Министерство науки и высшего образования Российской Федерации

федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»   
(НИУ ИТМО)

Факультет программной инженерии и компьютерной техники

Дисциплина: Программирование на C++

Отчетпо Лабораторной работе № 4

Выполнил

Студент группы P4119

Суховей Я. Ю.

Преподаватель

Лаздин А. В.

Санкт-Петербург  
2024

# **Часть 1 Исходный код**

Исходный код программы:

[https://github.com/Yars2021/cpp-labs/tree/main/lab3](https://github.com/Yars2021/cpp-labs/tree/main/lab2)

Основной класс с шаблонами:

#include <iostream>  
  
  
template <typename T> class FuncListItem {  
 private:  
 T \*value;  
 FuncListItem<T> \*next\_ptr, \*prev\_ptr;  
 public:  
 FuncListItem();  
 FuncListItem(T \*value);  
 ~FuncListItem();  
 void set\_value(T \*value);  
 void set\_next\_ptr(FuncListItem<T> \*next\_ptr);  
 void set\_prev\_ptr(FuncListItem<T> \*prev\_ptr);  
 T \*get\_value();  
 FuncListItem<T> \*get\_next\_ptr();  
 FuncListItem<T> \*get\_prev\_ptr();  
};  
  
  
template <typename T> FuncListItem<T>::FuncListItem() {  
 this->value = nullptr;  
 this->next\_ptr = nullptr;  
 this->prev\_ptr = nullptr;  
}  
  
template <typename T> FuncListItem<T>::FuncListItem(T \*value) {  
 this->value = value;  
 this->next\_ptr = nullptr;  
 this->prev\_ptr = nullptr;  
}  
  
template <typename T> FuncListItem<T>::~FuncListItem() {  
 delete(this->value);  
  
 if (this->next\_ptr == nullptr) {  
 if (this->prev\_ptr != nullptr) {  
 this->prev\_ptr->next\_ptr = nullptr;  
 this->prev\_ptr = nullptr;  
 }  
 } else {  
 if (this->prev\_ptr == nullptr) {  
 this->next\_ptr->prev\_ptr = nullptr;  
 this->next\_ptr = nullptr;  
 } else {  
 this->prev\_ptr->next\_ptr = this->next\_ptr;  
 this->next\_ptr->prev\_ptr = this->prev\_ptr;  
 this->prev\_ptr = nullptr;  
 this->next\_ptr = nullptr;  
 }  
 }  
}  
  
template <typename T> void FuncListItem<T>::set\_value(T \*value) {  
 this->value = value;  
}  
  
template <typename T> void FuncListItem<T>::set\_next\_ptr(FuncListItem<T> \*next\_ptr) {  
 this->next\_ptr = next\_ptr;  
}  
  
template <typename T> void FuncListItem<T>::set\_prev\_ptr(FuncListItem<T> \*prev\_ptr) {  
 this->prev\_ptr = prev\_ptr;  
}  
  
template <typename T> T \*FuncListItem<T>::get\_value() {  
 return this->value;  
}  
  
template <typename T> FuncListItem<T> \*FuncListItem<T>::get\_next\_ptr() {  
 return this->next\_ptr;  
}  
  
template <typename T> FuncListItem<T> \*FuncListItem<T>::get\_prev\_ptr() {  
 return this->prev\_ptr;  
}  
  
  
template <typename T> class FuncList {  
 private:  
 FuncListItem<T> \*head, \*tail;  
 public:  
 FuncList();  
 FuncList(T \*arr, int len);  
 ~FuncList();  
 FuncList<T> &operator>>(T item);  
 FuncList<T> &operator<<(T item);  
 void print(std::ostream &out\_stream);  
};  
  
  
template <typename T> FuncList<T>::FuncList() {  
 this->head = nullptr;  
 this->tail = nullptr;  
}  
  
template <typename T> FuncList<T>::FuncList(T \*arr, int len) {  
 this->head = nullptr;  
 this->tail = nullptr;  
  
 for (int i = 0; i < len; i++) {  
 \*this << arr[i];  
 }  
}  
  
template <typename T> FuncList<T>::~FuncList() {  
 FuncListItem<T> \*current = this->head, \*next;  
  
 this->head = nullptr;  
 this->tail = nullptr;  
  
 while (current != nullptr) {  
 next = current->get\_next\_ptr();  
 delete(current);  
 current = next;  
 }  
}  
  
template <typename T> FuncList<T> &FuncList<T>::operator>>(T item) {  
 T \*value = new T;  
 \*value = item;  
  
 FuncListItem<T> \*new\_item = new FuncListItem<T>(value);  
  
 if (this->head == nullptr) {  
 this->head = new\_item;  
 this->tail = this->head;  
 } else {  
 this->head->set\_prev\_ptr(new\_item);  
 new\_item->set\_next\_ptr(this->head);  
 this->head = new\_item;  
 }  
  
 return \*this;  
}  
  
template <typename T> FuncList<T> &FuncList<T>::operator<<(T item) {  
 T \*value = new T;  
 \*value = item;  
  
 FuncListItem<T> \*new\_item = new FuncListItem<T>(value);  
  
 if (this->head == nullptr) {  
 this->head = new\_item;  
 this->tail = this->head;  
 } else {  
 this->tail->set\_next\_ptr(new\_item);  
 new\_item->set\_prev\_ptr(this->tail);  
 this->tail = new\_item;  
 }  
  
 return \*this;  
}  
  
template <typename T> void FuncList<T>::print(std::ostream &out\_stream) {  
 FuncListItem<T> \*current = this->head;  
  
 out\_stream << "[";  
  
 while (current != nullptr) {  
 out\_stream << \*(current->get\_value()) << ((current->get\_next\_ptr() != nullptr) ? ", " : "");  
 current = current->get\_next\_ptr();  
 }  
  
 out\_stream << "]\n";  
}  
  
  
int main(int argc, char \*\*argv) {  
 int a[5] = {1, 2, 3, 4, 5};  
 FuncList<int> func\_list = FuncList<int>(a, 5);  
  
 func\_list.print(std::cout);  
 func\_list << 5 << 10 << 6 >> 4;  
 func\_list.print(std::cout);  
  
 return 0;  
}

# **Часть 2 Пример использования**

**Заключение**

В данной работе был реализован собственный класс с шаблонами на C++. В качестве примера взят двусвязный список с переопределенными операторами добавления в начало и конец, позволяющими выполнять несколько подряд. Также был создан метод print, выводящий содержимое списка в указанный поток вывода.