ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Факультет безопасности информационных технологий

Дисциплина:

«Программирование»

ОТЧЕТ ПО ОБОБЩЕННОЙ ЛАБОРАТОРНОЙ РАБОТЕ

“Разработка и иерархия классов”

Выполнила:

Студентка группы N3149

Синюта А. А.

Проверил:

Безруков В. А.

Санкт-Петербург

2022г.

Содержание

1 Введение

Объектно-ориентированное программирование используется при разработке программного обеспечения. Объектно-ориентированная парадигма имеет свои особенности, которые делают ее основным стандартом разработки. Объект является совокупностью данных, характеризующих его состояние, и процедур обработки. Поведение и взаимодействие объектов, выделяемых из предметной области, моделируется в программе. ООП дает возможность создавать приложения с разнообразными возможностями и четкой структурой. Также позволяет дорабатывать части программы, не меняя приложение целиком.

Цель:
Изучение ООП на примере языка C++, изучение классов, практика работы с ними. Приобретение навыков разработки программ и создание базового класса «СТРОКА» и производные от него классы «СТРОКА_ИДЕНТИФИКАТОР» и «ДЕСЯТИЧНАЯ_СТРОКА».

Задачи:
Разработка классов: базовый класс «СТРОКА», производные классы «СТРОКА_ИДЕНТИФИКАТОР» и «ДЕСЯТИЧНАЯ_СТРОКА» согласно техническому заданию.

2 Отчет

2.1 Техническое задание

1. Описать базовый класс Строка.

    1.1. Обязательные члены класса:

    1. указатель на char – хранит адрес динамически выделенной памяти для размещения символов строки,
    2. значение типа int – хранит длину строки в байтах.

    1.2. Обязательные методы должны выполнять следующие действия:

    1. конструктор без параметров;
    2. конструктор, принимающий в качестве параметра Си-строку (заканчивается нулевым байтом);
    3. конструктор, принимающий в качестве параметра символ (char);
    4. конструктор копирования;
    5. деструктор;
    6. проверка, пуста ли строка.

2.Производный от СТРОКА класс СТРОКА_ИНДЕНТИФИКАТОР.

Строки данного класса строятся по правилам записи идентификаторов в СИ, и могут включать в себя только те символы, которые могут входить в состав Си-идентификаторов. Если исходные данные противоречат правилам записи идентификатора, то создается пустая СТРОКА_ИДЕНТИФИКАТОР.

    2.1. Обязательные методы:

    1. Конструктор без параметров;
    2. Конструктор, принимающий в качестве параметра Си-строку (заканчивается нулевым байтом);
    3. Конструктор копирования;
    4. Деструктор;

    2.2. Переопределить следующие операции:

    1. Присваивание (=);
    2. Сложение (+) – операция конкатенации строк;
    3. Индексное выражение [].

3. Производный от СТРОКА класс ДЕСЯТИЧНАЯ_СТРОКА.

Строки данного класса могут содержать только символы десятичных цифр и символы - и +, задающие знак числа. Символы - или + могут находиться только в первой позиции числа, причем символ + может отсутствовать, в этом случае число считается положительным. Если в составе инициализирующей строки будут встречены любые символы, отличные от допустимых, ДЕСЯТИЧНАЯ_СТРОКА принимает нулевое значение.

Содержимое данных строк рассматривается как десятичное число.

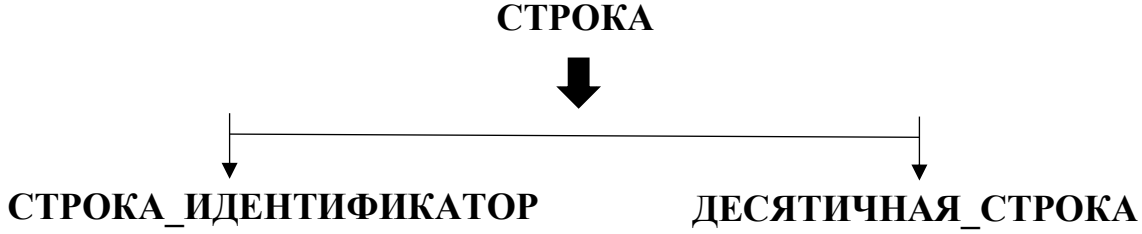
3.1. Обязательные методы:

1. Конструктор без параметров;
2. Конструктор, принимающий в качестве параметра Си-строку (заканчивается нулевым байтом);
3. Конструктор копирования;
4. Деструктор;

3.2. Переопределить следующие операции:

1. Присваивание (=);
2. Сложение (+) - арифметическая сумма строк.

Разработать иерархию классов по следующей схеме:



с обязательной поддержкой заданных членов и методов.

Написать тестовую программу, которая:

1. Динамически выделяет массив указателей на базовый класс (4–6 шт.);
2. Заполняет этот массив указателями на производные классы, при этом экземпляры производных классов создаются динамически с заданием начальных значений;
3. Для созданных экземпляров производных классов выполняется проверка всех разработанных методов (в соответствии с вариантом задания), с выводом исходных данных и результатов на дисплей.

4. Для конструкторов копирования каждого класса предусмотреть диагностическую печать количества его вызовов в заданное место дисплея, (рекомендуется использовать статические члены класса).

2.2 СТРОКА

2.2.1 Описание класса

Название класса: String. Является базовым классом для классов IdentStr и DecStr. Все методы прокомментированы в исходном коде, и их предназначение соответствует названию. Код, в котором описывается класс, разбит на два файла. Первый – String.h – заголовочный файл, в нем объявлены данные-члены, методы и конструкторы класса, представленные ниже. Второй – String.cpp – содержит тело функций-методов и конструкторов, которые были объявлены в заголовочном файле.

2.2.2 Структура класса

Поля класса объявлены как protected:

1. char* pCh – указатель на массив символов.
2. int len – переменная, хранящая длину строки.

Методы класса строка:

1. String(int = 0) - конструктор без параметров;
2. String(char) – конструктор, принимающий в качестве параметра один символ;
3. String(const char*) – конструктор, в качестве параметра принимающий Си-строку;
4. String(const String&) – конструктор копирования;
5. ~String() – деструктор;
6. void Show(void) – функция вывода значения и длины строки.
7. char* GetStr(void) const { return pCh; } – функция, возвращающая строку;
8. int GetLen(void) const { return len; } – функция, возвращающая длину строки.

2.3 СТРОКА_ИДЕНТИФИКАТОР

2.3.1 Описание класса

Название класса: IdentStr. Данный класс является производным классом класса String. Все методы прокомментированы в исходном коде, и их предназначение соответствует названию. Код, в котором описывается класс, разбит на два файла. Первый – IdentStr.h – заголовочный файл, в нем объявлены данные-члены, методы и конструкторы класса, представленные ниже. Второй – IdentStr.cpp – содержит тело функций-методов и конструкторов, которые были объявлены в заголовочном файле.

2.3.2 Структура класса

Методы данного класса:

1. IdentStr (int = 0) – конструктор без параметров;
2. IdentStr(char) – конструктор, принимающий в качестве параметра символ;
3. IdentStr (const char *) – конструктор, принимающий в качестве параметра Си-строку;
4. IdentStr (const IdentStr&) – конструктор копирования;
5. ~ IdentStr () – деструктор;
6. IdentStr& operator = (const IdentStr&) – переопределение оператора присваивания.
7. friend IdentStr operator + (const IdentStr&, const IdentStr& /*возможно использование const char* для конкатенации строки и объекта*/) – переопределение операции, реализующая конкатенацию строк (в виде объектов);
8. char& operator [] (int) – переопределение операции индексного выражения.
9. IdentStr operator ~ () – реверс строки.

2.4 ДЕСЯТИЧНАЯ_СТРОКА

2.4.1 Описание класса

Название класса: DecStr. Является производным классом класса String. Все методы прокомментированы в исходном коде, и их предназначение соответствует названию. Код, в котором описывается класс, разбит на два файла. Первый – DecStr.h – заголовочный файл, в нем объявлены данные-члены, методы и конструкторы класса, представленные ниже. Второй – DecStr.cpp – содержит тело функций-методов и конструкторов, которые были объявлены в заголовочном файле.

2.4.2 Структура класса

Методы класса:

1. DecStr (int = 0) – конструктор без параметров,
2. DecStr(const char*) – конструктор, принимающий в качестве параметра Си-строку,
3. DecStr(const DecStr&) – конструктор копирования,
4. ~ DecStr() – деструктор,
5. DecStr& operator = (const DecStr&) – переопределение оператора присваивания,
6. friend DecStr operator + (const DecStr&, const DecStr&) – переопределение арифметической суммы объектов.
7. friend DecStr operator + (const DecStr&, const int) – переопределение суммы объекта и числа.
8. friend DecStr operator - (const DecStr&, const DecStr&) – переопределение разницы объекта и числа.

3 Заключение

В лабораторной работе мною были созданы классы, такие как базовый класс «СТРОКА» и производные от него классы «СТРОКА_ИДЕНТИФИКАТОР», «ДЕСЯТИЧНАЯ_СТРОКА». Так же были реализованы и протестированы различные методы созданных классов. При проверке было выявлено, что тестируемые классы работают корректно. Вывод работы программы можно увидеть в консольном приложении. Таким образом, цель мною была достигнута.

4 Список использованных источников

1) Б.И. Березин. Начальный курс С и С++. – М.:Издательство Диалог-МИФИ, 2005 г. – 248 с.
2) Павловская Т.А. С/С++. Программирование на языке высокого уровня / Учебное пособие. - СПб: Издательство ПИТЕР, 2001 г. - 464 с.
3) Р. Лафоре. Объектно-ориентированное программирование в С++. 4-е издание. – СПб.: Издательство ПИТЕР, 2004 г. – 902 с.
4) Б. Страуструп. Язык программирования С++. Специальное издание. Пер. с англ. – М.: Издательство Бином, 2011 г. – 1136 с.

5 Приложение

5.1 Файл String.h

```
#ifndef __STRING__
#define __STRING__

class String
{
protected:
 int len;   //длина строки
 char* pCh; //копирование

public:

 String(int = 0);
 String(char);
 String(const char*);

 String(const String&);     //конструктор копирования
 ~String();                 //деструктор

 char* GetStr(void) const { return pCh; }
 int GetLen(void) const { return len; }
 void Show(void);           //вывод строки и её длины на экран
};
#endif
```

5.2 Файл String.cpp

```
#include <iostream>
#include "String.h"
using namespace std;

String::String(int val) : len(val), pCh(new char[len + 1])
{
	if (val == 0)
		pCh[0] = '\0';
	cout << "String::String(int val) " << endl;
};

String::String(char ch) : len(1), pCh(new char[len + 1])
{
	pCh[0] = ch;
	pCh[1] = '\0';
	cout << "String::String(char ch)" << endl;
};

String::String(const char* S) : len(strlen(S)), pCh(new char[len + 1])
{
	strcpy_s(pCh, len + 1, S);
	cout << "String::String(const char* S)" << endl;
};

String::String(const String& from) : len(strlen(from.pCh)), pCh(new
char[from.len + 1])
{
	strcpy_s(pCh, from.len + 1, from.pCh);
	cout << "String::String(const String& from)" << endl;
};

String::~String()
{
	if (pCh) delete[] pCh;
	cout << "String::~String() " << endl;
};

void String::Show(void)
{
	cout << "pCh = " << pCh << endl;
	cout << "len = " << len << endl;
};
```

5.3 Файл IdentStr.h

```
#ifndef __IDENTSTR__
#define __IDENTSTR__
#include "String.h"
class IdentStr : public String
{
public:

	IdentStr(int = 0);
	IdentStr(char);
	IdentStr(const char*);
	IdentStr(const IdentStr&);
	~IdentStr();

	//переопределение операторов
	IdentStr& operator = (const IdentStr&); //оператор присваивания
	char& operator [] (int);   // обращение по индексу
	IdentStr operator ~ ();    // реверс строки

	//дружественные функции [3]
	//сложение (конкатенация) двух объектов
	friend IdentStr operator + (const IdentStr&, const IdentStr&);
	//конкатенация си-строки и объекта
	friend IdentStr operator + (const char*, const IdentStr&);
	friend IdentStr operator + (const IdentStr&, const char*);
};
#endif
```

5.4 Файл IdentStr.cpp

```
#include <iostream>
#include <string.h>
#include "IdentStr.h"
#include "String.h"
using namespace std;

IdentStr::IdentStr(int val) : String(val)
{
 cout << "IdentStr::IdentStr(int val) " << endl;
}

IdentStr::IdentStr(char ch) : String(ch)
{
 if (!((pCh[0]>='a' && pCh[0]<='z')||(pCh[0]>='A' && pCh[0]<= 'Z'))) {
       cout << "Bad symbol; pCh[0] = " << pCh[0] << endl;
       if (pCh) delete[] pCh;
       len = 0;
       pCh = new char[len + 1];
       pCh[0] = '\0';
       return;
 }
 cout << "IdentStr::IdentStr(char ch) " << endl;
};

IdentStr::IdentStr(const char* str) : String(str)
{
  const char* keyword[] = {"alignas","alignof", "and", "and_eq", "asm",
"auto","bitand","bitor","bool","break","case","catch","char","char16_t"
,"char32_t","class","compl","const","constexpr","const_cast","continue"
,"decltype","default","delete","do","double","dynamic_cast","else","enu
m","explicit","export","extern","false","float","for","friend","goto","
if","inline","int","long","mutable","namespace","new","noexcept","not",
"not_eq","nullptr","operator","or","or_eq","private","protected","publi
c","register","reinterpret_cast","return","short","signed","sizeof","st
atic","static_assert","static_cast","struct","switch","template","this"
,"thread_local","throw","true","try","typedef","typeid","typename","uni
on","unsigned","using","virtual","void","volatile","wchar_t","while","x
or","xor_eq" };

 for (int i = 0; i < 84; i++) {
       if (strcmp(pCh, keyword[i]) == 0)
       {
             cout << "Bad string pCh = " << pCh << endl;
             if (pCh)delete[]pCh;
             len = 0;
             pCh = new char[len + 1];
             pCh[0] = '\0';
             return;
       }
 }
```

```
      if (!((pCh[0]>='a' && pCh[0]<='z')||(pCh[0]>='A' && pCh[0] <= 'Z')))
      {
            cout << "Bad symbol: pCh[0] = " << pCh[0] << endl;
            if (pCh) delete[] pCh;
len = 0;
pCh = new char[len + 1];
pCh[0] = '\0';
return;
      }

for (int i = 1; i < len; i++)
{
            if (!((pCh[i]>='a' && pCh[i]<='z') || (pCh[i]>='A' && pCh[i] <= 'Z') ||
            (pCh[i] >= '0' && pCh[i] <= '9') || (pCh[i] == '_')))
            {
cout << "Bad string: pCh[" << i << "] = " << pCh[i] << endl;
if (pCh)
                        delete[] pCh;
len = 0;
pCh = new char[len + 1];
pCh[0] = '\0';
return;
}
}

      cout << "IdentStr::IdentStr(const char* str) " << endl;
};

IdentStr::IdentStr(const IdentStr& from) : String(from)
{
  cout << "IdentStr::IdentStr(const IdentStr& from) " << endl;
};

IdentStr::~IdentStr()
{
  cout << "IdentStr::~IdentStr()" << endl;
};

IdentStr& IdentStr:: operator=(const IdentStr& S)
{
      if (&S != this)
      {
delete[] pCh;
len = strlen(S.pCh);
pCh = new char[len + 1];
strcpy_s(pCh, len + 1, S.pCh);
cout<<"IdentStr& IdentStr:: operator = "<< endl;
return *this;
      }
};
```

```
char& IdentStr:: operator [] (int index)
{
if (index >= 0 && index < len)
     {
cout << "char& IdentStr:: operator [] " << endl;
return pCh[index];
}

return pCh[0];
}

IdentStr IdentStr:: operator~()
{
     int i, j;
     char tmp;

     for (i = 0, j = len - 1; i < len / 2; i++, j--)
     {
          tmp = pCh[i];
          pCh[i] = pCh[j];
          pCh[j] = tmp;
     }

     const char* keyword[]={"alignas","alignof","and","and_eq", "asm",
    "auto","bitand","bitor","bool","break","case","catch","char","char16_t"
    ,"char32_t","class","compl","const","constexpr","const_cast","continue"
    ,"decltype","default","delete","do","double","dynamic_cast","else","enu
    m","explicit","export","extern","false","float","for","friend","goto","
    if","inline","int","long","mutable","namespace","new","noexcept","not",
    "not_eq","nullptr","operator","or","or_eq","private","protected","publi
    c","register","reinterpret_cast","return","short","signed","sizeof","st
    atic","static_assert","static_cast","struct","switch","template","this"
    ,"thread_local","throw","true","try","typedef","typeid","typename","uni
    on","unsigned","using","virtual","void","volatile","wchar_t","while","x
    or","xor_eq" };

     for (int i = 0; i < 84; i++)
     {
          if (strcmp(pCh, keyword[i]) == 0)
          {
               cout << "Bad string pCh = " << pCh << endl;
               if (pCh)
                    delete[]pCh;

               len = 0;
               pCh = new char[len + 1];
               pCh[0] = '\0';
               return *this;
          }
     }
```

```
      if (!((pCh[0]>='a' && pCh[0]<='z') || (pCh[0]>='A' && pCh[0]<= 'Z')))
      {
            cout << "Bad symbol: pCh[0] = " << pCh[0] << endl;
            if (pCh) delete[] pCh;
            len = 0;
            pCh = new char[len + 1];
            pCh[0] = '\0';
            return *this;
      }
      cout << "IdentStr IdentStr:: operator~()" << endl;
      return *this;
     }

      IdentStr operator + (const IdentStr& pobj1, const IdentStr& pobj2)
      {
            IdentStr tmp(pobj1.GetLen() + pobj2.GetLen());
            int i = 0, j = 0;
            while (tmp.pCh[i++] = pobj1.pCh[j++]);
            i--;
            j = 0;
            while (tmp.pCh[i++] = pobj2.pCh[j++]);
            cout << "IdentStr operator + [const IdentStr]" << endl;
            return tmp;
      }

IdentStr operator + (const IdentStr& pobj1, const char* pobj2)
{
IdentStr tmp1(pobj2);
IdentStr tmp(pobj1.GetLen() + tmp1.GetLen());
int i = 0, j = 0;
while (tmp.pCh[i++] = pobj1.pCh[j++]);
i--;
j = 0;
while (tmp.pCh[i++] = tmp1.pCh[j++]);

      cout<< "IdentStr operator + (const IdentStr&, const char*)" << endl;
return tmp;
}

IdentStr operator + (const char* pobj1, const IdentStr& pobj2)
{
IdentStr tmp2(pobj1);
IdentStr tmp(pobj2.GetLen() + tmp2.GetLen());
int i = 0, j = 0;
while (tmp.pCh[i++] = tmp2.pCh[j++]);
i--;
j = 0;
while (tmp.pCh[i++] = pobj2.pCh[j++]);

      cout<< "IdentStr operator + (const IdentStr&, const char*)" << endl;
return tmp;
}
```

5.5 Файл DecStr.h

```
#ifndef __DECSTR__
#define __DECSTR__
#include "String.h"
class DecStr : public String
{
public:

          DecStr(int = 0);
          DecStr(const char*);
          DecStr(const DecStr&);
          ~DecStr();

          DecStr& operator = (const DecStr&);

          //объект + объект
          friend DecStr operator + (const DecStr&,const DecStr&);

          //объект + число (int)
          friend DecStr operator + (const DecStr&, const int);

          //объект – объект
          friend DecStr operator - (const DecStr&, const DecStr&);
};
#endif
```

5.6 Файл DecStr.cpp

```
#include <iostream>
#include "DecStr.h"
#include "String.h"

using namespace std;

DecStr::DecStr(int val) : String(val)
{
    cout << "DecStr::DecStr(int val) " << endl;
}

DecStr::DecStr(const char* str) : String(str)
{

    if (!((pCh[0] >= '1' && pCh[0] <= '9') || (pCh[0] == '-' || pCh[0] ==
'+')))
    {
          cout << "Invalid symbol, pCh[0] = " << pCh[0] << endl;
          if (pCh) delete[] pCh;
          len = 0;
          pCh = new char[len + 1];
          *pCh = '\0';
          return;
    }

    for (int i = 1; i < len; i++)
          if (!(pCh[i] >= '0' && pCh[i] <= '9'))
          {
                cout << "Invalid string: pCh[" << i << "] = " << pCh[i] <<
endl;
                if (pCh) delete[] pCh;
                len = 0;
                pCh = new char[len + 1];
                *pCh = '\0';
                return;
          }

    cout << "DecStr::DecStr(const char* str) " << endl;
}

DecStr::DecStr(const DecStr& from) : String(from)
{
    cout << "DecStr::DecStr(const DecStr& from) " << endl;
}

DecStr:: ~DecStr()
{
    cout << "DecStr:: ~DecStr() " << endl;
}
```

```
DecStr& DecStr:: operator = (const DecStr& DS)
{
    if (&DS != this)
    {
        delete[] pCh;
        len = strlen(DS.pCh);
        pCh = new char[len + 1];
        strcpy_s(pCh, len + 1, DS.pCh);
    }
    cout << "DecStr& DecStr:: operator = (const DecStr& DS)" << endl;
    return *this;
}

DecStr operator + (const DecStr& pobj1, const DecStr& pobj2)
{
    int num1, num2;
    DecStr tmp(pobj1);
    num1 = atoi(tmp.GetStr());
    num2 = atoi(pobj2.GetStr());
    int A = num1 + num2;
    char* pTmpCh;
    if (tmp.len >= pobj2.len)
    {
        pTmpCh = new char[tmp.len + 1];
        _itoa_s(A, pTmpCh, tmp.len + 1, 10);
    }
    else
    {
        pTmpCh = new char[pobj2.len + 1];
        _itoa_s(A, pTmpCh, pobj2.len + 1, 10);
    }
    if (tmp.pCh) delete[] tmp.pCh;
    tmp.pCh = pTmpCh;
    tmp.len = strlen(tmp.pCh);
    return tmp;
}

DecStr operator + (const DecStr& pobj1, const int pobj2)
{
    int num1, num2;
    DecStr tmp(pobj1);
    num1 = atoi(tmp.GetStr());
    num2 = pobj2;
    long long int A = static_cast<long long>(num1) + num2;
    if ((A > 2147483647) || (A < -2147483648)) {
        delete[] tmp.pCh;
        tmp.len = 0;
        tmp.pCh = new char[tmp.len + 1];
        tmp.pCh[0] = '\0';
        cout << "Bad srtoka" << endl;
        return tmp;
```

```
    }
    char* pTmpCh;
    int count = 0;
    if (num2 < 0)
    {
         count++;
         num2 *= -1;
    };

    while (num2)
    {
         count++;
         num2 /= 10;
    };

    if (tmp.len >= count)
    {
         pTmpCh = new char[tmp.len + 2];
         _itoa_s(A, pTmpCh, tmp.len + 2, 10);
    }
    else
    {
         pTmpCh = new char[count + 2];
         _itoa_s(A, pTmpCh, count + 2, 10);
    };

    if (tmp.pCh) delete[] tmp.pCh;
    tmp.pCh = pTmpCh;
    tmp.len = strlen(tmp.pCh);
    return tmp;
}

DecStr operator - (const DecStr& pobj1, const DecStr& pobj2)
{
    int num1, num2;
    DecStr tmp(pobj1);
    num1 = atoi(tmp.GetStr());
    num2 = atoi(pobj2.GetStr());
    int A = num1 - num2;
    char* pTmpCh;
    if (tmp.len >= pobj2.len)
    {
         pTmpCh = new char[tmp.len + 1];
         _itoa_s(A, pTmpCh, tmp.len + 1, 10);
    }
    else
    {
         pTmpCh = new char[pobj2.len + 1];
         _itoa_s(A, pTmpCh, pobj2.len + 1, 10);
    }
    if (tmp.pCh)
```

```
        delete[] tmp.pCh;
    tmp.pCh = pTmpCh;
    tmp.len = strlen(tmp.pCh);
    return tmp;
}
```

5.7 Файл ASM.asm

```
.586
.MODEL FLAT, STDCALL
PUBLIC fun1

_DATA SEGMENT
StrMas db 11 dup(?), 0
dec1 dd 10
_DATA ENDS

_TEXT SEGMENT
fun1 PROC dec1: DWORD
    Lea ebx, StrMas
    mov ecx, 11
metka1:
    mov BYTE PTR [ebx], ' '
    inc ebx
    loop metka1
    mov eax, dec1
    push eax
    or eax, eax
    jns metka2
    neg eax
metka2:
    xor edx, edx
    div dec1
    add dx, '0'
    dec ebx
    mov BYTE PTR [ebx], dl
    inc ecx
    or eax, eax
    jnz metka2
    pop eax
    or eax, eax
    jns metka3
    dec ebx
    mov BYTE PTR [ebx], ' '
    inc ecx
metka3:
    mov eax, ebx
    ret
fun1 ENDP
_TEXT ENDS
END
```

5.8 Основной файл main.cpp

```
#include <iostream>
#include "String.h"
#include "IdentStr.h"
#include "DecStr.h"

using namespace std;

int main()
{
    String obj1("ITMO");
    String obj2 = obj1;
    obj2.Show();

    IdentStr obj3("hello");
    obj3.Show();
    IdentStr obj4 = ~obj3;
    obj4.Show();
    obj4 = obj4 + (~obj3);
    obj4.Show();

    IdentStr obj5("IT");
    obj5 = "FB" + obj5;
    obj5.Show();
    obj5 = obj5 + "MO";
    obj5.Show();

    IdentStr obj6("int")
    obj6.Show();

    DecStr obj7("13357");
    DecStr obj8("-357");
    obj7 = obj7 + 3;
    DecStr obj9 = obj7 + obj8;
    obj7.Show();
    obj9.Show();

    String **pStr = new String*[6];
    pStr[0] = new String("Hello");
    pStr[1] = new IdentStr("World");
    pStr[2] = new DecStr("-5");
    pStr[3] = new IdentStr("ITMO");
    pStr[4] = new DecStr("65535");
    pStr[5] = new DecStr("&255");
    pStr[0]->Show();
    ((DecStr*)pStr[4])->Show();
    delete[] pStr;
    return 0;
}
```

5.9 Скриншоты работы программы

1) Работа класса String

```
String::String(const char* S)
String::String(const String& from)
pCh = ITMO
len = 4
String::~String()
String::~String()
```

2) Работа класса IdentStr

```
String::String(const char* S)
IdentStr::IdentStr(const char* str)
pCh = hello
len = 5
IdentStr IdentStr:: operator~()
String::String(const String& from)
IdentStr::IdentStr(const IdentStr& from)
pCh = olleh
len = 5
IdentStr IdentStr:: operator~()
String::String(const String& from)
IdentStr::IdentStr(const IdentStr& from)
String::String(int val)
IdentStr::IdentStr(int val)
IdentStr operator + [const IdentStr]
String::String(const String& from)
IdentStr::IdentStr(const IdentStr& from)
IdentStr::~IdentStr()
String::~String()
IdentStr& IdentStr:: operator =
IdentStr::~IdentStr()
String::~String()
IdentStr::~IdentStr()
String::~String()
pCh = ollehhello
len = 10
String::String(const char* S)
IdentStr::IdentStr(const char* str)
String::String(const char* S)
IdentStr::IdentStr(const char* str)
```

3) Работа класса DecStr

```
DecStr::DecStr(const char* str)
String::String(const String& from)
DecStr::DecStr(const DecStr& from)
String::String(const String& from)
DecStr::DecStr(const DecStr& from)
DecStr:: ~DecStr()
String::~String()
DecStr& DecStr:: operator = (const DecStr& DS)
DecStr:: ~DecStr()
String::~String()
String::String(const String& from)
DecStr::DecStr(const DecStr& from)
String::String(const String& from)
DecStr::DecStr(const DecStr& from)
DecStr:: ~DecStr()
String::~String()
```

```
pCh = 13360
len = 5
pCh = 13003
len = 5
DecStr:: ~DecStr()
String::~String()
DecStr:: ~DecStr()
String::~String()
DecStr:: ~DecStr()
String::~String()
```

4) Динамическое выделение памяти

```
String::String(const char* S)
String::String(const char* S)
IdentStr::IdentStr(const char* str)
String::String(const char* S)
DecStr::DecStr(const char* str)
String::String(const char* S)
IdentStr::IdentStr(const char* str)
String::String(const char* S)
DecStr::DecStr(const char* str)
String::String(const char* S)
Invalid symbol, pCh[0] = &
pCh = Hello
len = 5
pCh = 65535
len = 5
```

5) Попытка создания строки-идентификатора, совпадающей с ключевым словом C++ (int)

```
String::String(const char* S)
Bad string pCh = int
pCh =
len = 0
IdentStr::~IdentStr()
String::~String()
```

6) Сложение объекта и строки слева и справа

```
IdentStr::IdentStr(const IdentStr& from)
IdentStr::~IdentStr()
String::~String()
IdentStr::~IdentStr()
String::~String()
IdentStr& IdentStr:: operator =
IdentStr::~IdentStr()
String::~String()
pCh = FBIT
len = 4
String::String(const char* S)
IdentStr::IdentStr(const char* str)
String::String(int val)
IdentStr::IdentStr(int val)
IdentStr operator + (const IdentStr&, const char*)
String::String(const String& from)
IdentStr::IdentStr(const IdentStr& from)
IdentStr::~IdentStr()
String::~String()
```

```
IdentStr::~IdentStr()
String::~String()
IdentStr& IdentStr:: operator =
IdentStr::~IdentStr()
String::~String()
pCh = FBITMO
len = 6
IdentStr::~IdentStr()
String::~String()
```