rr

**МIНIСТЕРСТВО ОСВIТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**

**ІМЕНІ ІГОРЯ СІКОРСЬКОГО”**

**Факультет прикладної математики**

**Кафедра програмного забезпечення комп’ютерних систем**

**Лабораторна робота №4**

**з дисципліни “ Основи програмування”**

**тема «Основи C++. Класи»**

|  |  |  |
| --- | --- | --- |
| **Виконав**  **студент I курсу**  **групи КП-83**  **Мортіков Владислав Євгенович**  **варіант №16** |  | **Перевірив**  **“\_\_\_\_” “\_\_\_\_\_\_\_\_\_\_\_\_”  2019  р.**  **викладач**  **Гадиняк Руслан Анатолійович**  **(прізвище, ім’я, по батькові)** |

**Київ 2019**

### **Мета**

Вивчити основні відмінності між кодом мови С та мови С++.  
Навчитись створювати С++ класи для реалізації базових АТД.

### **Загальні вимоги до завдання**

1. Скопіювати у проект даного завдання рішення лабораторної роботи №1.
2. Змінити розширення файлів .c на .cpp (або .cc) та спробувати зібрати проект за допомогою g++ \*.cpp -std=c++17 -pedantic-errors -Wall -Werror -Wno-unused.  
   Виправити всі помилки компіляції, наприклад:
   1. обов'язкове явне приведення вказівників з void \*
   2. замінити всі VLA масиви на динамічно створені масиви
3. Замінити всі застарілі підключення стандартної бібліотеки С на нові (приклад: <stdlib.h> на <cstdlib>)
4. Замінити всі виклики malloc() на оператор new, а виклики free() на оператор delete. Вийняток: виклики всередині динамічних АТД для використання realloc().
5. Переписати АТД Список та АТД за варіантом з використанням С++ класів:
   1. Змінити ключове слово struct на class (наприклад, struct \_\_List -> class \_\_List, це зробить всі поля приватними).  
      Клас обов'язково має бути описаний у заголовочному файлі.
   2. Видалити рядок із typedef для типу АТД, змінити тег класу на синонім, що був у typedef (class \_\_List -> class List).
   3. Додати до назв всіх полів класу суфікс \_ (size -> size\_).
   4. Додати всередині класу після всіх полів розділ public:.  
      Прототипи публічних методів писати у цьому розділі.
   5. Конструктори і деструктор:
      1. Створити у класі прототипи конструктора за замовчуванням і деструктора (наприклад, List(), ~List()).
      2. Реалізувати конструктор за замовчуванням (List::List()) і деструктор (List::~List()) перенісши до них код ініціалізацітора та деініціалізатора відповідно.
      3. Видалити в цих функціях параметр self і замість його імені використовувати прихований вказівник this.
      4. Видалити прототипи, реалізації і виклики ініціалізацітора, деініціалізатора, алокатора і деалокатора.
      5. Реалізації конструкторів та деструкторів мають бути винесені у окремий файл з кодом.
   6. Методи класу:
      1. Внести прототипи всіх операцій у клас без префіксів типу у назві (наприклад, size\_t size();).
      2. Змінити назву у реалізаціях цих методів з використанням оператора :: (наприклад, size\_t List::size() { ... }).
      3. Видалити параметр self і замість його імені використовувати прихований вказівник this.
      4. Змінити всі виклики методів з використанням оператора . (для екземплярів) або -> (для вказівників на екземпляри). Наприклад:  
         List\_size(&lst) -> lst.size() або  
         List\_size(pLst) -> pLst->size().
      5. Реалізації всіх методів мають бути винесені у окремий файл з кодом.
   7. Перевантаження операторів:
      1. Замість методів get() і set() для доступу до зчитування-запису елементів колекцій реалізувати operator [] (**Додаток А**).
      2. Реалізації всіх перевантажених операторів мають бути винесені у окремий файл з кодом.
6. У даній лабораторній роботі НЕ дозволяється використовувати:
   1. шаблонні класи
   2. стандартні контейнери
7. Опціонально можна:
   1. Підключити <iostream> та замінити весь вивід через printf() на вивід за допомогою std::cout, std::endl та спеціального оператора вставки у потік <<.
   2. Підключити <string> та використати тип std::string замість const char \* та char \* (якщо є використання).

**Тексти коду програм**

|  |
| --- |
| **main.hpp** |
| // #include <cstdio>  // #include <cstdlib>  #include "list.hpp"  #include "deque.hpp"  //------------------------------------------------  *void* modifyDeque(Deque \*firstDeque, Deque \*secondDeque, List \*mainlist);  *void* createListFromDequeu(Deque \*firstDeque, Deque \*secondDeque, List \*dequeList);  //------------------------------------------------  *int* main()  {  FILE \*myfile = fopen("data.txt", "r");  if (myfile == NULL)  {  return 1;  }  List mainList;  *float* buf;  while (fscanf(myfile, "%f", &buf) == 1)  {  // printf("%f\n", buf);  *float* \*p = static\_cast<*float* \*>(malloc(sizeof(*float*)));  \*p = buf;  mainList.add(p);  }  fclose(myfile);  mainList.print();  //------------------------------------  mainList.modifyList();  //------------------------------------  mainList.print();  puts("-----------------------------------");  Deque firstDeque;  Deque secondDeque;  List dequeList;  modifyDeque(&firstDeque, &secondDeque, &mainList);  createListFromDequeu(&firstDeque, &secondDeque, &dequeList);  //------------------------------------  return 0;  }  *void* modifyDeque(Deque \*firstDeque, Deque \*secondDeque, List \*mainlist)  {  for (*int* i = 0; i < mainlist->size(); i++)  {  *float* buf = \*(*float* \*)(\*mainlist)[i];  if (i % 2 == 0)  {  firstDeque->pushFront(buf);  }  else  {  secondDeque->pushBack(buf);  }  }  firstDeque->print();  secondDeque->print();  }  *void* createListFromDequeu(Deque \*firstDeque, Deque \*secondDeque, List \*dequeList)  {  while (!firstDeque->isEmpty())  {  *float* \*buf = static\_cast<*float* \*>(malloc(sizeof(*float*)));  // float p = Deque\_popBack(firstDeque);  \*buf = firstDeque->popBack();  dequeList->add(buf);  }  while (!secondDeque->isEmpty())  {  *float* \*buf = static\_cast<*float* \*>(malloc(sizeof(*float*)));  \*buf = secondDeque->popFront();  dequeList->add(buf);  }  dequeList->print();  } |

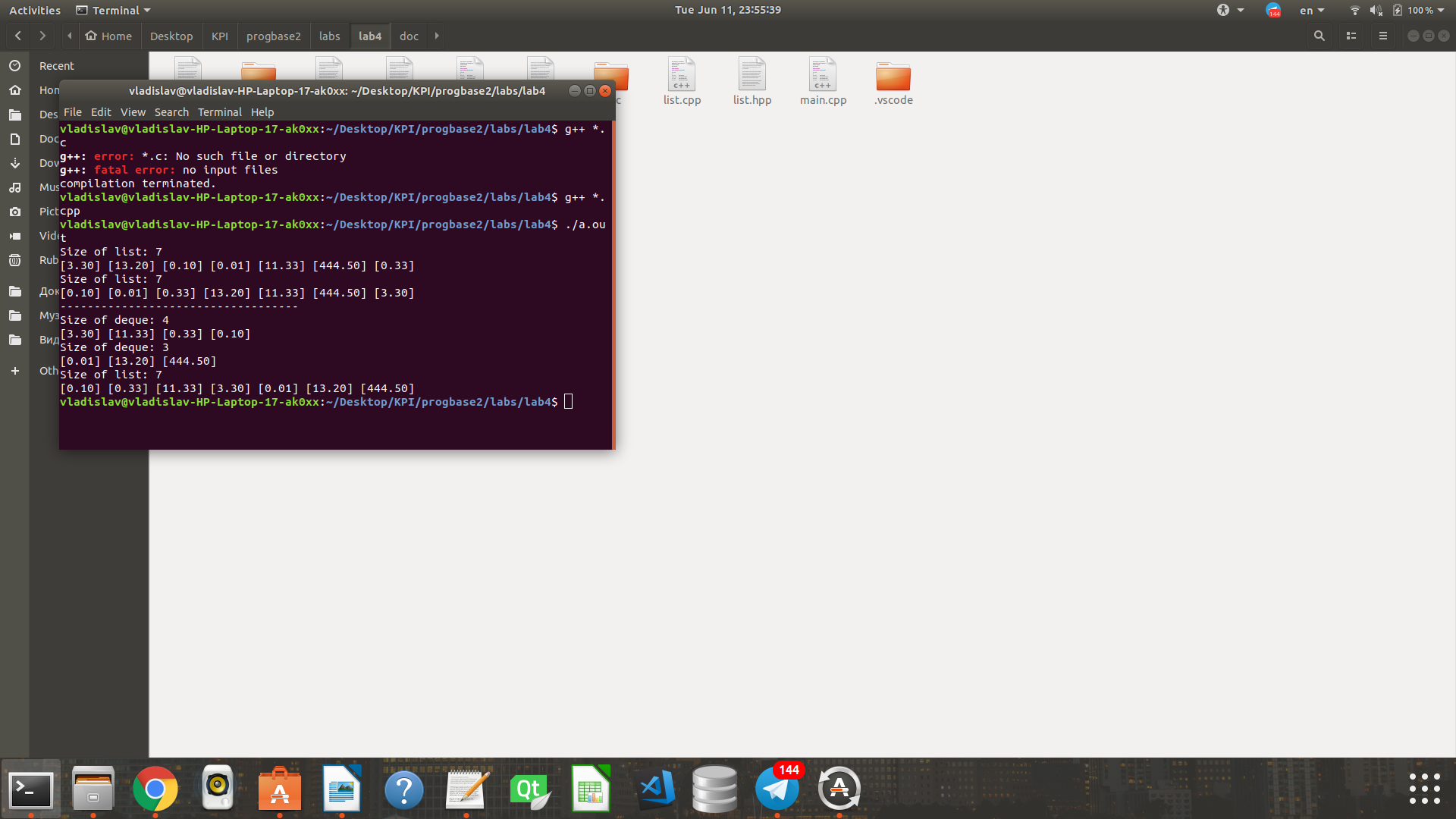
|  |
| --- |
| **list.hpp** |
| #pragma once  #include <cstdlib>  #include <cstdio>  *class* List  {  *void* \*\*items\_;  *int* size\_;  *int* capacity\_;  public:  List();  ~List();  *int* size();  *void* insert(*int* index, *void* \*value);  *void* removeAt(*int* index);  *void* add(*void* \*value);  *void* remove(*void* \*value);  *bool* isEmpty();  *void* print();  *void* modifyList();  *void* \* & operator [](*int* index); // TItem - item type  }; |

|  |
| --- |
| List.cpp |
| #include "list.hpp"  *void* List::print()  {  printf("Size of list: %i\n", size());  for (*int* i = 0; i < size(); i++)  {  printf("[%.2f] ", \*((*float* \*)this->items\_[i]));  }  puts("");  }  List::List()  {  this->capacity\_ = 30;  this->size\_ = 0;  this->items\_ = static\_cast<*void* \*\*>(malloc(sizeof(*void* \*) \* this->capacity\_));  }  List::~List()  {  delete this->items\_;  }  *void* \*&List::operator[](*int* index)  {  if (index < 0 || index > this->size())  {  abort();  }  return this->items\_[index];  }  *int* List::size()  {  return this->size\_;  }  *void* List::insert(*int* index, *void* \*value)  {  if (index > this->size\_ || index < 0)  {  fprintf(stderr, "ERROR: item cannot be inserted\n");  abort();  }  else if (index == this->size\_)  {  add(value);  }  else if (index < this->size\_)  {  if (this->size\_ + 1 < this->capacity\_)  {  *int* newcap = this->capacity\_ + 1;  *void* \*newitems\_ = realloc(this->items\_, sizeof(*void* \*) \* newcap);  if (newitems\_ == NULL)  {  free(this->items\_);  fprintf(stderr, "Reallocating memory error\n");  abort();  }  this->items\_ = static\_cast<*void* \*\*>(newitems\_);  this->capacity\_ = newcap;  }  for (*int* i = this->size\_; i >= index; i--)  {  this->items\_[i + 1] = this->items\_[i - 1];  }  this->items\_[index] = value;  this->size\_++;  }  }  *void* List::removeAt(*int* index)  {  if (index >= this->size\_ || index < 0)  {  printf("ERROR: Index is bigger than list's size\n");  return;  }  free(this->items\_[index]);  for (*int* i = index; i < this->size\_ - 1; i++)  {  this->items\_[i] = this->items\_[i + 1];  }  this->size\_--;  }  *void* List::add(*void* \*value)  {  this->items\_[this->size\_] = value;  this->size\_ += 1;  if (this->size\_ == this->capacity\_)  {  *int* newcap = this->capacity\_ \* 2;  *void* \*newitems\_ = realloc(this->items\_, sizeof(*void* \*) \* newcap);  if (newitems\_ == NULL)  {  free(this->items\_);  fprintf(stderr, "Reallocating memory error\n");  abort();  }  this->items\_ = static\_cast<*void* \*\*>(newitems\_);  ;  this->capacity\_ = newcap;  }  }  *void* List::remove(*void* \*value)  {  for (*int* i = 0; i < this->size\_; i++)  {  if (this->items\_[i] == value)  {  free(this->items\_[i]);  for (*int* j = i; j < this->size\_; j++)  {  if (j != this->size\_ - 1)  {  this->items\_[j] = this->items\_[j + 1];  }  }  this->size\_--;  return;  }  }  printf("ERROR: value that you want to remove isn't in list\n");  }  *bool* List::isEmpty()  {  return this->size\_ == 0;  }  *void* List::modifyList()  {  *int* tmp = 0;  for (*int* i = 0; i < size(); i++)  {  if (\*((*float* \*)this->items\_[i]) > -1 && \*((*float* \*)this->items\_[i]) < 1)  {  *float* \*buf = static\_cast<*float* \*>(this->items\_[tmp]);  this->items\_[tmp] = this->items\_[i];  this->items\_[i] = buf;  tmp++;  }  }  } |

|  |
| --- |
| **deque.hpp** |
| #pragma once  #include <cstdlib>  #include <cstdio>  *class* Deque  {  *float* \*items\_;  *int* capacity\_;  *int* first\_;  *int* last\_;  public:  Deque();  ~Deque();  *void* pushBack(*float* value);  *float* popBack();  *void* pushFront(*float* value);  *float* popFront();  *int* size();  *bool* isEmpty();  *void* print();  }; |

|  |
| --- |
| **deque.cpp** |
| #include "deque.hpp"  Deque::Deque()  {  this->capacity\_ = 20;  this->first\_ = 10;  this->last\_ = 10;  this->items\_ = static\_cast<*float* \*>(malloc(sizeof(*float*) \* this->capacity\_));  }  Deque::~Deque()  {  delete this->items\_;  }  *void* Deque::pushFront(*float* value)  {  if (this->first\_ == -1)  {  this->first\_ = 0;  this->last\_ = 0;  }  else if (this->first\_ == 0 || this->last\_ == this->capacity\_ - 1)  {  if (this->last\_ + 1 < this->capacity\_)  {  *int* newCap = this->capacity\_ + 1;  *void* \*newItems = realloc(this->items\_, sizeof(*float*) \* newCap);  if (newItems == NULL)  {  free(this->items\_);  fprintf(stderr, "Reallocating memory error\n");  abort();  }  this->items\_ = static\_cast<*float* \*>(newItems);  this->capacity\_ = newCap;  }  for (*int* i = this->last\_; i >= this->capacity\_; i--)  {  this->items\_[i + 1] = this->items\_[i - 1];  }  this->items\_[this->last\_] = value;  this->last\_++;  }  else  {  this->first\_--;  }  this->items\_[this->first\_] = value;  }  *void* Deque::pushBack(*float* value)  {  this->items\_[this->last\_] = value;  this->last\_ += 1;  if (this->last\_ == this->capacity\_)  {  this->last\_ = 0;  }  if (this->last\_ == this->first\_)  {  fprintf(stderr, "Deque is full");  *int* newCap = this->capacity\_ \* 2;  *float* \*newItems = static\_cast<*float* \*>(realloc(this->items\_, sizeof(*float*) \* newCap));  if (newItems == NULL)  {  free(this->items\_);  fprintf(stderr, "Reallocating memory error\n");  abort();  }  this->items\_ = newItems;  this->capacity\_ = newCap;  }  }  *float* Deque::popFront()  {  *float* value = this->items\_[this->first\_];  this->items\_[this->first\_] = -1;  if (this->first\_ == this->last\_)  {  this->first\_ = -1;  this->last\_ = -1;  }  else if (this->first\_ == this->capacity\_ - 1)  {  this->first\_ = 0;  }  else  {  this->first\_++;  }  return value;  }  *float* Deque::popBack()  {  *float* value = this->items\_[this->last\_ - 1];  this->items\_[this->last\_] = -1;  if (this->first\_ == this->last\_)  {  this->first\_ = -1;  this->last\_ = -1;  }  else if (this->last\_ == 0)  {  this->last\_ = this->capacity\_ - 1;  }  else  {  this->last\_--;  }  return value;  }  *int* Deque::size()  {  if (this->last\_ >= this->first\_)  return this->last\_ - this->first\_;  return this->capacity\_ - this->first\_ + this->last\_;  }  *bool* Deque::isEmpty()  {  if (this->first\_ == this->last\_)  {  return true;  }  return false;  }  *void* Deque::print()  {  printf("Size of deque: %i\n", size());  for (*int* i = this->first\_; i < this->last\_; i++)  {  printf("[%.2f] ", this->items\_[i]);  }  puts(""); |

**Приклади результатів**

****

**Висновки**

Виконавши дану лабораторну роботу ми навчилися вивчити основні відмінності між кодом мови С та мови С++, навчились створювати С++ класи для реалізації базових АТД.

Програма компілювалася за допомогою компілятора g++.