Laboratory 6

SOURCE

Exercise 1

```
#include <string>
#include <iostream>
#include <algorithm>
using namespace std;

void replace_message(std::string *str)
{
  const string msg = "message";
  const string real_msg = "real MESSAGE";
  const string og_msg = "original MESSAGE";
  size_t msg_index;
  while(str->find(real_msg) < 9999)
  {
    msg_index = str->find(real_msg);
    str->replace(msg_index, real_msg.size(), og_msg);
  }
  while(str->find(msg) < 9999)
  {
    msg_index = str->find(msg);
    str->replace(msg_index, msg.size(), real_msg);
  }
}

int main(){
  string txt;
  txt = " real message!!!"; // overloaded operator =
  cout << txt << endl;

  if (txt == "real message!!!") {
    cout << "comparison with use of the overloaded operator"
         << endl;
  } else {
    cout << "!@#$% ???" << endl;
  }

  cout << "text size =" << txt.size() << endl;
  cout << "Where is NULL???" << endl;

  for (size_t i = 0; i < txt.size(); i++) {
    cout << txt.at(i) << " ";
    /* cout << txt[i] << endl;
       alternative, but using .at() is safer */
    /*  2.
        operator[] does not do range checking. Accessing element not presenting in
```

```
vector silently leads to undefined behavior.
        .at() member function does range checking and throws an exception when you
are trying to access nonexisting element.
    */
  }
  std::cout<<"\n";

  string txt2(" Find out what will happen with the message.");

  txt.append(txt2).append(" One more message?");

  txt.replace
    (txt.find("message!!!"),
     string("message!!!").length(),
     "MESSAGE!!!");

  cout << txt << endl;

  replace_message(&txt);

  cout << txt << "\n";

  return 0;
}
```

```
output:
 real message!!!
!@#$% ???
text size =16
Where is NULL???
  r e a l   m e s s a g e ! ! !
 real MESSAGE!!! Find out what will happen with the message. One more message?
 original MESSAGE!!! Find out what will happen with the real MESSAGE. One more
real MESSAGE?
```

Exercise 2

```
#include <iostream>
#include <list>
#include <stdlib.h>
#include <ctime>
#include <iterator>
#include <bits/stdc++.h>

template<typename T>
void print_list(std::list<T> *list)
{
    for(T c : *list){ std::cout<<c<<" "; }
    std::cout<<"\n";
```

```
}

void add_random_letters(std::list<char> *list, int count)
{
    std::srand(time(NULL));
    while(count > 0)
    {
        list->push_back(('a'+rand()%24));
        count--;
    }
}

void pop_char_in_middle(std::list<char> *list, int count)
{
    while(count > 0)
    {
        std::list<char>::iterator it = list->begin();
        std::advance(it, list->size()/2);
        list->erase(it);
        count--;
    }
}

void capitilize_letters(std::list<char> *list, std::list<char>::iterator it, int
count)
{
    while(count > 0)
    {
        *it=char(*it-32);
        std::advance(it,1);
        count--;
    }
    print_list(list);
}

int main()
{
    std::list<char> list;
    // 1.
    std::cout<<"list with 10 letter in alphabetical order:\n";
    char letter = 'a';
    while(list.size() < 10)
    {
        list.push_back(letter++);
    }
    print_list(&list);
    // 2.
    std::cout<<"list with added 3 random letters:\n";
    add_random_letters(&list, 3);
    print_list(&list);
    // 3.
    std::cout<<"list with 2 erased elements\n";
    pop_char_in_middle(&list, 2);
    print_list(&list);
```

```
    std::cout<<"capitilized 3 elements\n";
    capitilize_letters(&list,list.begin(),3);
    // 4.
    std::cout<<"reversed list\n";
    list.reverse();
    print_list(&list);
}
```

```
output:
list with 10 letter in alphabetical order:
a b c d e f g h i j
list with added 3 random letters:
a b c d e f g h i j r h p
list with 2 erased elements
a b c d e f i j r h p
capitilized 3 elements
A B C d e f i j r h p
reversed list
p h r j i f e d C B A
```