МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
ФАКУЛЬТЕТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ НАУК ИМЕНИ
ПРОФЕССОРА Н.И.ЧЕРВЯКОВА

ЛАБОРАТОРНАЯ РАБОТА №21

Алгоритмизация и программирование

Классы

Выполнил студент:
Сивко Иван Андреевич
студент 2 курса
группа ПМИ-б-о-23-2,
направление подготовки 01.03.02

Проверил:
Ассистент кафедры вычислительной
математики и кибернетики, к.ф.-м.н.,
Черкашина Анастасия Андреевна

2024 г.

Вариант 9

Цель:

- Совершенствование навыков разработки программ в среде программирования MS Visual C++
- Совершенствование навыков в программировании с использованием модульного подхода

Задание 1

1 Условие:

Составить описание класса для определения одномерных массивов строк фиксированной длины. Предусмотреть следующие возможности:

- обращение к отдельным строкам массива по индексам;
- контроль выхода за пределы массива;
- выполнение операций поэлементного сцепления двух массивов с образованием нового массива;
- слияния двух массивов с исключением повторяющихся элементов;
- вывод на экран элемента массива по заданному индексу и всего массива.

Написать программу, демонстрирующую работу с этим классом.

2 Алгоритм / Мат. модель

Программа использует класс, предназначенный для работы с одномерными массивами строк фиксированной длины. Класс поддерживает различные операции с массивами строк, такие как обращение к отдельным строкам, сцепление массивов, слияние с исключением дубликатов и вывод на экран. Все операции с массивами контролируют выход за пределы массива и выполняют необходимые действия для предотвращения ошибок. Программа демонстрирует работу с этим классом и позволяет эффективно управлять строками в массиве.

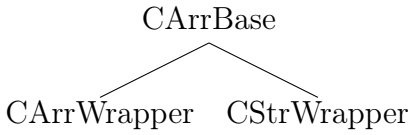
1) **Инициализация структуры данных:** Создается класс `CArrWrapper`, который инкапсулирует массив строк. Каждое строковое поле представляет собой объект `CStrWrapper`, хранящий строку.
   - `size_` — количество элементов в массиве (строк),
   - `data` — указатель на массив строк типа `T*`,
   - `CArrBase<T>` — базовый класс, предоставляющий общие методы для работы с массивом: доступ к элементам по индексу, операции с памятью.
2) **Обращение к строкам массива:** Массив строк поддерживает доступ к отдельным элементам по индексу с помощью оператора `[]`. При попытке доступа к несуществующему элементу генерируется исключение.
3) **Контроль выхода за пределы массива:** Для предотвращения выхода за пределы массива, при обращении к элементам используется проверка `assert(idx < size_)` для гарантии, что индекс находится в пределах допустимого диапазона.
4) **Сцепление двух массивов:** Операция сцепления двух массивов строк выполняется через оператор `+=`, что приводит к созданию нового массива, содержащего элементы обоих исходных массивов. При этом формируется новый объект класса `CStrWrapper`, в который добавляются элементы из обоих массивов.

5) **Слияние массивов с исключением повторяющихся элементов:** Функция `mergeUnique` сливает два массива строк, исключая повторяющиеся элементы. Для каждого элемента второго массива осуществляется проверка наличия его в первом массиве. Если элемент не найден, он добавляется в новый массив.

6) **Вывод массива:** Функция `operator«` выводит все элементы массива на экран в виде списка, разделенного запятыми. Каждый элемент массива отображается по порядку.

7) **Реализация через командную строку:** Программа создает два массива строк, выполняет их слияние с исключением повторов, а затем выводит результат на экран.

Дерево наследования



Член класса	**Описание**
size_	Размер массива (количество элементов)
data	Указатель на массив данных (тип T*)
Метод класса	**Описание**
CArrBase(size_t, T*)	Конструктор, инициализирующий размер и данные массива
CArrBase()	Конструктор по умолчанию (неинициализированный массив)
CArrBase(size_t)	Конструктор, выделяющий память для массива указанного размера
~CArrBase()	Деструктор, освобождает память массива
operator=(CArrBase&&)	Оператор перемещения, присваивает данные из другого объекта
operator[](size_t)	Оператор индексации для доступа к элементам массива (по ссылке)
operator[](size_t) const	Оператор индексации для доступа к элементам массива (константный)
size()	Метод, возвращающий размер массива (не используется в вашем примере)
begin()	Метод, возвращающий указатель на начало массива
end()	Метод, возвращающий указатель на конец массива

Таблица 1: Методы класса CArrBase<T>

Член класса	**Описание**
size_	Размер массива (количество элементов)
data	Указатель на массив данных (тип T*)
Метод класса	**Описание**
CArrWrapper(std::initializer_list<T>)	Конструктор, инициализирует массив из списка инициализации
CArrWrapper(const CArrWrapper&)	Конструктор копирования
CArrWrapper(CArrWrapper&&)	Конструктор перемещения
operator=(const CArrWrapper&)	Оператор присваивания (копирование)
operator=(CArrWrapper&&)	Оператор присваивания (перемещение)
push_back(const T&)	Метод для добавления элемента в конец массива

Таблица 2: Методы класса CArrWrapper<T>

Член класса	**Описание**
size_	Размер строки (количество символов)
data	Указатель на строку данных (тип char*)
Метод класса	**Описание**
CStrWrapper(size_t)	Конструктор, инициализирует строку указанного размера
CStrWrapper(const CStrWrapper&)	Конструктор копирования
operator=(const CStrWrapper&)	Оператор присваивания (копирование)
CStrWrapper(const char*)	Конструктор, инициализирует строку из C-строки
operator+=(const CStrWrapper&)	Оператор сложения для строк (конкатенация)

Таблица 3: Методы класса CStrWrapper

3 Код:

- arr_base.hpp

```
#pragma once
#include <cstdlib>
#include <cassert>

template <typename T>
class CArrBase {
protected:
    size_t size_{ 0 };
    T* data { nullptr };

    CArrBase(size_t size, T* data);
public:
    CArrBase() = default;
    CArrBase(size_t size);
    ~CArrBase();
    CArrBase& operator=(CArrBase&& other) noexcept;
    const T& operator[](size_t idx) const;
    T& operator[](size_t idx);
    size_t size() const;
    const T *begin() const;
    const T *end() const;
};
```

- arr_base.cpp

```
#include "arr_base.hpp"
#include "str.hpp"

template <typename T>
CArrBase<T>::CArrBase(size_t size, T* data)
    : size_{ size }, data{ data }
{ }

template <typename T>
CArrBase<T>::CArrBase(size_t size)
    : CArrBase( size, size > 0? new T[size] : nullptr)
{ }
```

```
template <typename T>
CArrBase<T>::~CArrBase() {
    delete[] data;
}

template <typename T>
CArrBase<T>& CArrBase<T>::operator=(CArrBase&& other) noexcept {
    if (this == &other)
        return *this;
    delete[] data;
    size_ = other.size_;
    data = other.data;
    other.size_ = 0;
    other.data = nullptr;
    return *this;
}

template <typename T>
const T& CArrBase<T>::operator[](size_t idx) const {
    assert(idx < size_);
    return data[idx];
}

template <typename T>
T& CArrBase<T>::operator[](size_t idx) {
    assert(idx < size_);
    return data[idx];
}

template <typename T>
size_t CArrBase<T>::size() const {
    return size_;
}

template <typename T>
const T *CArrBase<T>::begin() const {
    return data;
}

template <typename T>
const T *CArrBase<T>::end() const {
    return data+size_;
}

template class CArrBase<char>;
template class CArrBase<CStrWrapper>;
```

- arr.hpp

```
#pragma once
#include <cstdlib>
#include <iostream>
#include <initializer_list>
```

```
#include "arr_base.hpp"
#include "str.hpp"

template <typename T>
class CArrWrapper : public CArrBase<T> {
protected:
    using CArrBase<T>::size_;
    using CArrBase<T>::data;
public:
    using CArrBase<T>::CArrBase;

    CArrWrapper(std::initializer_list<T> iniList);
    CArrWrapper(const CArrWrapper& other);
    CArrWrapper(CArrWrapper&& other) noexcept;
    CArrWrapper& operator=(const CArrWrapper& other);
    CArrWrapper& operator+=(const T& el);
};

template <typename T>
constexpr T max(T a, T b) {
    return a > b ? a : b;
}

template <typename T>
constexpr T min(T a, T b) {
    return a < b ? a : b;
}

template <typename T>
std::ostream& operator<<(std::ostream& out, const CArrWrapper<T>& arr);

CArrWrapper<CStrWrapper> zip(const CArrWrapper<CStrWrapper>& arr1, const CArrWrapper<C

CArrWrapper<CStrWrapper> mergeUnique(const CArrWrapper<CStrWrapper>& arr1, const CArrW
```

- arr.cpp

```
#include "arr.hpp"
#include "str.hpp"

template <typename T>
CArrWrapper<T>::CArrWrapper(std::initializer_list<T> iniList)
    : CArrBase<T>( iniList.size() )
{
    size_t i{0};
    for (const auto& item : iniList)
        (*this)[i++] = item;
}

template <typename T>
CArrWrapper<T>::CArrWrapper(const CArrWrapper& other)
```

```
    : CArrBase<T>( other.size_ )
{
    for (size_t i{0}; i<size_; ++i)
        (*this)[i] = other[i];
}

template <typename T>
CArrWrapper<T>::CArrWrapper(CArrWrapper&& other) noexcept
    : CArrBase<T>( other.size_,  other.data )
{
    other.size_ = 0;
    other.data = nullptr;
}

template <typename T>
CArrWrapper<T>& CArrWrapper<T>::operator=(const CArrWrapper& other) {
    delete[] data;
    size_ = other.size_;
    data = new T[size_];
    size_t i{0};
    for (const T& el : other)
        (*this)[i++] = el;
    return *this;
}

template <typename T>
CArrWrapper<T>& CArrWrapper<T>::operator+=(const T& el) {
    CArrWrapper res(size_ + 1);
    size_t i{0};
    for (; i<size_; ++i)
        res[i] = (*this)[i];
    res[size_] = el;
    return *this = std::move(res);
}

template <typename T>
std::ostream& operator<<(std::ostream& out, const CArrWrapper<T>& arr) {
    bool isFirst{ true };
    for (const T& item : arr) {
        out << (!isFirst ? ", " : "") << item;
        isFirst = false;
    }
    return out;
}

CArrWrapper<CStrWrapper> zip(const CArrWrapper<CStrWrapper>& arr1, const CArrWrapper<C
    CArrWrapper<CStrWrapper> res;
    const CArrWrapper<CStrWrapper> *add;
    if (arr1.size() == arr2.size()) {
        res =  arr1;
        add = &arr2;
    } else {
```

```
        res  = max(arr1.size(), arr2.size()) == arr1.size() ? arr1 : arr2;
        add = ((min(arr1.size(), arr2.size()) == arr1.size()) ? &arr1 : &arr2);
    }
    for (size_t i{0}; i < add->size(); ++i)
        res[i] += (*add)[i];
    return res;
}

CArrWrapper<CStrWrapper> mergeUnique(const CArrWrapper<CStrWrapper>& arr1, const CArrW
    CArrWrapper<CStrWrapper> res{arr1};
    for (const auto& elCopy : arr2) {
        bool unique{ true };
        for (const auto& elRes : res)
            if (elCopy == elRes) {
                unique = false;
                break;
            }
        if (unique)
            res += elCopy;
    }
    return res;
}

template class CArrWrapper<CStrWrapper>;
template std::ostream& operator<<(std::ostream& out, const CArrWrapper<CStrWrapper>& a
```

- str.hpp

```
#pragma once
#include <cstdlib>
#include <iostream>

#include "arr_base.hpp"

class CStrWrapper : public CArrBase<char> {
protected:
    static constexpr size_t CStrLen(const char* str) {
        if (str==nullptr)
            return 0;
        size_t i{0};
        for (; str[i]!='\0'; ++i)
            ;
        return i;
    }
public:
    using CArrBase<char>::CArrBase;
    CStrWrapper(size_t iniSize);
    CStrWrapper(const CStrWrapper& other);
    CStrWrapper& operator=(const CStrWrapper& other);
    CStrWrapper(const char* str);
    CStrWrapper& operator+=(const CStrWrapper& other);
};
```

```
std::ostream& operator<<(std::ostream& out, const CStrWrapper& str);
CStrWrapper operator+(const CStrWrapper& str1, const CStrWrapper& str2);
bool operator==(const CStrWrapper& str1, const CStrWrapper& str2);
```

- main.cpp

```
#include <cstdlib>
#include <cstdint>
#include <iostream>

#include "arr.hpp"
#include "str.hpp"


int main(int32_t, const char**) {
    std::cout <<
        mergeUnique(zip( CArrWrapper<CStrWrapper>{ "Hello", "World", "All" }, { "Foo",
                    {"All", "WorldBar", "Foo", "All"}) << '\n';
    return EXIT_SUCCESS;
}
```

src: arr_base.hpp src: arr_base.cpp src: arr.hpp src: arr.cpp src: str.hpp src: str.cpp src: main.cpp

4 Результат работы программы:

```
$ ./a.out
HelloFoo, WorldBar, All, Foo
```