ФГБОУ ВПО

Московский государственный технический университет имени Н.Э. Баумана

Факультет ИУ «Информатика и системы управления»

Кафедра ИУ-3 «Информационные системы и телекоммуникации»

Отчет

по лабораторной работе №1

По предмету “Алгоритмы и структуры данных”

Вариант 12

Выполнил: Сёмин К.А.

Группа ИУ3-31

Проверил: Сакулин С.А.

Москва 2016

Содержание

[Аналитическая часть 2](#_Toc466239602)

[Исследовательская часть 4](#_Toc466239603)

[Экспериментальная часть: 5](#_Toc466239604)

# Аналитическая часть

Проведем сравнительный анализ выведенного нами класса String и c-стиль хранения и использования строк.

Выделим плюсы и минусы использования String:

|  |  |
| --- | --- |
| Плюсы | Минусы |
| * Способен генерировать исключения * Имеет перегруженные математические операторы * Все функции безопасны и не допускают выхода за пределы (кроме operator[]) * Для каждой функции имеет strong-guarantee при возникновении исключительной ситуации * Не допускает утечек памяти ни при каких обстоятельствах | * Ниже скорость работы некоторых функций и операторов, из-за необходимости обеспечения copy-and-swap идиомы * Некоторые функции имеют худшую асимптоматику по памяти (O(n) вместо O(1)) |

## Экспериментальная часть:

Листинг main.cpp:

|  |
| --- |
| #include "String.hpp"  #include <iostream>  #include <cstring>  #include <stdexcept>  using std::cout;  using std::cin;  using std::endl;  int main(int argc, char \*\*argv)  {  drmatters::String foo;  foo += ("hey there");    drmatters::String bar(foo);  drmatters::String foobar = bar;  foobar += foo;  foo = foobar;  cout << foo << endl;  cout << bar << endl;  cout << foobar << endl;  try {  cout << foo[1] << " " << bar.at(1) << endl;  } catch (std::out\_of\_range& oor) {  cout << "Error: :" << oor.what() << endl;  }  foo.clear();  cin >> foo;  cout << foo << endl;    drmatters::String processed;  for(size\_t i = 0; i < foo.size(); i++) {  if(foo.at(i) != '\*') {  processed.pushBack(foo.at(i));  processed.pushBack(foo.at(i));  } else {  continue;  }  }    foo = processed;  cout << foo << endl;    system("pause");  return 0;  } |

Листинг String.hpp:

|  |
| --- |
| #ifndef STRING\_H  #define STRING\_H  #include <iostream>  namespace drmatters {    class String {  public:    String(); //default  String(const String& toCpy); //copy  String(const char\* cStr); //c-str    ~String(); //destr    String& operator=(String str);  String& operator=(const char\* cStr);  char& operator[](size\_t pos);  const char& operator[](size\_t pos) const;  char& at(size\_t pos);  const char& at(size\_t pos) const;    String& operator+(const String& str);  String& operator+=(const String& str);  String& insert(const String& str, const size\_t pos);    bool operator==(const String& toCheck);  bool operator!=(const String& toCheck);  const char\* data() const;  const char\* c\_str() const;    friend std::ostream& operator<<(std::ostream& stream, String& obj);  friend std::istream& operator>>(std::istream& stream, String& obj);    size\_t size() const;  void clear(); //delete all string  bool empty() const;  friend void swap(String& lh, String& rh) {  using std::swap;    swap(lh.curSize, rh.curSize);  swap(lh.capacity, rh.capacity);  swap(lh.pStr, rh.pStr);  }  void pushBack(char c);    private:  size\_t curSize;  size\_t capacity;  char\* pStr;    void increaseCapacity(size\_t n);  String(size\_t length) :  curSize(length),  capacity(length),  pStr(new char[length]) {  }  };  }  #endif //#ifndef STRING\_H |

Листинг String.cpp:

|  |
| --- |
| #include "String.hpp"  #include <cstddef> //std::size\_t  #include <algorithm>  #include <cstring>  #include <stdexcept>  namespace drmatters {  String::String() : curSize(0), capacity(0), pStr(nullptr) {  }    String::String(const String& toCpy) : curSize(toCpy.curSize),  capacity(toCpy.capacity),  pStr(toCpy.capacity ? new char[toCpy.capacity] : nullptr) {    std::copy(toCpy.pStr, toCpy.pStr + curSize, pStr);  }    String::String(const char\* cStr) : curSize(strlen(cStr) + 1),  capacity(strlen(cStr) + 1),  pStr(new char[strlen(cStr) + 1]) {  memcpy(pStr, cStr, curSize);  }    String::~String() {  delete[] pStr;  }    String& String::operator=(String str) { //parameter taken by-value!  swap(\*this, str);    return \*this;  }    String& String::operator=(const char\* cStr) {  String tempS(strlen(cStr) + 1);  //on this stage bad\_alloc can be thrown  strcpy(tempS.pStr, cStr);  swap(\*this, tempS);    return \*this;  }    char& String::operator[](size\_t pos) {  return pStr[pos];  }  const char& String::operator[](size\_t pos) const {  return pStr[pos];  }  char& String::at(size\_t pos) {  if(pos >= curSize) throw std::out\_of\_range("out of range in String::at");//out of range  return pStr[pos];  }  const char& String::at(size\_t pos) const {  if(pos >= curSize) throw std::out\_of\_range("out of range in String::at");//out of range  return pStr[pos];  }    bool String::operator==(const String& toCheck) {  if(curSize != toCheck.curSize) {  return false;  }  for(size\_t i = 0; i < curSize; ++i) {  if(pStr[i] != toCheck.pStr[i]) {  return false;  }  }  return true;  }  bool String::operator!=(const String& toCheck) {  if(curSize != toCheck.curSize) {  return true;  }  for(size\_t i = 0; i < curSize; ++i) {  if(pStr[i] != toCheck.pStr[i]) {  return true;  }  }  return false;  }    void String::clear() {  delete [] pStr;  pStr = nullptr;  curSize = 0;  capacity = 0;  }  bool String::empty() const {  return curSize == 0;  }  String& String::operator +=(const String& str) {  String tempS(curSize + str.curSize);    std::copy(pStr, pStr + curSize, tempS.pStr);  std::copy(str.pStr, str.pStr + str.curSize, tempS.pStr + curSize);    swap(\*this, tempS);  return \*this;  }  String& String::operator+(const String& str) {  String tempS(curSize + str.curSize);    std::copy(pStr, pStr + curSize, tempS.pStr);  std::copy(str.pStr, str.pStr + str.curSize, tempS.pStr + curSize);    return \*this;  }  String& String::insert(const String& str, const size\_t pos) {  if((curSize + str.curSize) > capacity) {  increaseCapacity(curSize + str.curSize - capacity);  }    String tempS(pStr + pos);  std::copy(str.pStr, str.pStr + str.curSize, pStr + pos);  \*this += tempS;    return \*this;  }    const char\* String::data() const {  return pStr;  }  const char\* String::c\_str() const {  return pStr;  }  size\_t String::size() const {  return curSize - 1;  }  void String::increaseCapacity(size\_t n) {  String tempS(capacity + n);  tempS.curSize = curSize;  std::copy(pStr, pStr + curSize, tempS.pStr);    swap(\*this, tempS);  }  void String::pushBack(char c) {  if(capacity <= curSize) {  increaseCapacity(10);  }  pStr[curSize] = c;  curSize++;  }  std::ostream& operator<<(std::ostream& stream, String& obj) {  if(obj.pStr != nullptr) {  for(size\_t i = 0; i < obj.size(); i++) {  stream << obj.pStr[i];  }  }  return stream;  }  std::istream& operator>>(std::istream& stream, String& obj) {  char tempC;  while(!(stream.rdstate() & std::istream::eofbit)) {  stream.get(tempC);  if(tempC == '\n') {  obj.pushBack('\0');  break;  }  obj.pushBack(tempC);  }  obj.curSize++;  return stream;  }  } |