Содержание

[Постановка задачи 3](#_TOC_250006)

[Технологии ООП 4](#_TOC_250005)

Структура классов 8

[Программная реализация 9](#_TOC_250004)

[Результаты работы 11](#_TOC_250003)

[Заключение 12](#_TOC_250002)

[Используемые источники 13](#_TOC_250001)

[Приложение. Текст программы 14](#_TOC_250000)

**Постановка задачи**

Необходимо описать объект строка, инкапсулирующий методы для работы со строками.

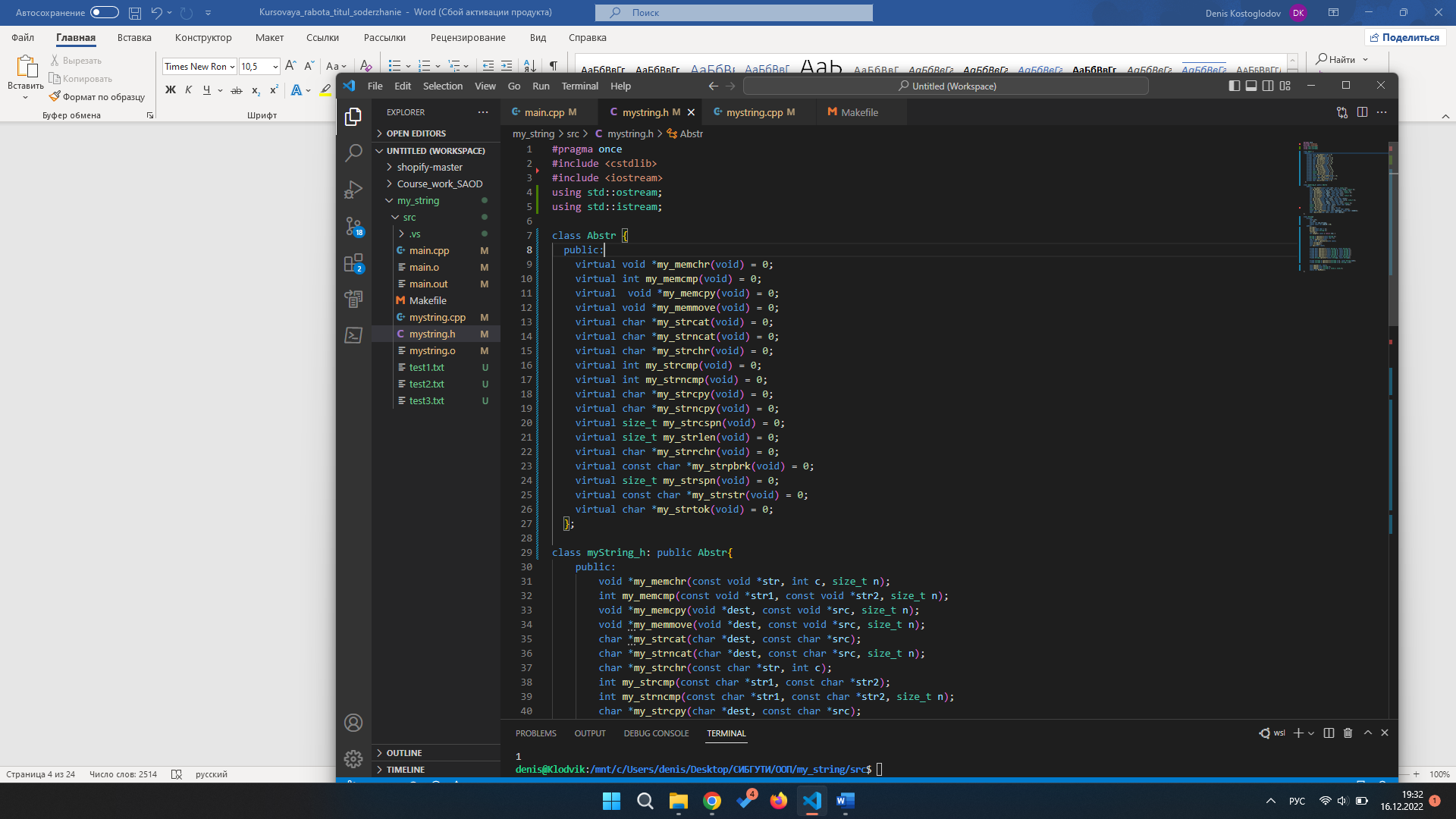
Программа должна быть написана с использованием объектно-ориентированных технологий. Описания объектов и методов необходимо оформить в отдельном модуле.

В программе будет описаны все основные методы и операции со строка.

**Технологии ООП**

Наследование — концепция объектно-ориентированного программирования, согласно которой абстрактный тип данных может наследовать данные и функциональность некоторого существующего типа, способствуя повторному использованию компонентов программного обеспечения.

Родительский класс (также он является абстрактным). (рис. 1).

  
Рисунок 1 – Родительский класс

Дочерний класс. (рис. 2).

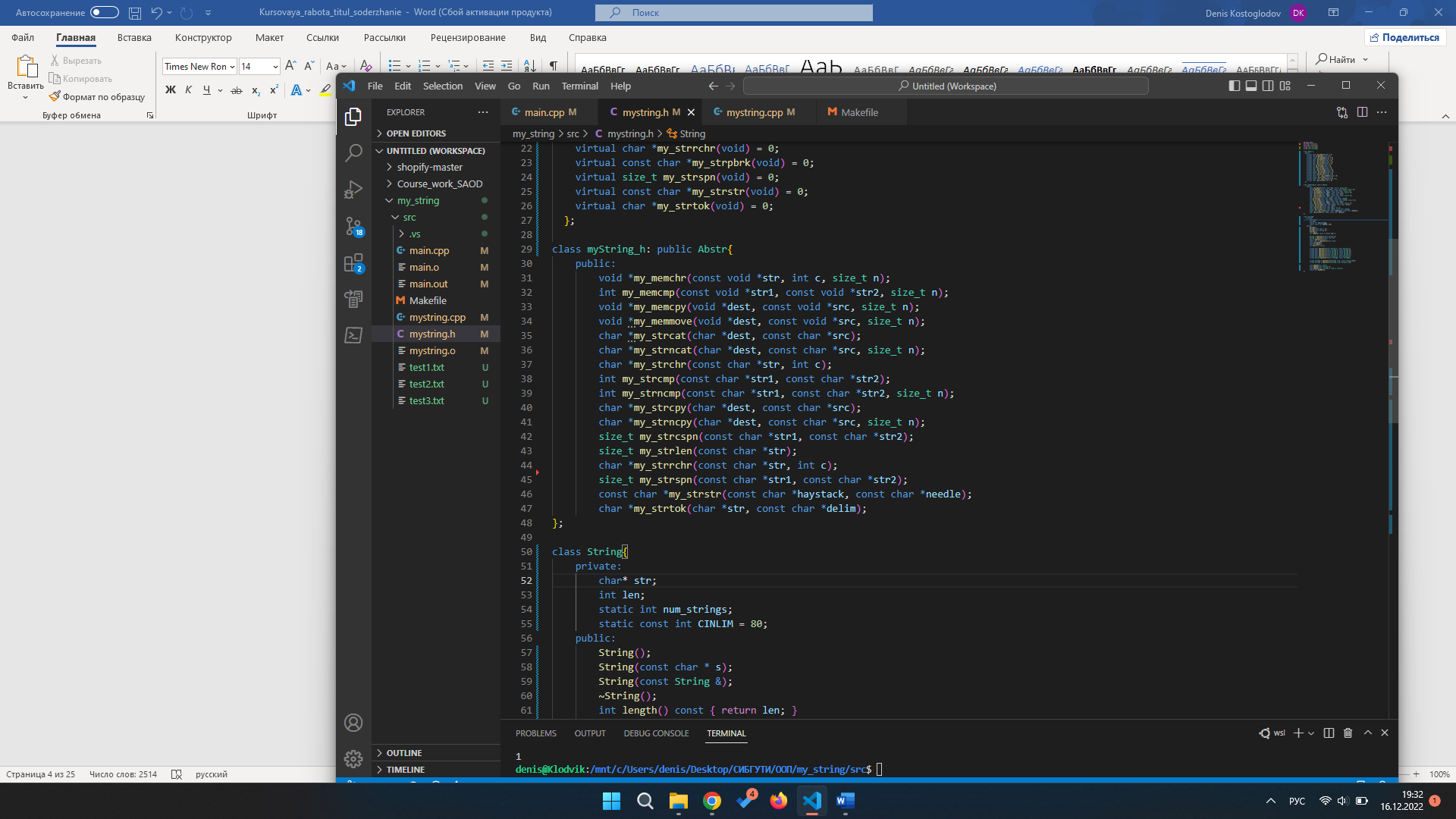


Рисунок 2 – Дочерний класс

Конструкторы и их перегрузка. Конструкторы – специальный блок инструкций, вызываемый при создании объекта. Перегрузка конструктора – один и тот же конструктор, но с разной сигнатурой. (рис. 3).

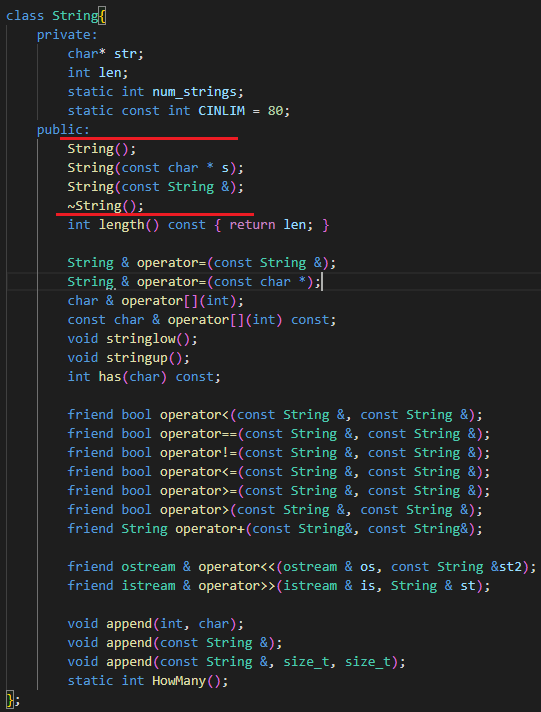


Рисунок 3 – Конструкторы

Списки инициализации – это свойство конструктора для определения переменных в классе. (рис. 4).

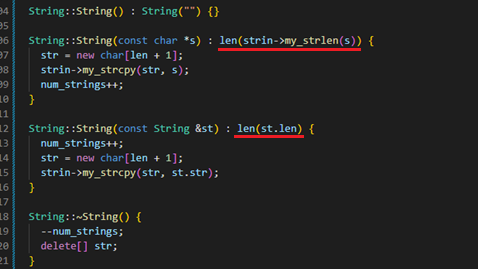


Рисунок 4 – Списки инициализации

Полиморфизм – это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта. В моём случае он реализован в виде перегрузок функций. (рис. 5).

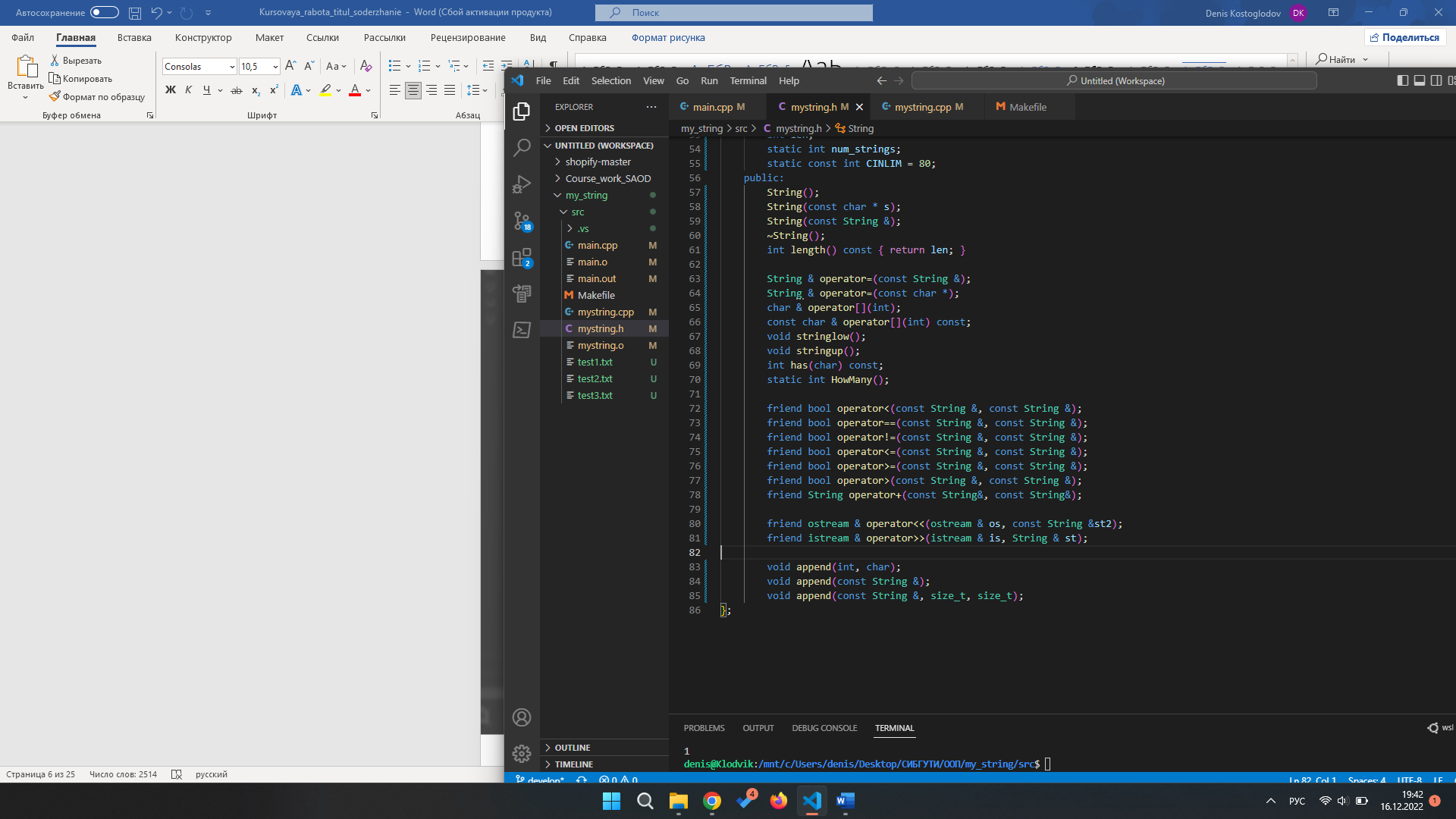


Рисунок 5 – Полиморфизм

Инкапсуляция – это поля, которые меняются или используются только внутри класса, т.е. доступна извне к ним нет (из экземпляра класса).

(рис. 6).

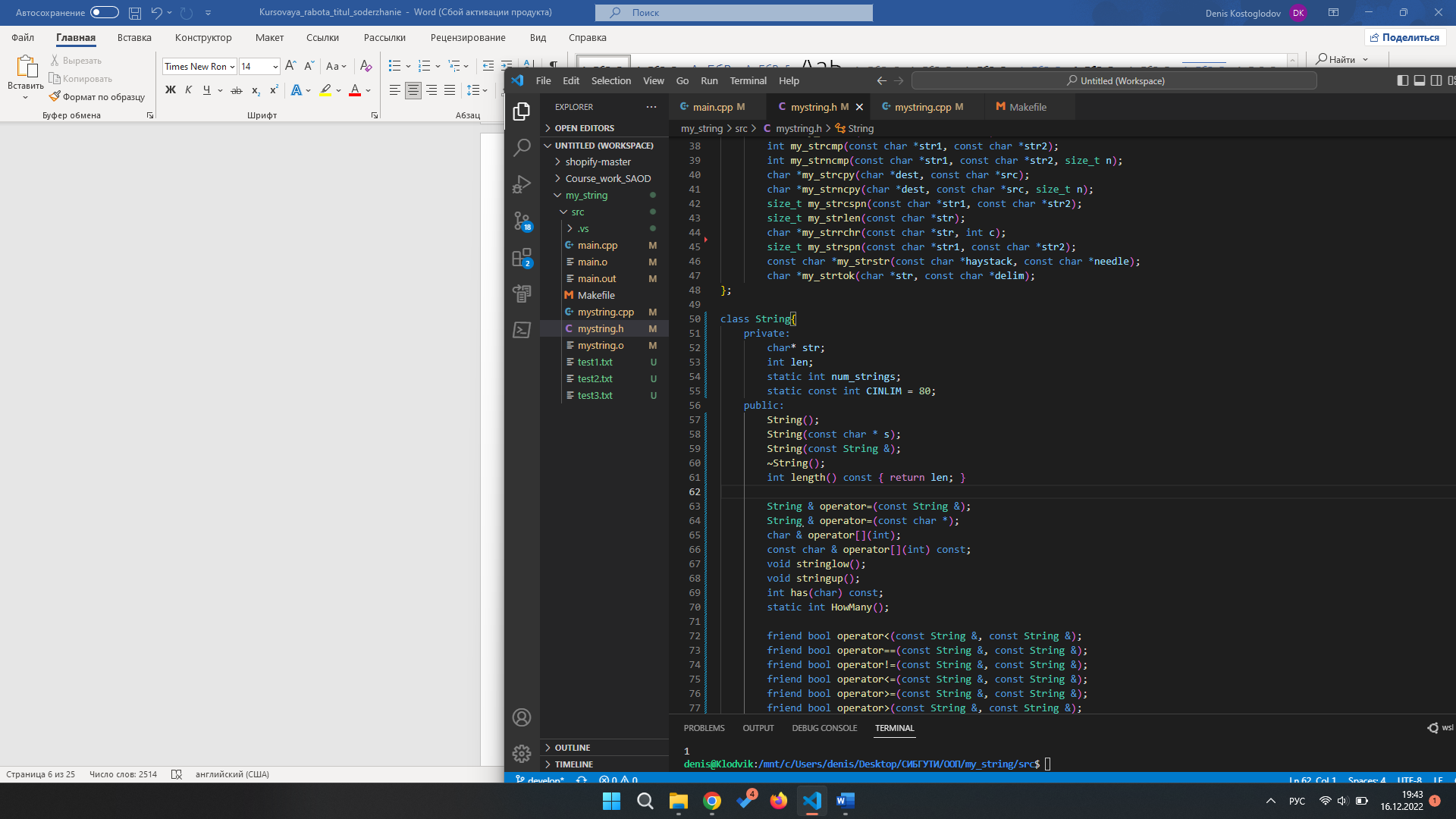


Рисунок 6 – Инкапсуляция

Дружественные функции класса – функции, которые не являются компонентом некоторого класса, имеющие доступ ко всем его компонентам. (рис. 7).

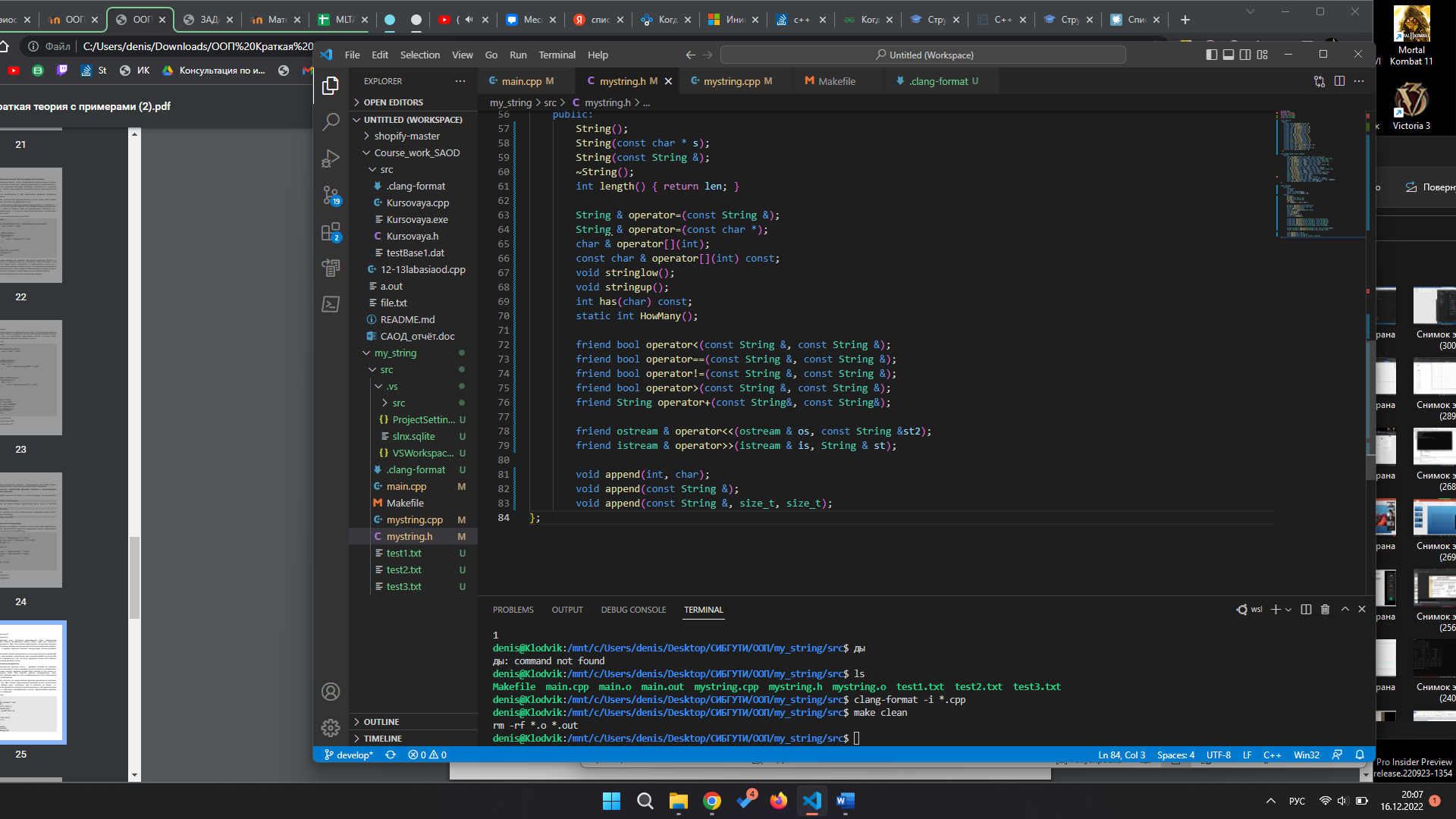


Рисунок 7 – Дружественная функция класса

**Структура классов**

Классы в С++ — это абстракция, которая описывает методы, свойства, еще не существующих объектов.

Объекты – конкретное представление абстракции, имеющие свои свойства и методы.

В ООП существует три основных принципа построения классов:

1. Инкапсуляция
2. Наследование
3. Полиморфизм

Спецификаторы доступа:

1. Public – это члены структуры или класса, к которым можно получить доступ извне структуры или класса (рис 8).
2. Private – это члены класса, к которым могут получить доступ только другие члены класса. (рис. 8).

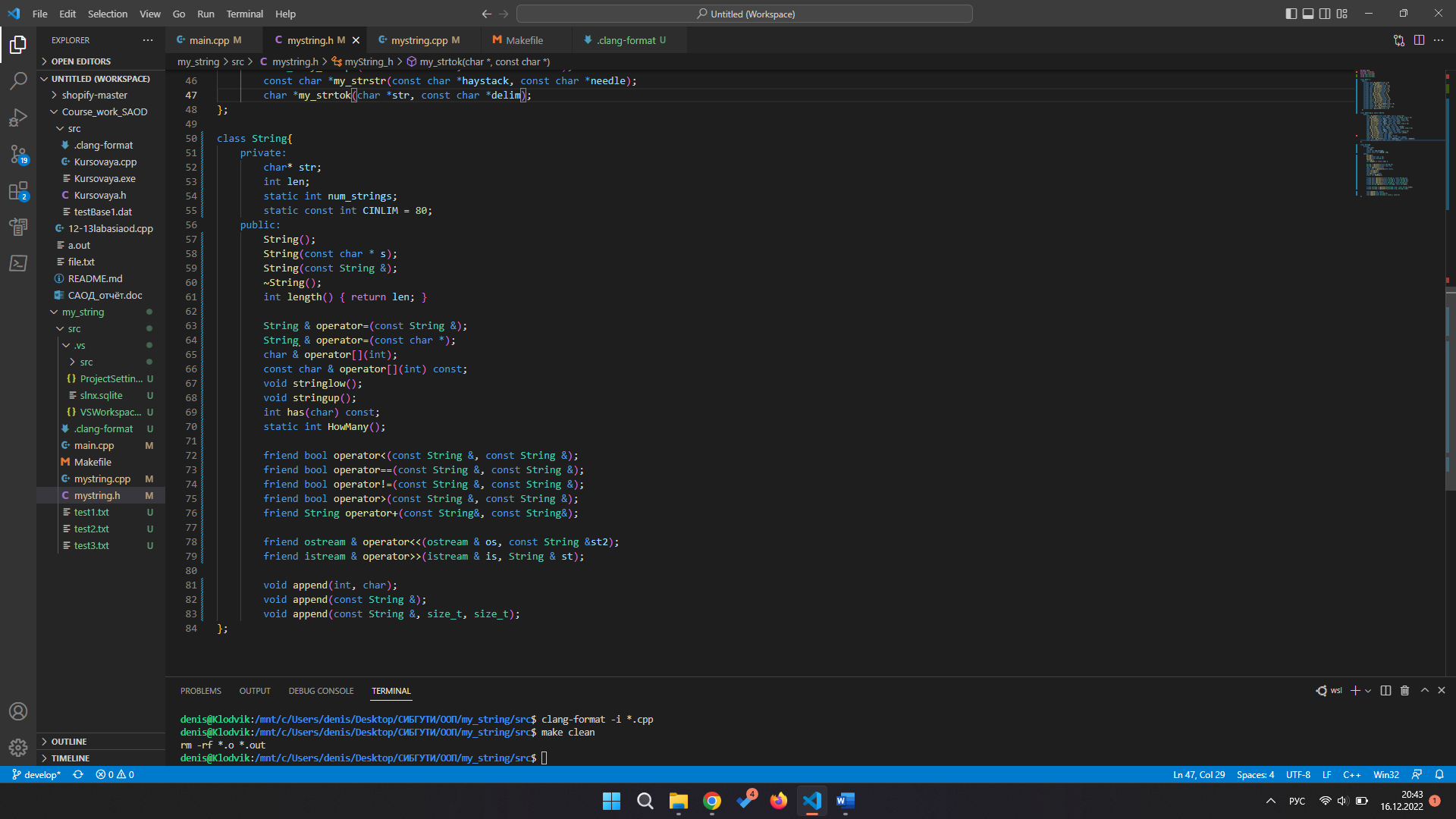


Рисунок 8– Класс

Виртуальный метод (виртуальная функция) — в объектно-ориентированном программировании метод (функция) класса, который может быть переопределён в классах-наследниках так, что конкретная реализация метода для вызова будет определяться во время исполнения (рис. 9).

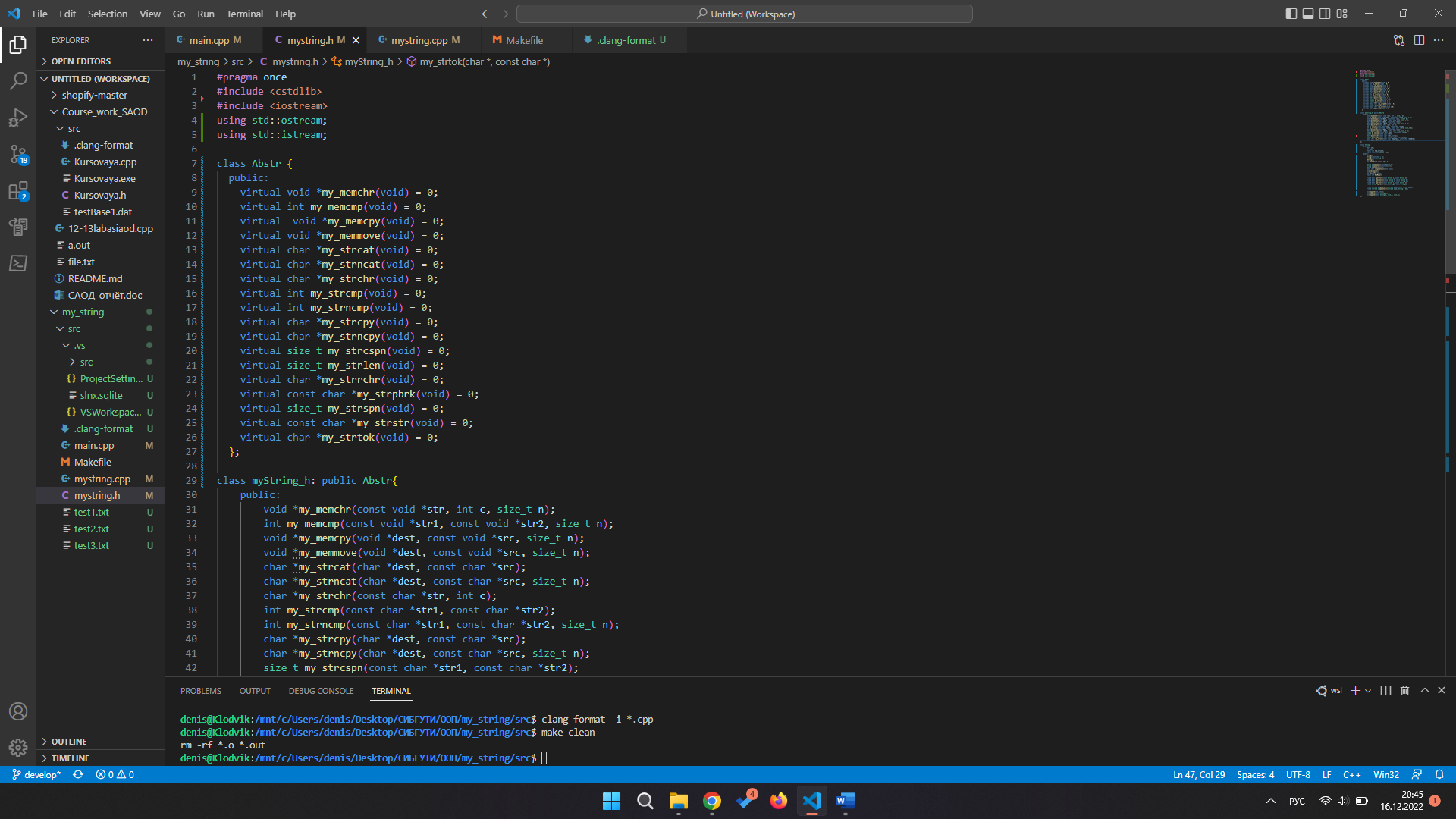


Рисунок 9 – Чистые виртуальные функции

**Программная реализация**

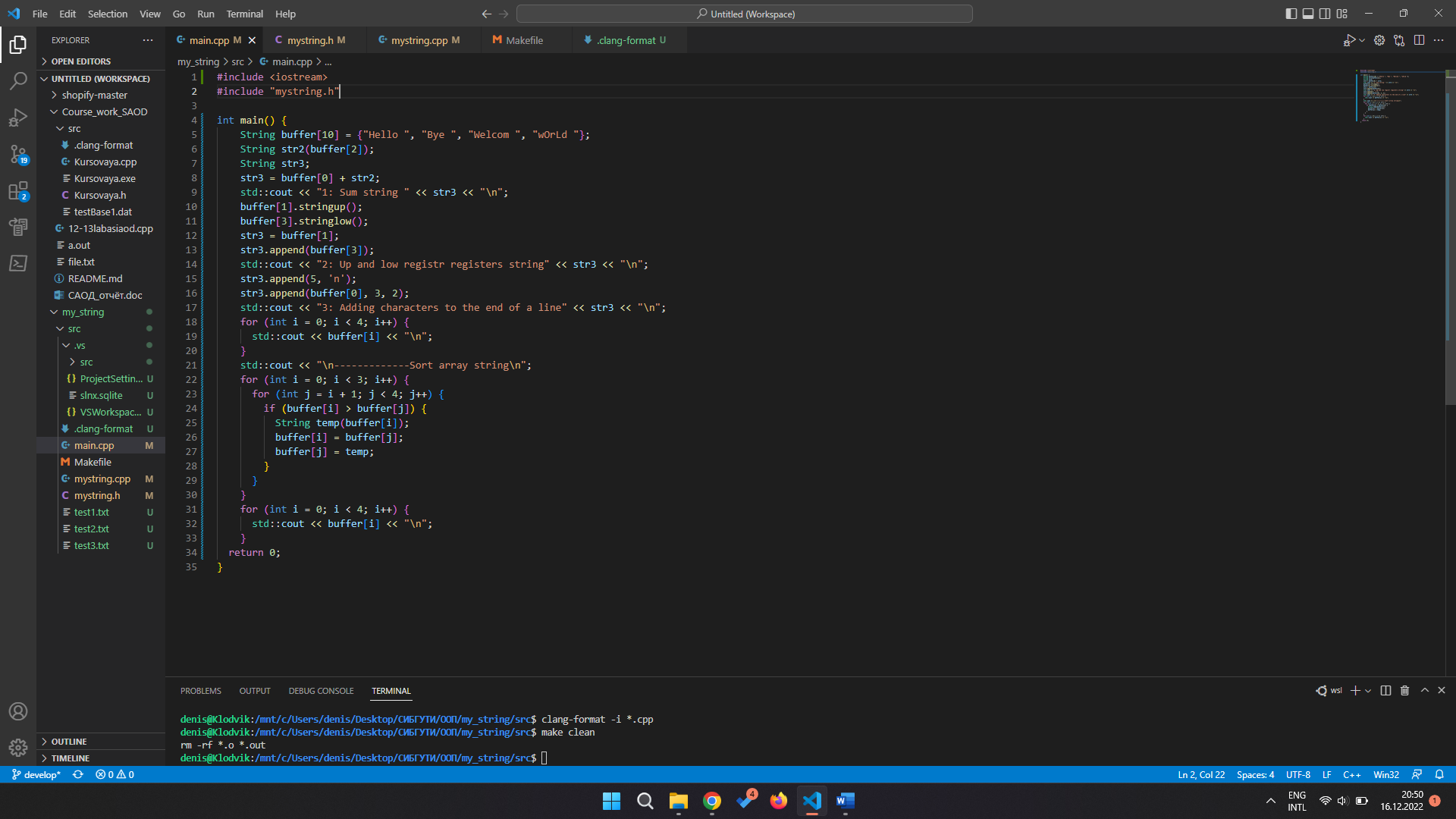


Рисунок 10 – Функция main

int main () – основная функция любой программы. В данной функции для тестирования класса String его методов и операторов.

Также в данном классе происходит инициализация и вызов конструктора String (рис. 11).

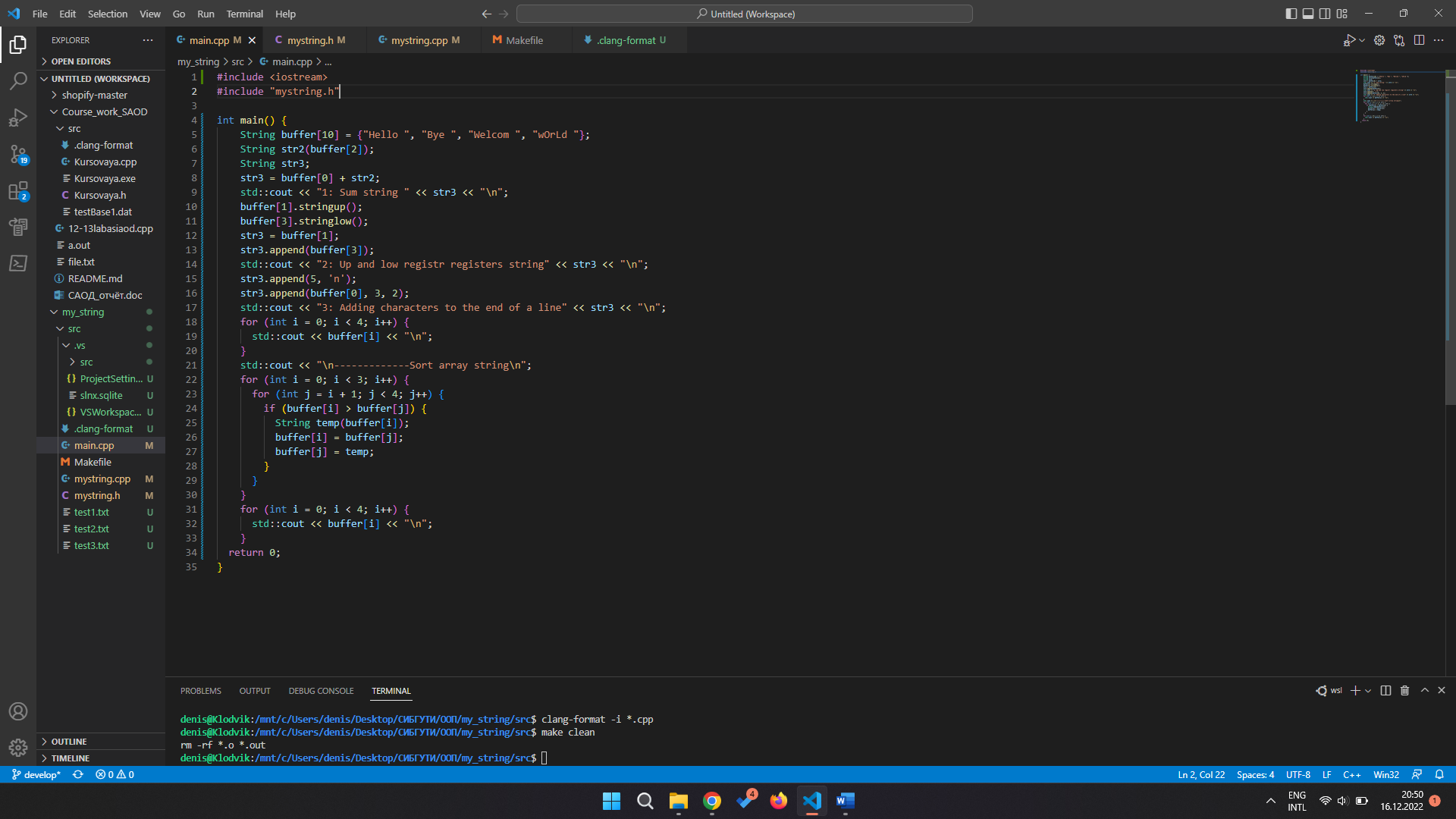
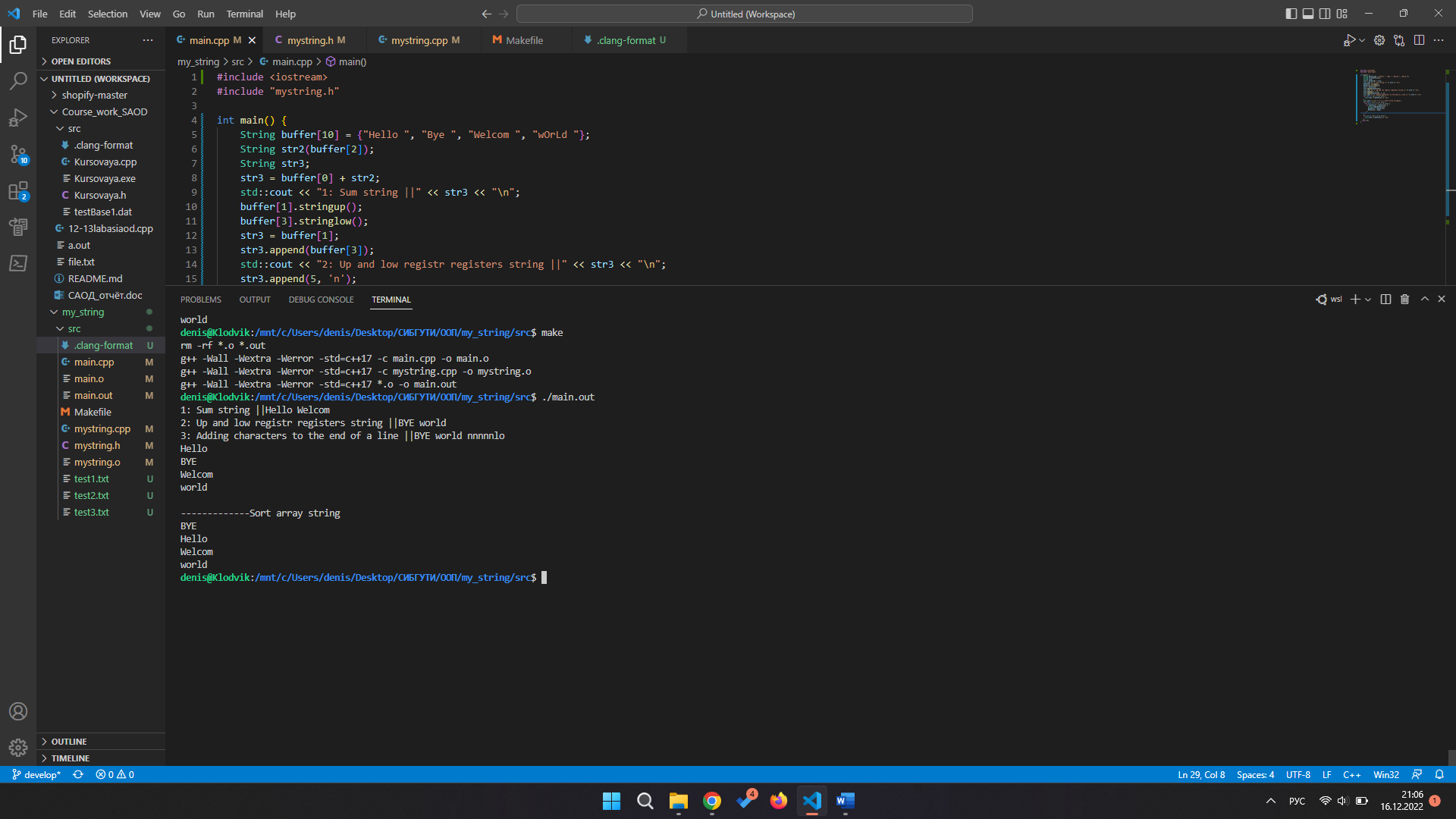


Рисунок 11 – Вызов конструкторов

**Результаты работы**

Результатом моей программы стало тестирование класса String.



В тесте показаны основные методы и операции которые я реализовал в этом классе. Например сложение строк, добавление в конец строки, сравнение строк и т.д.

**Заключение**

Результатом моей работы стало описание объекта строка и полностью рабочие методы библиотеки <cstring> для работы со строками. При разработке были задействованы технологии ООП.

# Используемые источники

1. Стенли Б.Л., Жози Л., Барбара Э.М. Язык программирования C++. - 5-e изд. - Нью-Йорк: Издательский дом "Вильямс", 2014. - 1251 с.
2. C++ Programming Language // Geeksforgeeks URL: https://[www.geeksforgeeks.org/c-plus-plus/](http://www.geeksforgeeks.org/c-plus-plus/) (дата обращения: 03.12.2022).
3. Что такое объектно-ориентированное программирование // timeweb URL: https://timeweb.com/ru/community/articles/obektno-orientirovannoe- programmirovanie (дата обращения: 04.12.2022).

**Приложение. Текст программы**

1.main.cpp

|  |  |
| --- | --- |
| 1 | #include <iostream> |
| 2 | #include "mystring.h" |
| 3 |  |
| 4 | int main() { |
| 5 | String buffer[10] = {"Hello ", "Bye ", "Welcom ", "wOrLd "}; |
| 6 | String str2(buffer[2]); |
| 7 | String str3; |
| 8 | str3 = buffer[0] + str2; |
| 9 | std::cout << "1: Sum string ||" << str3 << "\n"; |
| 10 | buffer[1].stringup(); |
| 11 | buffer[3].stringlow(); |
| 12 | str3 = buffer[1]; |
| 13 | str3.append(buffer[3]); |
| 14 | std::cout << "2: Up and low registr registers string ||" << str3 << "\n"; |
| 15 | str3.append(5, 'n'); |
| 16 | str3.append(buffer[0], 3, 2); |
| 17 | std::cout << "3: Adding characters to the end of a line ||" << str3 << "\n"; |
| 18 | for (int i = 0; i < 4; i++) { |
| 19 | std::cout << buffer[i] << "\n"; |
| 20 | } |
| 21 | std::cout << "\n-------------Sort array string\n"; |
| 22 | for (int i = 0; i < 3; i++) { |
| 23 | for (int j = i + 1; j < 4; j++) { |
| 24 | if (buffer[i] > buffer[j]) { |
| 25 | String temp(buffer[i]); |
| 26 | buffer[i] = buffer[j]; |
| 27 | buffer[j] = temp; |
| 28 | } |
| 29 | } |
| 30 | } |
| 31 | for (int i = 0; i < 4; i++) { |
| 32 | std::cout << buffer[i] << "\n"; |
| 33 | } |
| 34 | return 0; |
| 35 | } |

2.mystring.h

|  |  |
| --- | --- |
| 1 | #pragma once |
| 2 | #include <cstdlib> |
| 3 | #include <iostream> |
| 4 | using std::ostream; |
| 5 | using std::istream; |
| 6 |  |
| 7 | class Abstr { |
| 8 | public: |
| 9 | virtual void \*my\_memchr(void) = 0; |
| 10 | virtual int my\_memcmp(void) = 0; |
| 11 | virtual  void \*my\_memcpy(void) = 0; |
| 12 | virtual void \*my\_memmove(void) = 0; |
| 13 | virtual char \*my\_strcat(void) = 0; |
| 14 | virtual char \*my\_strncat(void) = 0; |
| 15 | virtual char \*my\_strchr(void) = 0; |
| 16 | virtual int my\_strcmp(void) = 0; |
| 17 | virtual int my\_strncmp(void) = 0; |
| 18 | virtual char \*my\_strcpy(void) = 0; |
| 19 | virtual char \*my\_strncpy(void) = 0; |
| 20 | virtual size\_t my\_strcspn(void) = 0; |
| 21 | virtual size\_t my\_strlen(void) = 0; |
| 22 | virtual char \*my\_strrchr(void) = 0; |
| 23 | virtual const char \*my\_strpbrk(void) = 0; |
| 24 | virtual size\_t my\_strspn(void) = 0; |
| 25 | virtual const char \*my\_strstr(void) = 0; |
| 26 | virtual char \*my\_strtok(void) = 0; |
| 27 | }; |
| 28 |  |
| 29 | class myString\_h: public Abstr{ |
| 30 | public: |
| 31 | void \*my\_memchr(const void \*str, int c, size\_t n); |
| 32 | int my\_memcmp(const void \*str1, const void \*str2, size\_t n); |
| 33 | void \*my\_memcpy(void \*dest, const void \*src, size\_t n); |
| 34 | void \*my\_memmove(void \*dest, const void \*src, size\_t n); |
| 35 | char \*my\_strcat(char \*dest, const char \*src); |
| 36 | char \*my\_strncat(char \*dest, const char \*src, size\_t n); |
| 37 | char \*my\_strchr(const char \*str, int c); |
| 38 | int my\_strcmp(const char \*str1, const char \*str2); |
| 39 | int my\_strncmp(const char \*str1, const char \*str2, size\_t n); |
| 40 | char \*my\_strcpy(char \*dest, const char \*src); |
| 41 | char \*my\_strncpy(char \*dest, const char \*src, size\_t n); |
| 42 | size\_t my\_strcspn(const char \*str1, const char \*str2); |
| 43 | size\_t my\_strlen(const char \*str); |
| 44 | char \*my\_strrchr(const char \*str, int c); |
| 45 | size\_t my\_strspn(const char \*str1, const char \*str2); |
| 46 | const char \*my\_strstr(const char \*haystack, const char \*needle); |
| 47 | char \*my\_strtok(char \*str, const char \*delim); |
| 48 | }; |
| 49 |  |
| 50 | class String{ |
| 51 | private: |
| 52 | char\* str; |
| 53 | int len; |
| 54 | static int num\_strings; |
| 55 | static const int CINLIM = 80; |
| 56 | public: |
| 57 | String(); |
| 58 | String(const char \* s); |
| 59 | String(const String &); |
| 60 | ~String(); |
| 61 | int length() { return len; } |
| 62 |  |
| 63 | String & operator=(const String &); |
| 64 | String & operator=(const char \*); |
| 65 | char & operator[](int); |
| 66 | const char & operator[](int) const; |
| 67 | void stringlow(); |
| 68 | void stringup(); |
| 69 | static int HowMany(); |
| 70 |  |
| 71 | friend bool operator<(const String &, const String &); |
| 72 | friend bool operator==(const String &, const String &); |
| 73 | friend bool operator!=(const String &, const String &); |
| 74 | friend bool operator>(const String &, const String &); |
| 75 | friend String operator+(const String&, const String&); |
| 76 |  |
| 77 | friend ostream & operator<<(ostream & os, const String &st2); |
| 78 | friend istream & operator>>(istream & is, String & st); |
| 79 |  |
| 80 | void append(int, char); |
| 81 | void append(const String &); |
| 82 | void append(const String &, size\_t, size\_t); |
| 83 | }; |

3.mystring.cpp

|  |  |
| --- | --- |
| 1 | #include "mystring.h" |
| 2 |  |
| 3 | void \*myString\_h::my\_memchr(const void \*str, int c, size\_t n) { |
| 4 | char \*ptr = (char \*)str; |
| 5 | void \*res = NULL; |
| 6 |  |
| 7 | for (size\_t i = 0; (i < n) && ptr; i++) { |
| 8 | if (\*ptr == (char)c) { |
| 9 | res = ptr; |
| 10 | break; |
| 11 | } |
| 12 |  |
| 13 | ptr++; |
| 14 | } |
| 15 |  |
| 16 | return res; |
| 17 | } |
| 18 |  |
| 19 | int myString\_h::my\_memcmp(const void \*str1, const void \*str2, size\_t n = -1) { |
| 20 | char \*ptr1 = (char \*)str1; |
| 21 | char \*ptr2 = (char \*)str2; |
| 22 | int cmp = 0; |
| 23 |  |
| 24 | if (n) { |
| 25 | while (\*(char \*)ptr1 == \*(char \*)ptr2 && --n) { |
| 26 | ptr1 = (char \*)ptr1 + 1; |
| 27 | ptr2 = (char \*)ptr2 + 1; |
| 28 | } |
| 29 | cmp = (\*((unsigned char \*)ptr1) - \*((unsigned char \*)ptr2)); |
| 30 | } else { |
| 31 | while (\*(char \*)ptr1 == \*(char \*)ptr2) { |
| 32 | ptr1 = (char \*)ptr1 + 1; |
| 33 | ptr2 = (char \*)ptr2 + 1; |
| 34 | } |
| 35 | cmp = (\*((unsigned char \*)ptr1) - \*((unsigned char \*)ptr2)); |
| 36 | } |
| 37 |  |
| 38 | return cmp; |
| 39 | } |
| 40 |  |
| 41 | void \*myString\_h::my\_memcpy(void \*dest, const void \*src, size\_t n) { |
| 42 | char \*cdest = (char \*)dest; |
| 43 | const char \*csrc = (char \*)src; |
| 44 |  |
| 45 | for (size\_t i = 0; i < n; ++i) { |
| 46 | \*cdest++ = \*csrc++; |
| 47 | } |
| 48 |  |
| 49 | return dest; |
| 50 | } |
| 51 |  |
| 52 | void \*myString\_h::my\_memmove(void \*dest, const void \*src, size\_t n) { |
| 53 | void \*ret = dest; |
| 54 |  |
| 55 | if (dest <= src || (char \*)dest >= ((char \*)src + n)) { |
| 56 | while (n--) { |
| 57 | \*(char \*)dest = \*(char \*)src; |
| 58 | dest = (char \*)dest + 1; |
| 59 | src = (char \*)src + 1; |
| 60 | } |
| 61 | } else { |
| 62 | dest = (char \*)dest + n - 1; |
| 63 | src = (char \*)src + n - 1; |
| 64 | while (n--) { |
| 65 | \*(char \*)dest = \*(char \*)src; |
| 66 | dest = (char \*)dest - 1; |
| 67 | src = (char \*)src - 1; |
| 68 | } |
| 69 | } |
| 70 |  |
| 71 | return ret; |
| 72 | } |
| 73 |  |
| 74 | char \*myString\_h::my\_strcat(char \*dest, const char \*src) { |
| 75 | char \*ptr = dest; |
| 76 |  |
| 77 | while (\*dest != '\0') { |
| 78 | dest++; |
| 79 | } |
| 80 |  |
| 81 | while (\*src != '\0') { |
| 82 | \*dest++ = \*src++; |
| 83 | } |
| 84 |  |
| 85 | \*dest = '\0'; |
| 86 |  |
| 87 | return ptr; |
| 88 | } |
| 89 |  |
| 90 | char \*myString\_h::my\_strncat(char \*dest, const char \*src, size\_t n) { |
| 91 | char \*ptr = dest; |
| 92 |  |
| 93 | while (\*dest != '\0') { |
| 94 | dest++; |
| 95 | } |
| 96 |  |
| 97 | while (n > 0) { |
| 98 | \*dest++ = \*src++; |
| 99 | n--; |
| 100 | } |
| 101 |  |
| 102 | \*dest = '\0'; |
| 103 |  |
| 104 | return ptr; |
| 105 | } |
| 106 |  |
| 107 | char \*myString\_h::my\_strchr(const char \*str, int c) { |
| 108 | char \*ptr = (char \*)str; |
| 109 |  |
| 110 | for (; (\*ptr != (unsigned char)c) && (\*ptr != '\0'); ptr++) { |
| 111 | } |
| 112 | if (\*ptr == '\0') { |
| 113 | ptr = NULL; |
| 114 | } |
| 115 |  |
| 116 | return ptr; |
| 117 | } |
| 118 |  |
| 119 | int myString\_h::my\_strcmp(const char \*str1, const char \*str2) { |
| 120 | int res = 0; |
| 121 |  |
| 122 | if (str1 && str2) { |
| 123 | int i = 0; |
| 124 |  |
| 125 | while (str1[i] && str1[i] == str2[i]) { |
| 126 | i++; |
| 127 | } |
| 128 |  |
| 129 | res = str1[i] - str2[i]; |
| 130 | } |
| 131 |  |
| 132 | return res; |
| 133 | } |
| 134 |  |
| 135 | int myString\_h::my\_strncmp(const char \*str1, const char \*str2, size\_t n) { |
| 136 | int res = 0; |
| 137 |  |
| 138 | if (str1 && str2 && n > 0) { |
| 139 | for (size\_t i = 0; i < n; i++) { |
| 140 | if (str1[i] == '\0' || str1[i] != str2[i]) { |
| 141 | res = str1[i] - str2[i]; |
| 142 | break; |
| 143 | } |
| 144 | } |
| 145 | } |
| 146 |  |
| 147 | return res; |
| 148 | } |
| 149 |  |
| 150 | char \*myString\_h::my\_strcpy(char \*dest, const char \*src) { |
| 151 | char \*ptr = dest; |
| 152 |  |
| 153 | while (\*src != '\0') { |
| 154 | \*dest++ = \*src++; |
| 155 | } |
| 156 |  |
| 157 | \*dest = '\0'; |
| 158 |  |
| 159 | return ptr; |
| 160 | } |
| 161 |  |
| 162 | char \*myString\_h::my\_strncpy(char \*dest, const char \*src, size\_t n) { |
| 163 | char \*ptr = dest; |
| 164 |  |
| 165 | while (\*src && n-- && (dest != NULL)) { |
| 166 | \*dest++ = \*src++; |
| 167 | } |
| 168 |  |
| 169 | \*dest = '\0'; |
| 170 |  |
| 171 | return ptr; |
| 172 | } |
| 173 |  |
| 174 | size\_t myString\_h::my\_strcspn(const char \*str1, const char \*str2) { |
| 175 | size\_t result = 0; |
| 176 | bool flag = false; |
| 177 |  |
| 178 | for (; \*str1; result++, str1++) { |
| 179 | for (const char \*dest = str2; \*dest; dest++) { |
| 180 | if (\*str1 == \*dest) { |
| 181 | flag = true; |
| 182 | break; |
| 183 | } |
| 184 | } |
| 185 | if (flag == true) { |
| 186 | break; |
| 187 | } |
| 188 | } |
| 189 |  |
| 190 | return result; |
| 191 | } |
| 192 |  |
| 193 | size\_t myString\_h::my\_strlen(const char \*str) { |
| 194 | size\_t count = 0; |
| 195 | if (str != NULL) { |
| 196 | while (\*str != '\0') { |
| 197 | str++; |
| 198 | count++; |
| 199 | } |
| 200 | } else { |
| 201 | count = -1; |
| 202 | } |
| 203 |  |
| 204 | return count; |
| 205 | } |
| 206 |  |
| 207 | char \*myString\_h::my\_strrchr(const char \*str, int c) { |
| 208 | char \*ptr = (char \*)str; |
| 209 |  |
| 210 | for (; \*ptr != '\0'; ptr++) { |
| 211 | } |
| 212 | size\_t i; |
| 213 | for (i = my\_strlen(str) + 1; (\*ptr != (unsigned char)c) && (i > 0); |
| 214 | i--, ptr--) { |
| 215 | } |
| 216 | if (i == 0) { |
| 217 | ptr = NULL; |
| 218 | } |
| 219 |  |
| 220 | return ptr; |
| 221 | } |
| 222 |  |
| 223 | size\_t myString\_h::my\_strspn(const char \*str1, const char \*str2) { |
| 224 | size\_t i; |
| 225 | bool flag = false; |
| 226 |  |
| 227 | for (i = 0; str1[i] != '\0'; i++) { |
| 228 | for (size\_t j = 0; str2[j] != str1[i]; j++) { |
| 229 | if (str2[j] == '\0') { |
| 230 | flag = true; |
| 231 | break; |
| 232 | } |
| 233 | } |
| 234 | if (flag == true) { |
| 235 | break; |
| 236 | } |
| 237 | } |
| 238 |  |
| 239 | return i; |
| 240 | } |
| 241 |  |
| 242 | const char \*myString\_h::my\_strstr(const char \*haystack, const char \*needle) { |
| 243 | char \*hst = (char \*)haystack; |
| 244 | char \*ndl = (char \*)needle; |
| 245 | char \*res = NULL; |
| 246 |  |
| 247 | if (\*ndl == '\0') { |
| 248 | res = hst; |
| 249 | } |
| 250 |  |
| 251 | for (size\_t i = 0; i < my\_strlen(hst); i++) { |
| 252 | if (hst[i] == \*ndl) { |
| 253 | char \*tmp = &hst[i]; |
| 254 | int tmp\_res = my\_strncmp(tmp, ndl, my\_strlen(ndl)); |
| 255 | if (tmp\_res == 0) { |
| 256 | res = &hst[i]; |
| 257 | break; |
| 258 | } |
| 259 | } |
| 260 | } |
| 261 |  |
| 262 | return res; |
| 263 | } |
| 264 |  |
| 265 | char \*myString\_h::my\_strtok(char \*str, const char \*delim) { |
| 266 | static char \*s = NULL; |
| 267 | char \*ret = NULL; |
| 268 |  |
| 269 | if (str != NULL) { |
| 270 | s = str; |
| 271 | } |
| 272 |  |
| 273 | if (s != NULL && my\_strlen(s)) { |
| 274 | const size\_t dlen = my\_strlen(delim); |
| 275 | while (\*s && my\_memchr(delim, \*s, dlen) != NULL) { |
| 276 | ++s; |
| 277 | } |
| 278 | if (\*s) { |
| 279 | ret = s; |
| 280 | while (\*s) { |
| 281 | if (my\_memchr(delim, \*s, dlen) != NULL) { |
| 282 | break; |
| 283 | } else { |
| 284 | ++s; |
| 285 | } |
| 286 | } |
| 287 | if (\*s) { |
| 288 | s[0] = 0; |
| 289 | ++s; |
| 290 | } |
| 291 | } |
| 292 | } |
| 293 |  |
| 294 | return ret; |
| 295 | } |
| 296 |  |
| 297 | // |
| 298 |  |
| 299 | int String::num\_strings = 0; |
| 300 | myString\_h \*strin; |
| 301 |  |
| 302 | int String::HowMany() { return num\_strings; } |
| 303 |  |
| 304 | String::String() : String("") {} |
| 305 |  |
| 306 | String::String(const char \*s) : len(strin->my\_strlen(s)) { |
| 307 | str = new char[len + 1]; |
| 308 | strin->my\_strcpy(str, s); |
| 309 | num\_strings++; |
| 310 | } |
| 311 |  |
| 312 | String::String(const String &st) : len(st.len) { |
| 313 | num\_strings++; |
| 314 | str = new char[len + 1]; |
| 315 | strin->my\_strcpy(str, st.str); |
| 316 | } |
| 317 |  |
| 318 | String::~String() { |
| 319 | --num\_strings; |
| 320 | delete[] str; |
| 321 | } |
| 322 |  |
| 323 | String &String::operator=(const String &st) { |
| 324 | if (this == &st) return \*this; |
| 325 | delete[] str; |
| 326 | len = st.len; |
| 327 | str = new char[len + 1]; |
| 328 | strin->my\_strcpy(str, st.str); |
| 329 | return \*this; |
| 330 | } |
| 331 |  |
| 332 | String &String::operator=(const char \*s) { |
| 333 | delete[] str; |
| 334 | len = strin->my\_strlen(s); |
| 335 | str = new char[len + 1]; |
| 336 | strin->my\_strcpy(str, s); |
| 337 | return \*this; |
| 338 | } |
| 339 |  |
| 340 | void String::stringlow() { |
| 341 | for (int j = 0; j < len; ++j) str[j] = (char)tolower(str[j]); |
| 342 | } |
| 343 |  |
| 344 | void String::stringup() { |
| 345 | for (int j = 0; j < len; ++j) str[j] = (char)toupper(str[j]); |
| 346 | } |
| 347 |  |
| 348 | char &String::operator[](int i) { return str[i]; } |
| 349 |  |
| 350 | const char &String::operator[](int i) const { return str[i]; } |
| 351 |  |
| 352 | bool operator<(const String &st1, const String &st2) { |
| 353 | return (strin->my\_memcmp(st1.str, st2.str) < 0); |
| 354 | } |
| 355 |  |
| 356 | bool operator==(const String &st1, const String &st2) { |
| 357 | return (strin->my\_memcmp(st1.str, st2.str) == 0); |
| 358 | } |
| 359 |  |
| 360 | bool operator!=(const String &st1, const String &st2) { |
| 361 | return (strin->my\_memcmp(st1.str, st2.str) != 0); |
| 362 | } |
| 363 |  |
| 364 | bool operator>(const String &st1, const String &st2) { return st2 < st1; } |
| 365 |  |
| 366 | ostream &operator<<(ostream &os, const String &st) { |
| 367 | os << st.str; |
| 368 | return os; |
| 369 | } |
| 370 |  |
| 371 | String operator+(const String &st1, const String &st2) { |
| 372 | String \_temp; |
| 373 | delete[] \_temp.str; |
| 374 | \_temp.len = st1.len + st2.len; |
| 375 | \_temp.str = new char[\_temp.len + 1]; |
| 376 |  |
| 377 | strin->my\_strcpy(\_temp.str, st1.str); |
| 378 | strin->my\_strcpy(\_temp.str + st1.len, st2.str); |
| 379 |  |
| 380 | return \_temp; |
| 381 | } |
| 382 |  |
| 383 | istream &operator>>(istream &is, String &st) { |
| 384 | char temp[String::CINLIM]; |
| 385 | is.get(temp, String::CINLIM); |
| 386 | if (is) st = temp; |
| 387 | while (is && is.get() != '\n') continue; |
| 388 | return is; |
| 389 | } |
| 390 |  |
| 391 | void String::append(int n, char c) { |
| 392 | String temp(str); |
| 393 | delete[] str; |
| 394 | len += n; |
| 395 | str = new char[len + 1]; |
| 396 | strin->my\_strcpy(str, temp.str); |
| 397 | char ch[] = "T"; |
| 398 | ch[0] = c; |
| 399 | while (n != 0) { |
| 400 | strin->my\_strcat(str, ch); |
| 401 | n--; |
| 402 | } |
| 403 | } |
| 404 |  |
| 405 | void String::append(const String &str1) { |
| 406 | String temp(str); |
| 407 | delete[] str; |
| 408 | len += str1.len; |
| 409 | str = new char[len + 1]; |
| 410 | strin->my\_strcpy(str, temp.str); |
| 411 | strin->my\_strcat(str, str1.str); |
| 412 | } |
| 413 |  |
| 414 | void String::append(const String &str1, size\_t pos, size\_t cout) { |
| 415 | if ((str1.len - (pos + cout)) > 0) { |
| 416 | String temp(str); |
| 417 | delete[] str; |
| 418 | len += str1.len - (pos + cout); |
| 419 | str = new char[len + 1]; |
| 420 | strin->my\_strcpy(str, temp.str); |
| 421 | char \*temp1 = str1.str; |
| 422 | while (pos != 0) { |
| 423 | temp1++; |
| 424 | pos--; |
| 425 | } |
| 426 | strin->my\_strncat(str, temp1, cout); |
| 427 | } |
| 428 | } |