Государственное ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ   
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«Донецкий национальный технический университет»

Д09.03.04-ПОИС.20-20/5847.ЛР

***Кафедра*** искусственного интеллекта  
 и системного анализа

Лабораторная работа №2

по дисциплине "Объектно-ориентированное программирования"

на тему: "Одиночное и множественное наследование. Полиморфизм."

Проверили:

\_\_\_\_\_\_\_\_\_\_\_\_ асс. А.П. Семёнова

(дата, подпись)

\_\_\_\_\_\_\_\_\_\_\_\_ асс. И.В. Савицкая

(дата, подпись)

Выполнил:

\_\_\_\_\_\_\_\_\_\_\_\_\_ ст.гр.ПИ-20г М.А. Евсеев

(дата, подпись)

Донецк – 2021

Тема: Одиночное и множественное наследование. Полиморфизм.

Цель работы: Приобретение практических навыков реализации наследования и иерархии классов на языке С++.

Задание. Создать и протестировать следующие классы:

1. Абстрактный класс

Создать абстрактный класс Container, в котором определить чисто виртуальные функции для добавления, удаления и определения количества элементов в контейнере.

1. Класс одиночного наследования.

2.1 Модифицировать класс динамической структуры данных, содержащей строки (л.р. №1), чтобы он был производным от класса Container.

2.2 Создать 2 различных класса, производных от класса динамической структуры, позволяющие сортировать и фильтровать выходные данные. Для этого вводятся виртуальные функции, соответственно, для сортировки – функция для сравнения 2 строк, для фильтрации – предикат, определяющий, будет ли строка присутствовать в выборке фильтра.

3. Класс множественного наследования.

Создать класс, осуществляющий одновременно сортировку и фильтрацию.

Вариант:

Сортировка – 5) По возрастанию количеству пробелов в строке.

Фильтрация – 7) Строки четной длины.

Ход работы:

Листинг:

#include <iostream>

#include <string>

#include <windows.h>

using namespace std;

int tabs = 0;

int compare(string s1, string s2);

class Sort;

class A

{

    friend class Filt;

    friend class Sort;

protected:

    int k;

    int count;

    string s;

    A \*right = 0, \*left = 0;

    void c\_plus(A \*a)

    {

        if (a == 0)

            return;

        a->count++;

        c\_plus(a->right);

        c\_plus(a->left);

    }

    int spaces(string a, string b)

    {

        int j = 0;

        for (size\_t i = 0; i < a.size(); i++)

            (a[i] == ' ') ? j++ : (0);

        for (size\_t i = 0; i < b.size(); i++)

            (a[i] == ' ') ? j-- : (0);

        if (j != 0)

        {

            if (j > 0)

                return 1;

            else

                return -1;

        }

        return 0;

    }

public:

    A() : s("empty"), count(1), k(1) {}                     //по умолчанию

    A(const string &s, int c) : s(s), count(c) {}           // с параметрами

    A(const A &a) : s(a.s), right(a.right), left(a.left) {} // копирования

    string get\_s()

    {

        return s;

    }

    int get\_c()

    {

        return count;

    }

    //увеличивает кол-во повторений всех значений веток данного дерева

    A \*operator++(int)

    {

        c\_plus(this);

        return this;

    }

    //увеличивает кол-во повторений всех значений веток данного дерева

    A \*operator++()

    {

        return (\*this)++;

    }

    //удаляет самый низший правый элемент

    A \*operator--()

    {

        if (this->right != 0)

        {

            if (this->right->right != 0)

            {

                --(\*this->right);

            }

            else

            {

                this->right = del\_tree(this->right);

                this->right = 0;

                return this;

            }

        }

        return this;

    }

    //удаляет самый низший левый элемент

    A \*operator--(int)

    {

        if (this->left != 0)

        {

            if (this->left->left != 0)

            {

                --(\*this->left);

            }

            else

            {

                this->left = del\_tree(this->left);

                this->left = 0;

                return this;

            }

        }

        return this;

    }

    //есть ли строка в дереве

    bool operator[](const string &a)

    {

        return find(this, a);

    }

    //удаляет ветку с этим значением

    A \*operator-(const string &a)

    {

        if (this)

        {

            A \*t = this;

            int x = compare(a, this->get\_s());

            cout << x << endl;

            if (!x)

                return del\_tree(this);

            if (x > 0)

                t->right = \*this->right - a;

            if (x < 0)

                t->left = \*this->left - a;

        }

        return this;

    }

    //добавление элемента

    A \*operator+(const string &a)

    {

        return tree(a, this);

    }

    bool operator>(A \*a)

    {

        if (this->k > a->k)

            return true;

        return false;

    }

    bool operator<(A \*a)

    {

        if (this->k < a->k)

            return true;

        return false;

    }

    bool operator==(A \*a)

    {

        if (this->k == a->k)

            return true;

        return false;

    }

    //коприрование

    A &operator=(A &a)

    {

        cout << a.count;

        k = a.k;

        count = a.count;

        s = a.s;

        left = new A(\*a.left);

        right = new A(\*a.right);

        return \*this;

    }

    //сравнение 2 строк

    int compare(string s1, string s2)

    {

        int a = s1.length() + 1, b = s2.length() + 1;

        char \*text1 = new char[a];

        char \*text2 = new char[b];

        for (size\_t i = 0; i < a; i++)

            text1[i] = s1[i];

        for (size\_t i = 0; i < b; i++)

            text2[i] = s2[i];

        int x = strcmp(text1, text2);

        free(text1);

        free(text2);

        return x;

    }

    //вывод дерева

    void print(A \*a)

    {

        if (!a)

            return; //Если ветки не существует - выходим. Выводить нечего

        tabs += 5;  //Иначе увеличим счетчик рекурсивно вызванных процедур

        //Который будет считать нам отступы для вывода

        print(a->left); //Выведем ветку и ее подветки слева

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

            cout << " "; //Потом отступы

        cout << a->count << " \_ " << a->k << " \_ "

             << "\"" << a->s << "\"" << endl; //Данные этой ветки

        print(a->right);                      //И ветки, что справа

        tabs -= 5;                            //После уменьшим кол-во отступов

    }

    bool find(A \*a, const string &s)

    {

        if (!a)

            return false;

        int x = compare(s, a->get\_s());

        if (!x)

            return true;

        if (x > 0)

            return find(a->right, s);

        if (x < 0)

            return find(a->left, s);

        return false;

    }

    A \*tree(const string &s, A \*a)

    {

        if (!a)

        {

            a = new A();

            a->s = s;

        }

        else

        {

            a->k++;

            int x = compare(s, a->get\_s());

            if (!x)

            {

                a->count++;

            }

            else if (x > 0)

            {

                a->right = tree(s, a->right);

            }

            else

            {

                a->left = tree(s, a->left);

            }

        }

        return a;

    }

    A \*del\_tree(A \*a)

    {

        if (a)

        {

            del\_tree(a->left);

            del\_tree(a->right);

            free(a);

        }

        return 0;

    }

    //~A() { del\_tree(this); } //деструктор

};

class Sort : virtual public A

{

    bool sort = false;

public:

    A \*a = 0;

    void sorted(bool a)

    {

        sort = a;

        (a) ? (cout << "Сортировка изменена: по возрастанию пробелов в строке" << endl) : (cout << "Стандартная сортировка" << endl);

    }

    Sort operator+(string const s)

    {

        a = tree(s, a);

    }

    A \*tree(const string &s, A \*a)

    {

        if (sort)

        {

            cout << "\_" << s << "\_" << endl;

            if (!a)

            {

                a = new A();

                a->s = s;

            }

            else

            {

                a->k++;

                int x = spaces(s, a->get\_s());

                if (!x)

                {

                    a->count++;

                }

                else if (x > 0)

                {

                    a->right = tree(s, a->right);

                }

                else

                {

                    a->left = tree(s, a->left);

                }

            }

        }

        else

        {

            if (!a)

            {

                a = new A();

                a->s = s;

            }

            else

            {

                a->k++;

                int x = compare(s, a->get\_s());

                if (!x)

                {

                    a->count++;

                }

                else if (x > 0)

                {

                    a->right = tree(s, a->right);

                }

                else

                {

                    a->left = tree(s, a->left);

                }

            }

        }

        return a;

    }

};

class Filt : virtual public A

{

    bool filt = false;

public:

    A \*a;

    void filted(bool x)

    {

        filt = x;

        if (x)

        {

            cout << "Фильтр включен\n";

        }

        else

        {

            cout << "Фильтр выключен\n";

        }

    }

    void print(A \*a)

    {

        if (!a)

            return; //Если ветки не существует - выходим. Выводить нечего

        tabs += 5;  //Иначе увеличим счетчик рекурсивно вызванных процедур

        //Который будет считать нам отступы для вывода

        print(a->left); //Выведем ветку и ее подветки слева

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

            cout << " "; //Потом отступы

        if (!(filt && ((a->s.size() % 2) == 1)))

            cout << a->count << " \_ " << a->k << " \_ "

                 << "\"" << a->s << "\"" << endl; //Данные этой ветки

        print(a->right);                          //И ветки, что справа

        tabs -= 5;                                //После уменьшим кол-во отступов

    }

};

class All : public Filt, public Sort

{

public:

    A \*a;

};

int main()

{

    setlocale(LC\_ALL, "Russian");

    SetConsoleCP(1251);

    SetConsoleOutputCP(1251);

    A \*node = 0;

    All a;

    string s;

    int n;

    a.filted(1);

    a.sorted(0);

    cin >> n;

    cin.ignore(32767, '\n');

    for (size\_t i = 0; i < n; i++)

    {

        getline(cin, s);

        a.a = a.tree(s, a.a);

    }

    cout << "\n\n\n";

    a.print(a.a);

    return 0;

}

Экранные формы

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1 – С Фильтром и стандартной сортировкой

Изображение выглядит как стрела

Автоматически созданное описание

Рисунок 2 – Без фильтра

Изображение выглядит как текст, экран, устройство, темный

Автоматически созданное описание

Рисунок 3 – Без фильтра и стандартной сортировки

Изображение выглядит как текст, экран, устройство, счетчик

Автоматически созданное описание

Рисунок 3 – С фильтром и нестандартной сортировкой

Вывод: приобрели практические навыки реализации наследования и иерархии классов на языке С++.