(!head) head = tail;
      cnt++;
      return tail;
    }

//Method adding an element (el1) after an element (el2)
    lst_el * insert(lst_el * el1, lst_el * el2){
      el1->next = el2->next;
      el2->next = el1;
      if(!(el1->next)) tail = el1;
      cnt++;
      return el1;
    }

//Method deleting the first element of the list
    lst_el * rmHead(){
      lst_el * el;

      if(head){
        el = head;
        head = head->next;
        if(!head) tail = NULL;
        cnt--;
        return el;
      }
      else return NULL;
    }

//Method deleting the last element of the list
    lst_el * rmTail(){
      lst_el * el;

      if(tail){
        el = tail;
        if(el == head) head = tail = NULL;
        else{
          tail = head;
          while(tail->next != el) tail = tail->next;
          tail->next = NULL;
        }
        cnt--;
        return el;
      }
      else return NULL;
    }
```

```
//Method deleting the el element of the list
    lst_el * erase(lst_el * el){
      lst_el * el1;

      if(el == head) return rmHead();
      else{
        el1 = head;
        while(el1->next != el) el1 = el1->next;
        el1->next = el->next;
        if(!(el1->next)) tail = el1;
        cnt--;
        return el;
      }
    }

//Method returns nth element of the list
    lst_el * index(int n){
      lst_el * el;

      if((!n) || (n > cnt)) return NULL;
      else if(n == cnt) return tail;
      else{
        el = head;
        while(--n) el = el->next;
        return el;
      }
    }

//Methods used to display data stored in the list
    void showKeys(){
      lst_el * el;

      if(!head) cout << "List is empty." << endl;
      else{
        el = head;
        while(el){
          cout << el->key << " ";
          el = el->next;
        }
        cout << endl;
      }
    }
```

```

     void showNames(){
      lst_el * el;

      if(!head) cout << "List is empty." << endl;
      else{
        el = head;
        while(el){
          cout << el->name << " ";
          el = el->next;
        }
        cout << endl;
      }
    }

    void showElements(){
      lst_el * el;

      if(!head) cout << "List is empty." << endl;
      else{
        el = head;
        while(el){
          cout << "Name: " << el->name << ", key: " << el->key <<";
              ";
          el = el->next;
        }
        cout << endl;
      }
    }
};



int main(){
  TsingleList  sl;
  lst_el * p;
  int          i;

  cout << "List should be empty : "; sl.showKeys();

//This will add 5 elements at the front of the list
  for(i = 1; i <= 5; i++){
    p = new lst_el;
```

```cpp
    p->key = i;
    cout << "Enter name of the element: ";
    cin >> p->name;
    sl.push_head(p);
  }

  cout << "Now there should be "<< sl.size() <<" elements in the
     list: "; sl.showElements(); cout << endl;
  cout << "Program also displays single fields of the elements\n";
  cout << "Keys: "; sl.showKeys(); cout << endl;
  cout << "Names: "; sl.showNames(); cout << endl;

//This will add 5 elements at the back of the list
  for(i = 1; i <= 5; i++){
    p = new lst_el;
    p->key = i;
    p->name = to_string(i);
    sl.push_tail(p);
  }

  cout << "Keys of the list: "; sl.showKeys();
  cout << "Names of the list: "; sl.showNames();

//Removing first element
  sl.rmHead();

  cout << "Keys of the list after operations: "; sl.showKeys();

//Removing last element
  sl.rmTail();

  cout << "Keys of the list after operations: "; sl.showKeys();

//Removing n-th element
  delete sl.erase(sl.index(3));

  cout << "Keys of the list after operations: "; sl.showKeys();

//Another way of removing an element
  delete sl.erase(sl.index(sl.size() - 1));

  cout << "Keys of the list after operations: "; sl.showKeys();

```

```
//Adding new element after 4th element
  p = new lst_el;
  p->key = 9;
  p->name = to_string(9);
  sl.insert(p,sl.index(4));

  cout << "Keys of the list after operations: "; sl.showKeys();

//learing the list
  while(sl.size()) sl.rmHead();

  cout << "Empty list: "; sl.showElements();

  cout << endl << endl;


  return 0;
}
```

Task

Based on the informations provided in this manual, please improve the simple RPG caracter creation program.

Program requirements:
1. Add a history of the last 10 fights of the selected hero displayed as a list.