**Краткое описание**: Необходимо создать класс для работы со строками, не использующий существующих библиотек и основанный на хранении строки в виде указателя на char.

**Функциональность:**

Класс должен иметь следующие виды конструкторов:

- Без параметров

- Копирования

- Принимающий на вход строку типа char\*

- Принимающий на вход значение типа int

- Принимающий на вход значение типа double

**Класс должен перегружать следующие операции:**

+=, =

Параметром может быть: char\*, T, int, double.

Возвращаемое значение - ссылка на результат.

[ ]

Параметром должен быть индекс символа типа int, возвращаемое значение символ с заданным индексом типа char. Нумерация символов с 0.

**Класс должен реализовывать следующие методы:**

Длина строки в символах

Поиск первого вхождения подстроки what

В паре с предыдущим методом метод нахождения следующего вхождения

Удаление подстроки. Две реализации. Первая удаляет первое вхождение

указанной подстроки

Вторая удаляет подстроку указанной длины по указанной позиции

Удаление всех вхождений подстроки

Вставка подстроки на указанную позицию

Замена первого вхождения подстроки what на replace\_with

Замена всех вхождений подстроки what на replace\_with

Отсечение пробельных символов (перенос строки, пробел, табуляция) с обеих сторон

Приведение всех буквенных символов (и латинских и кириллицы) к нижнему регистру

Приведение всех буквенных символов (и латинских и кириллицы) к верхнему регистру

Преобразование в int

Преобразование в double

Инвертирование строки

**Класс должен иметь дружественные функции:**

Длина строки в символах

Для конкатенации двух строк

Обмена содержимого двух строк

Отсечение пробельных символов (перенос строки, пробел, табуляция) с обеих

сторон

Сравнение строк

**Описание класса**

Класс MedString хранит данные в массиве элементов типа char. Его единственным параметром является указатель на char *String*.

Класс имеет 5 различных конструкторов.

MedString(void);

Конструктор по умолчанию. Присваивает указателю *String* ссылку на NULL.

MedString(MedString &From);

Конструктор копирования. Принимает на вход другой экземпляр того же класса и копирует строку из этого экземпляра в создаваемый объект.

Реализация: выделяем необходимую память, посимвольно переносим символы из входной строки в строку нового экземпляра класса, ставим символ окончания строки ‘\0’ в конец.

MedString :: MedString(char \*From)

Конструктор, принимающий на вход строку, содержащуюся в массиве типа char. Принцип работы идентичен с конструктором копирования.

MedString :: MedString(int From)

Конструктор, принимающий на вход целое число и записывающий его в объект как строку.

MedString :: MedString(double from)

Идентичен предыдущему конструктору, за исключением того, что число дробное.

MedString::~MedString()

Деструктор класса. Удаляет строку, которую содержит объект.

MedString & MedString :: operator = (char \*add)

Присваивание строке в объекте значение введенной строки.

MedString & MedString :: operator = (int add)

Присваивание строке в объекте значение введенного целого числа, преобразованного в строку.

MedString & MedString :: operator = (MedString add)

Работает как конструктор копирования, но может быть вызван не только при объявлении объекта.

MedString & MedString :: operator = (double add)

Присваивание строке в объекте значение введенного дробного числа, преобразованного в строку.

MedString & MedString :: operator += (char \*add)

MedString & MedString :: operator += (int add)

MedString & MedString :: operator += (MedString add)

MedString & MedString :: operator += (double add)

Присоединение к строке в объекте введенной строки (отдельной или из другого объекта), целого/дробного числа (преобразованного в строку).

char& MedString :: operator[](int count) throw(...)

Возвращает символ из строки объекта с индексом *count*. Генерирует исключение типа char\* (“Out of range”) при выходе за границы строки.

void Copy(MedString& Dest, MedString& Source)

Копирует строку из объекта *Source* в объект *Dest*.

void MedString :: Tcopy(char \*Source)

Аналог strcpy для класса. Копирует из поданной на вход строки символы в строку из объекта, из которого был вызван данный метод. Использует раннее реализованную функцию Copy.

int Tlen(MedString& Copy)

Определяет длину строки объекта *Copy*

int CharLen(char \*s)

Аналог функции strlen – определяет длину строки *s*.

void CharCpy(char \*&Dest, char \*Source)

Аналог функции strcpy – копирует содержимое строки *Source* в строку *Dest*.

char\* validDouble(char \*s)

Удаляет нули в конце строки *s.* Функция нужна для реализации методов, использующих параметры типа double

int MedString :: len()

Метод, аналогичный дружественной функции *Tlen.* Использует эту функцию.

MedString operator+(MedString& first, MedString& second)

Конкатенация строк из объектов *first* и *second* и возвращение экземпляра класса с этой строкой. Использует оператор +=.

void MedSwap(MedString& first, MedString &second)

Меняет местами строки во входных объектах.

MedString Ttrim(MedString& from)

Дружественная функция, удаляющая из строки объекта *from* пробельные символы в начале и конце.

bool operator>(MedString& first, MedString& second)

bool operator<(MedString& first, MedString& second)

bool operator==(MedString& first, MedString& second)

Перегрузка операторов сравнения для класса *MedString*.

int MedString :: pos(MedString& what)

Определяет позицию подстроки *what* в строке объекта, из которого вызван данный метод. Возвращает позицию первого символа подстроки, если она находится в строке и -1 если подстрока не входит в строку.

void MedString :: del(MedString& what)

Удаляет подстроку *what* из строки объекта при наличии первой во второй.

int MedString :: nextpos(MedString& what)

Определяет второе вхождение подстроки в строку. Возвращает позицию второго вхождения подстроки в строку или -1 при отсутствии таковой.

void MedString :: del(int begin, int length) throw(...)

Удаляет *length* символов в строке объекта, начиная с символа *begin.* Генерирует исключение при попытке удалить символы «за» строкой.

void MedString :: delall(MedString &what)

Удаляет все подстроки *what* из строки объекта класса.

void MedString :: ins(MedString& what, int whereto) throw(...)

Вставляет в строке объекта подстроку *what* с символа *whereto.* Генерирует исключение при попытке вставить подстроку с несуществующего символа.

void MedString :: replace(MedString& what, MedString& replace\_with)

Заменяет первое вхождение подстроки *what* в строку объекта подстрокой *replace\_with*.

void MedString :: replaceall(MedString& what, MedString& replace\_with)

Действует как предыдущий метод, но заменяет все вхождения подстроки. Использует тот метод как основной алгоритм.

void MedString :: trim()

Аналогично методу *Ttrim* удаляет пробельные символы из начала и конца строки. Использует этот метод как основной.

void MedString :: lower()

void MedString :: upper()

Заменяет все буквы нижнего/верхнего регистра на верхний/нижний регистр.

int MedString :: to\_int() throw(...)

double MedString :: to\_double() throw(...)

Приводит строку в объекте к целому/дробному числу и возвращает его.

void MedString :: reverse()

Обращает строку.

**Тестовый пример**

#include "headers.h"

int main()

{

setlocale(LC\_ALL, "rus");

// Объявление объектов через различные конструкторы и их вывод

MedString a, b("string"), c(15), d(1.2), e(b);

cout << a << endl << b << endl << c << endl << d << endl << e << endl;

// Присоединение и приравнивание

a = 16;

b = 12.4;

c = "Stroka";

d = b;

cout << a << endl << b << endl << c << endl << d << endl;

a += 9;

a += 1.3;

a += "qwe";

a += a;

cout << a << endl;

// []

try

{

for (int i = 0; i < Tlen(c) + 1; i++)

cout << i << ": " << c[i] << endl;

}

catch (char \*s)

{

cout << s << endl;

}

// Tcopy, len

a.Tcopy("to copy");

cout << a << " (" << a.len() << ")" << endl;

// +, swap

b = "1234";

c = "5678";

a = b + c;

MedSwap(b, c);

cout << a << endl << b << " " << c << endl;

// Ttrim

a = " \t qwer qer cad \n";

a = Ttrim(a);

cout << a << endl;

// <, >, ==

a = "qwerty";

b = "qwer";

c = "qwer";

cout << (b == c) << " " << (a < b) << " " << (a > b) << endl;

// pos, del, nextpos, del[2], delall, ins

a = "qwer";

b = "e";

cout << a.pos(b) << endl;

a.del(b);

cout << a << endl;

a = "string string";

b = "ri";

cout << a.nextpos(b) << endl;

a.del(2, 4);

cout << a << endl;

try

{

a.del(4, 144);

}

catch (char \*s)

{

cout << s << endl;

}

a = "string string";

b = "ri";

a.delall(b);

cout << a << endl;

a.ins(b, 0);

cout << a << endl;

// replace, replaceall

a = "we we wewe r t we";

b = "we";

c = 15;

a.replace(b, c);

cout << a << endl;

a.replaceall(b, c);

cout << a << endl;

// lower, upper

a = "ASDASFASDFASDF";

a.lower();

cout << a << endl;

a.upper();

cout << a << endl;

// reverse

d = 15623.12;

d.reverse();

cout << d << endl;

system("pause");

return 0;

}

**Исходный код**

MedString.h

#pragma once

#include <iostream>

using namespace std;

class MedString

{

private:

char \*String;

public:

// конструкторы

MedString(void);

MedString(MedString &From);

MedString(char \*From);

MedString(int From);

MedString(double From);

// деструктор

~MedString();

// перегрузки операций +=, = для char\*, int, double, MedString

MedString &operator += (char \*add);

MedString &operator += (int add);

MedString &operator += (double add);

MedString &operator += (MedString add);

MedString &operator = (char \*add);

MedString &operator = (int add);

MedString &operator = (double add);

MedString &operator = (MedString add);

//перегрузка оператора []

char& operator[](int count) throw(...);

// методы

void Tcopy(char \*Source);

int len();

int pos(MedString& what);

void del(MedString& what);

int nextpos(MedString& what);

void del(int begin, int length) throw(...);

void delall(MedString& what);

void ins(MedString& what, int whereto) throw(...);

void replace(MedString& what, MedString& replace\_with);

void replaceall(MedString& what, MedString& replace\_with);

void trim();

void lower();

void upper();

int to\_int() throw(...);

double to\_double() throw(...);

void reverse();

// дружественные функции

friend void Copy(MedString& Dest, MedString& Source);

friend int Tlen(MedString& Copy);

friend MedString operator+(MedString& first, MedString& second);

friend void MedSwap(MedString& first, MedString &second);

friend MedString Ttrim(MedString& from);

friend bool operator>(MedString& first, MedString& second);

friend bool operator<(MedString& first, MedString& second);

friend bool operator==(MedString& first, MedString& second);

friend ostream& operator<<(ostream& out, MedString& p);

};

int CharLen(char \*s);

void CharCpy(char \*&Dest, char \*Source);

char\* validDouble(char \*s);

MedString.cpp

#include "headers.h"

int CharLen(char \*s)

{

if (!s)

return 0;

int i = 0;

while (s[i++] != '\0');

return --i;

}

void CharCpy(char \*&Dest, char \*Source)

{

int i;

for (i = 0; i < CharLen(Source); i++)

Dest[i] = Source[i];

Dest[i] = '\0';

}

char\* validDouble(char \*s)

{

int i = CharLen(s) - 1;

while (s[i] == '0')

i--;

s[i + 1] = '\0';

return s;

}

int Tlen(MedString& Copy)

{

int i = 0;

while (Copy.String[i++] != '\0');

return --i;

}

void Copy(MedString& Dest, MedString& Source)

{

if (!Dest.String)

delete [] Dest.String;

Dest.String = new char[Tlen(Source) + 1];

int i;

for (i = 0; i < Tlen(Source); i++)

Dest.String[i] = Source.String[i];

Dest.String[i] = '\0';

}

MedString :: MedString()

{

String = NULL;

}

MedString :: MedString(MedString &From)

{

String = new char[Tlen(From) + 1];

int i;

for (i = 0; i < Tlen(From); i++)

String[i] = From.String[i];

String[i] = '\0';

}

MedString :: MedString(char \*From)

{

String = new char[CharLen(From) + 1];

int i;

for (i = 0; i < CharLen(From); i++)

String[i] = From[i];

String[i] = '\0';

}

MedString :: MedString(int From)

{

char \*buf = new char[1024];

itoa(From, buf, 10);

String = new char[CharLen(buf) + 1];

CharCpy(String, buf);

}

MedString :: MedString(double from)

{

char \*buf = new char[1024];

sprintf(buf, "%f", from);

buf = validDouble(buf);

String = new char[CharLen(buf) + 1];

CharCpy(String, buf);

}

MedString::~MedString()

{

if (String)

delete [] String;

}

MedString & MedString :: operator = (char \*add)

{

if (String)

delete String;

String = new char[CharLen(add) + 1];

CharCpy(String, add);

return \*this;

}

MedString & MedString :: operator = (int add)

{

if (String)

delete String;

char \*buf = new char[1024];

itoa(add, buf, 10);

String = new char[CharLen(buf) + 1];

CharCpy(String, buf);

return \*this;

}

MedString & MedString :: operator = (MedString add)

{

\*this = add.String;

String[Tlen(add)] = '\0';

return \*this;

}

MedString & MedString :: operator = (double add)

{

if (String)

delete String;

char \*buf = new char[1024];

sprintf(buf, "%f", add);

buf = validDouble(buf);

String = new char[CharLen(buf) + 1];

CharCpy(String, buf);

return \*this;

}

MedString & MedString :: operator += (char \*add)

{

char \*buf = new char[CharLen(String) + 1 + CharLen(add)];

CharCpy(buf, String);

int i;

for (i = 0; i < CharLen(add); i++)

buf[CharLen(String) + i] = add[i];

buf[CharLen(String) + i] = '\0';

if (String)

delete [] String;

String = buf;

return \*this;

}

MedString & MedString :: operator += (int add)

{

char \*buf = new char[1024];

return \*this += itoa(add, buf, 10);

}

MedString & MedString :: operator += (MedString add)

{

return \*this += add.String;

}

MedString & MedString :: operator += (double add)

{

char \*buf = new char[1024];

sprintf(buf, "%f", add);

buf = validDouble(buf);

return \*this += buf;

}

char& MedString :: operator[](int count) throw(...)

{

if (count < 0 || count > Tlen(\*this) - 1)

throw "Out of range";

return String[count];

}

void MedString :: Tcopy(char \*Source)

{

Copy(\*this, MedString(Source));

}

int MedString :: len()

{

return Tlen(\*this);

}

MedString operator+(MedString& first, MedString& second)

{

return MedString(first) += second;

}

void MedSwap(MedString& first, MedString &second)

{

MedString t(first);

first = second;

second = t;

}

MedString Ttrim(MedString& from)

{

int i = Tlen(from) - 1;

char \*buf = new char[Tlen(from) + 1];

CharCpy(buf, from.String);

while (buf[i] == ' ' || buf[i] == '\t' || buf[i] == '\n')

i--;

int j = 0;

while (buf[j] == ' ' || buf[j] == '\t' || buf[j] == '\n')

j++;

char \*buf2 = new char[Tlen(from) + 1 - j - (Tlen(from) + 1 - i)];

int q = 0;

for (int k = j; k < i + 1; k++)

buf2[q++] = buf[k];

buf2[q] = '\0';

delete [] buf;

return buf2;

}

bool operator>(MedString& first, MedString& second)

{

int min;

if (Tlen(second) > Tlen(first))

min = Tlen(first);

else

min = Tlen(second);

for (int i = 0; i < min; i++)

{

if (first[i] == second[i])

continue;

if (first[i] > second[i])

return true;

else

return false;

}

return false;

}

bool operator<(MedString& first, MedString& second)

{

int min;

if (Tlen(second) > Tlen(first))

min = Tlen(first);

else

min = Tlen(second);

for (int i = 0; i < min; i++)

{

if (first[i] == second[i])

continue;

if (first[i] < second[i])

return true;

else

return false;

}

return false;

}

bool operator==(MedString& first, MedString& second)

{

if (Tlen(first) != Tlen(second))

return false;

for (int i = 0; i < Tlen(first); i++)

if (first[i] != second[i])

return false;

return true;

}

int MedString :: pos(MedString& what)

{

char \*buf = new char[Tlen(what) + 1];

buf[Tlen(what)] = '\0';

for (int i = 0; i < Tlen(\*this) - Tlen(what) + 1; i++)

{

if (String[i] != what[0])

continue;

else

{

for (int j = 0; j < Tlen(what); j++)

buf[j] = String[i + j];

if (MedString(buf) == what)

return i;

}

}

return -1;

}

void MedString :: del(MedString& what)

{

int d = pos(what);

char \*buf = new char[Tlen(\*this) + 1 - Tlen(what)];

int j;

for (j = 0; j < d; j++)

buf[j] = String[j];

for (int i = d + Tlen(what); i < CharLen(String); i++)

buf[j++] = String[i];

buf[j] = '\0';

delete [] String;

String = buf;

}

int MedString :: nextpos(MedString& what)

{

MedString t(\*this);

if (t.pos(what) == -1)

return -1;

t.del(what);

return t.pos(what) + Tlen(what);

}

void MedString :: del(int begin, int length) throw(...)

{

if (begin + length > len())

throw "Out of range";

char \*buf = new char[len() + 1 - length];

int j = 0;

for (int i = 0; i < begin; i++)

buf[j++] = String[i];

for (int i = begin + length; i < len(); i++)

buf[j++] = String[i];

buf[j] = '\0';

delete [] String;

String = buf;

}

void MedString :: delall(MedString &what)

{

while (pos(what) != -1)

del(what);

}

void MedString :: ins(MedString& what, int whereto) throw(...)

{

if (whereto > len())

throw "Out of range";

char \*buf = new char[len() + 1 + Tlen(what)];

int i;

for (i = 0; i < whereto; i++)

buf[i] = String[i];

buf[i] = '\0';

MedString t(buf);

t = t + what + MedString(&String[whereto]);

delete [] String;

String = NULL;

\*this = t;

}

void MedString :: replace(MedString& what, MedString& replace\_with)

{

if (pos(what) == -1)

return;

int a = pos(what);

del(what);

ins(replace\_with, a);

}

void MedString :: replaceall(MedString& what, MedString& replace\_with)

{

while (pos(what) != -1)

replace(what, replace\_with);

}

void MedString :: trim()

{

\*this = Ttrim(\*this);

}

void MedString :: lower()

{

for (int i = 0; i < len(); i++)

{

if (String[i] >= 'A' && String[i] <= 'Z')

String[i] += 32;

/\*if (String[i] <= -128 && String[i] >= -143)

String[i] += 64;\*/

}

}

void MedString :: upper()

{

for (int i = 0; i < len(); i++)

{

if (String[i] >= 'a' && String[i] <= 'z')

String[i] -= 32;

}

}

int MedString :: to\_int() throw(...)

{

return atoi(String);

}

double MedString :: to\_double() throw(...)

{

return atof(String);

}

void MedString :: reverse()

{

char ch;

for (int i = 0; i < len() / 2; i++)

{

ch = String[i];

String[i] = String[len() - i - 1];

String[len() - i - 1] = ch;

}

}

ostream& operator<<(ostream& out, MedString& p)

{

if (p.String)

out << p.String;

else

out << "";

return out;

}

**Список литературы**

Подбельский. Язык С++

Полный справочник по C++. Герберт Шилдт. 2006

Прата С. - Язык программирования С++. Лекции и упражнения (5-е изд.) - 2007