МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И КИБЕРНЕТИКИ

**КУРСОВАЯ РАБОТА**

по дисциплине «Объектно-ориентированное программирование»

Тема: «Объект строка»

Выполнил: студент группы ИС-142

Григорьев Ю.В.

Проверил: доцент кафедры ПМиК

Ситняковская Е.И.

Новосибирск – 2022

**Содержание**

## Постановка задачи

В реализуемом задании требуется создать собственный класс для работы с типом данных «строка» (инкапсулирующий методы для работы со строками) с использованием объектно-ориентированных технологий.

Описания объектов и методов необходимо оформить в отдельном модуле.

Необходимый минимум содержания работы:

1. Инкапсуляция (все поля данных не доступны из внешних функций)
2. Наследование (минимум 3 класса, один из которых - абстрактный)
3. Полиморфизм
4. Конструкторы, Перегрузка конструкторов
5. Списки инициализации

Также желательно использование как минимум ещё 2 технологий ООП (статические элементы, дружественные функции, классы, виртуальные функции, шаблоны, множественное наследование, массивы указателей на объекты, конструкторы копирования, параметры по умолчанию, использование объектов в качестве аргументов или возвращаемых значений).

Отчет должен содержать:

1. Постановка задачи (вариант курсовой работы)
2. Описание алгоритма основной программы (желательно)
3. Текст программы и модуля (если классы оформлены в отдельном модуле)
4. Скриншоты работы программы (в нескольких экземплярах).

## Технологии ООП

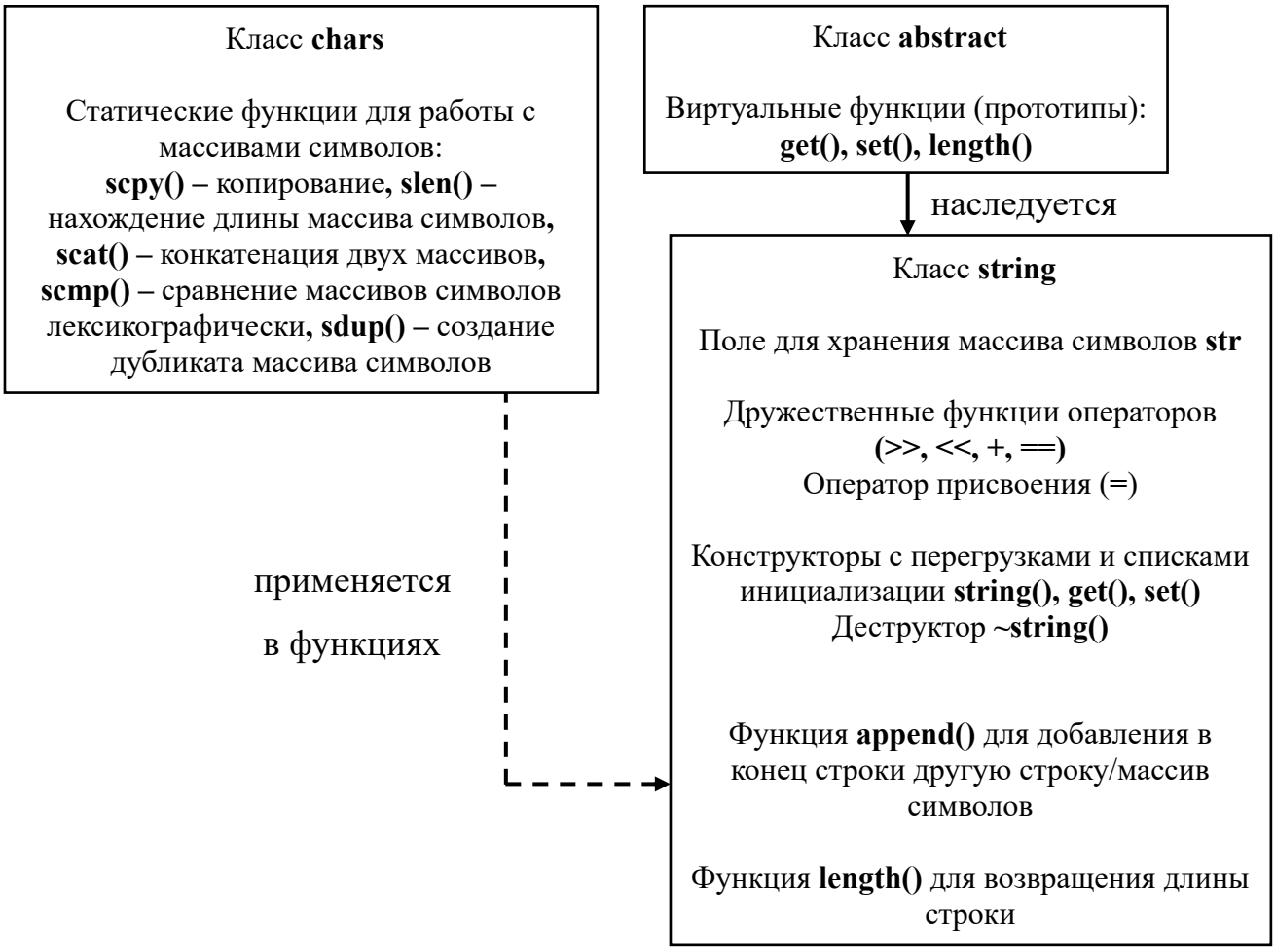
Работа реализована с помощью следующих технологий/методологий объектно-ориентированного программирования:

1. Инкапсуляция (все поля данных не доступны из внешних функций)
2. Наследование (3 класса, один из которых абстрактный)
3. Полиморфизм

4. Конструкторы, перегрузка конструкторов
5. Списки инициализации
6. Параметры по умолчанию
7. Статические элементы (функции)
8. Дружественные функции
9. Виртуальные функции
10. Использование объектов в качестве аргументов и возвращаемых значений функций

## Структура классов

Структура классов и их назначение показаны на следующей диаграмме:



## Программная реализация

Для выполнения данной работы я сделал набросок тестовой программы в файле **main.cpp**, чтобы проверять возможности для работы со своим классом «строка». Первое, что требовалось реализовать – инициализацию строк с помощью конструкторов тремя способами – по символьному массиву, размеру будущей строки и по другой строке (копирование). Во втором случае (инициализация по размеру строки) применяется обработка исключений, при которой проверяется введённый размер строки. Если он меньше или равен 0, обрабатывается исключение, предупреждающее, что размер строки должен быть больше нуля. В первом и третьем и случаях я сделал это с помощью списков инициализации в главном классе строки, во втором — последовательным заполнением поля **str** (в главном классе **string**) завершающими нулями. В методах главного класса (**string**) используются преимущественно статические функции из другого класса — **chars**, для работы с символами. chars::slen() – для получения длины строки в методе string::length(), chars::scpy() – копирования одной символьной последовательности в другую для методов string::set() / string::append(), chars::scat() – конкатенации двух символьных массивов для метода string::append() и *оператора +,* chars::scmp() – лексикографического сравнения двух символьных массивов для *оператора* ==, chars::sdup() – получения дубликата вводимой строки, который нужен преимущественно для конвертации **const char*** в **char***, что требуется для записи в поле str символьной последовательности. Также в главном классе имеются дружественные функции для работы с операторами ввода и вывода (>> и << соответственно). *Оператор присвоения* (=) реализован на базе метода string::set(), который присваивает строке некоторую символьную последовательность. Абстрактный класс **abstract** создан для создания чистых виртуальных функций abstract::get(), abstract::set() и abstract::length(). Метод string::get() используется для получения чистого символьного массива вместо

базового класса строки. Деструктор ~string() в базовом классе последовательно удаляет все символы строки (в поле **str**), если **str** не равен **NULL**. Пустой конструктор строки ( **string()** ) существует для инициализации пустой строки (состоящей из одного завершающего нуля). В тестовой программе прорабатываются все сценарии использования строки, сложение строк, создание пустой строки, замена строковых ключей.

## Результат работы

Результаты работы можно увидеть на следующем скриншоте:

```
s1 =
s1 = hello
s2 = world
s3 = !!!
s3 = ...
(s1 == s2) = 0 (not equal)
s1 + s2 + s3 = hello world...
s2.append(' is ').append('great!') = world is great!
```

## Заключение

Выполнив данную работу, я смог разработать собственную реализацию класса «строка» и методы для работы с ней, применив и закрепив знания из изученного курса объектно-ориентированного программирования.

## Используемые источники


1. Лафоре Р. Объектно-ориентированное программирование в C++
2. Бьерн Страуструп. Язык программирования C++
3. Бертран Мейер. Почувствуй класс. Учимся программировать хорошо с объектами и контрактами
4. Гради Буч. Объектно-ориентированный анализ и проектирование с примерами приложений
5. Мэтт Вайсфельд. Объектно-ориентированное мышление

## Приложение. Листинг

main.cpp

```cpp
#include "string.hpp"
#include <iostream>

int main()
{
    string s1("");
    std::cout << "s1 = " << s1 << std::endl;

    s1.set("hello");
    std::cout << "s1 = " << s1 << std::endl;

    string s2("world");
    std::cout << "s2 = " << s2 << std::endl;

    string s3("!!!");
    std::cout << "s3 = " << s3 << std::endl;

    s3 = "...";
    std::cout << "s3 = " << s3 << std::endl;

    std::cout << "(s1 == s2) = " << std::noboolalpha << (s1 == s2) << std::boolalpha << " (not equal)" << std::endl;

    std::cout << "s1 + s2 + s3 = " << s1 + " " + s2 + s3 << std::endl;

    s2.append(" is ");
    std::cout << "s2.append(' is ').append('great!') = " << s2.append("great!") << std::endl;
    exit(0);
}
```

string.hpp

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

class abstract
{
public:
    virtual char *get() const = 0;
    virtual void set(char *val) = 0;
    virtual size_t length() const = 0;
};

class chars
{
public:
    static char *scpy(char *dest, const char *src)
    {
        assert(dest != NULL && src != NULL);
        char *temp = dest;
        while ((*dest++ = *src++) != '\0')
        {
        }
        return temp;
    }
```

```c
static unsigned int slen(const char *str)
{
    unsigned int count = 0;
    while (*str != '\0')
    {
        count++;
        str++;
    }
    return count;
}
static char *scat(char *dest, const char *src)
{
    char *rdest = dest;
    while (*dest)
    {
        dest++;
    }
    while ((*dest++ = *src++))
    {
    }
    return rdest;
}
```

```c
static int scmp(const char *s1, const char *s2)
{
    char c1, c2;
    while (1)
    {
        c1 = *s1++;
        c2 = *s2++;
        if (c1 > c2)
        {
            return 1;
        }
        else if (!c1)
        {
            break;
        }
        else
        {
            return -1;
        }
    }
    return 0;
}
```

```
68      static char *sdup(const char *val)
69      {
70          char *dest = new char[slen(val) + 1];
71          if (dest != NULL)
72          {
73              scpy(dest, val);
74          }
75          return dest;
76      }
77  };
78
79  class string : public abstract
80  {
81  private:
82      char *str;
83      friend std::ostream &operator<<(std::ostream &os, const string &obj);
84      friend std::istream &operator>>(std::istream &is, string &obj);
85      friend string operator+(const string &s1, const string &s2);
86      friend bool operator==(const string &s1, const string &s2);
87
88  public:
89      string()
90      {
91          string("");
92      }
93      string(const char *val) : str(chars::sdup(val)) {}
94      string(size_t size = 1);
95      string(const string &s) : str(chars::sdup(s.get())) {}
96      char *get() const { return str; };
97      void set(const char *val)
98      {
99          set(chars::sdup(val));
100     }
101     void set(char *val)
102     {
103         delete[] str;
104         str = new char[chars::slen(val) + 1];
105         chars::scpy(str, val);
106     }
107     size_t length() const { return chars::slen(str); }
108     string append(const char *val);
109     string append(const string &s);
110     ~string()
111     {
112         if (str != NULL)
113         {
114             delete[] str;
115         }
116     }
117     string &operator=(const string &s);
118 };
```

string.cpp

```cpp
#include "string.hpp"
#include <iostream>

string::string(size_t size)
{
    try
    {
        if (size <= 0)
        {
            throw "Size of a string must be more than zero.";
        }
        else
        {
            str = new char[size];
            if (str != NULL)
            {
                for (size_t i = 0; i < size; i++)
                {
                    str[i] = '\0';
                }
            }
        }
    }
    catch (const char *e)
    {
        std::cout << e << std::endl;
    }
}

string string::append(const char *val)
{
    int l = length() + chars::slen(val);
    char *buff = new char[l + 1];
    chars::scpy(buff, str);
    chars::scat(buff, val);
    buff[l] = '\0';
    set(buff);
    delete[] buff;
    return *this;
}

string string::append(const string &s)
{
    append(s.get());
    return *this;
}
```

```cpp

string &string::operator=(const string &s)
{
    if (this == &s)
    {
        return *this;
    }
    set(s.get());
    return *this;
}

std::ostream &operator<<(std::ostream &os, const string &obj)
{
    return (os << obj.get());
}

std::istream &operator>>(std::istream &is, string &obj)
{
    char *buff = new char[1000];
    memset(&buff[0], 0, sizeof(buff));
    is >> buff;
    obj = string(buff);
    delete[] buff;
    return is;
}

string operator+(const string &s1, const string &s2)
{
    string temp(s1);
    temp.append(s2);
    return temp;
}

bool operator==(const string &s1, const string &s2)
{
    if (chars::scmp(s1.get(), s2.get()) != 0)
    {
        return false;
    }
    return true;
}
```

Makefile

```
all: main
.PHONY: main
main: main.cpp string.cpp
    g++ -fno-rtti -Wall -Wextra -o $@ $^

clean:
    rm -rf *.o main
```