Здесь будет титульник, листай ниже

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

Иметь возможность доступа из текущего объекта к любому объекту системы, «мечта» разработчика программы.

Расширить функциональность базового класса:

- метод переопределения головного объекта для текущего в дереве иерархии. Метод должен иметь один параметр, указатель на объект базового класса, содержащий указатель на новый головной объект. Переопределение головного объект для корневого объекта недопустимо. Недопустимо создать второй корневой объект. Недопустимо при переопределении, чтобы у нового головного появились два подчиненных объекта с одинаковым наименованием. Новый головной объект не должен принадлежать к объектам из ветки текущего. Если переопределение выполнено, метод возвращает значение «истина», иначе «ложь»;
- метод удаления подчиненного объекта по наименованию. Если объект не найден, то метод завершает работу. Один параметр строкового типа, содержит наименование удаляемого подчиненного объекта;
- метод получения указателя на любой объект в составе дерева иерархии объектов согласно пути (координаты). В качестве параметра методу передать путь (координату) объекта. Координата задаться в следующем виде:
    - / - корневой объект;
    - //«имя объекта» - поиск объекта по уникальной имени от корневого (для однозначности уникальность требуется в рамках дерева);
    - . - текущий объект;
    - .«имя объекта» - поиск объекта по уникальной имени от текущего (для однозначности уникальность требуется в рамках ветви дерева от

текущего объекта);

- «имя объекта 1»[/«имя объекта 2»] . . . - относительная координата от текущего объекта, «имя объекта 1» подчиненный текущего;
- /«имя объекта 1»[/«имя объекта 2»] . . . - абсолютная координата от корневого объекта.

Примеры координат:

```
/
//ob_3
.
.ob_2
ob_2/ob_3
/ob_1/ob_2/ob_3
```

Если координата - пустая строка или объект не найден или определяется неоднозначно (дубляж имен на ветке, на дереве), тогда вернуть нулевой указатель.

Наименование объекта не содержит символы «.» и «/».

Система содержит объекты пяти классов, не считая корневого. Номера классов: 2,3,4,5,6.

Состав и иерархия объектов строиться посредством ввода исходных данных. Ввод организован как в версии № 2 курсовой работы. Единственное различие. В строке ввода первым указано не наименование головного объекта, а абсолютный путь к нему. При построении дерева уникальность наименования относительно множества непосредственно подчиненных объектов для любого головного объекта необходимо соблюдать. Если это требование исходя из входных данных нарушается, то соответствующий подчиненный объект не создается.

Добавить проверку допустимости исходной сборки. Собрать дерево невозможно, если по заданной координате головной объект не найден (например, ошибка в наименовании или еще не расположен на дереве объектов). Если номер класса объекта задан некорректно, то объект не создается.

Собранная система отрабатывает следующие команды:

- SET «координата» – устанавливает текущий объект;
- FIND «координата» – находит объект относительно текущего;
- MOVE «координата» – переопределить головной для текущего объекта, «координата» задает новый головной объект;
- DELETE «наименование объекта» – удалить подчиненный объект у текущего;
- END – завершает функционирование системы (выполнение программы).

Изначально, корневой объект для системы является текущим. При вводе данных в названии команд ошибок нет. Если при переопределении головного объекта нарушается уникальность наименований подчиненных объектов для нового головного, переопределение не производится.

1.1 Описание входных данных

Состав и иерархия объектов строиться посредством ввода исходных данных. Ввод организован как в версии № 2 курсовой работы. Единственное различие. В строке ввода первым указано не наименование головного объекта, а абсолютный путь к нему.

После ввода состава дерева иерархии построчно вводятся команды:

- SET «координата» – установить текущий объект;
- FIND «координата» – найти объект относительно текущего;
- MOVE «координата» – переопределить головной для текущего объекта, «координата» соответствует новому головному объекту;
- DELETE «наименование объекта» – удалить подчиненный объект у текущего;
- END – завершить функционирование системы (выполнение программы).

Команды SET, FIND, MOVE и DELETE вводятся произвольное число раз.

Команда END присутствует обязательно.

Пример ввода иерархии дерева объектов:

```
rootela
/ object_1 3
/ object_2 2
/object_2 object_4 3
/object_2 object_5 4
/ object_3 3
/object_2 object_3 6
/object_1 object_7 5
/object_2/object_4 object_7 3
endtree
FIND object_2/object_4
SET /object_2
FIND //object_7
FIND object_4/object_7
FIND .
FIND .object_7
FIND object_4/object_7
MOVE .object_7
SET object_4/object_7
MOVE //object_1
MOVE /object_3
END
```

1.2 Описание выходных данных

Первая строка:

```
Object tree
```

Со второй строки вывести иерархию построенного дерева как в работе версия №2.

При ошибке определения головного объекта, прекратить сборку, вывести иерархию уже построенного фрагмента дерева, со следующей строки сообщение:

```
The head object «координата головного объекта» is not found
```

и прекратить работу программы с кодом возврата 1.

Если при построении при попытке создания объекта обнаружен дубляж, то вывести:

«координата головного объекта» Dubbing the names of subordinate objects

Если дерево построено, то далее построчно вводятся команды.

Для команд SET если объект найден, то вывести:

Object is set: «имя объекта»

в противном случае:

The object was not found at the specified coordinate: «искомая координата объекта»

Для команд FIND вывести:

«искомая координата объекта» Object name: «наименование объекта»

Если объект не найден, то:

«искомая координата объекта» Object is not found

Для команд MOVE вывести:

New head object: «наименование нового головного объекта»

Если головной объект не найден, то:

«искомая координата объекта» Head object is not found

Если переопределить головной объект не удалось, то:

«искомая координата объекта» Redefining the head object failed

Если у нового головного объекта уже есть подчиненный с таким же именем, то вывести:

«искомая координата объекта» Dubbing the names of subordinate objects

При попытке переподчинения головного объекта к объекту на ветке, вывести:

«координата нового головного объекта» Redefining the head object failed

Для команды DELETE:

Если подчиненный объект удален, то вывести:

The object «абсолютный путь удаленного объекта» has been deleted

Если объект не найден, то ничего не выводить.

После команды END с новой строки вывести:

```
Current object hierarchy tree
```

Со следующей строки вывести текущую иерархию дерева.

Пример вывода иерархии дерева объектов:

```
Object tree
rootela
    object_1
        object_7
    object_2
        object_4
            object_7
        object_5
        object_3
    object_3
object_2/object_4     Object name: object_4
Object is set: object_2
//object_7     Object is not found
object_4/object_7     Object name: object_7
.     Object name: object_2
.object_7     Object name: object_7
object_4/object_7     Object name: object_7
.object_7     Redefining the head object failed
Object is set: object_7
//object_1     Dubbing the names of subordinate objects
New head object: object_3
Current object hierarchy tree
rootela
    object_1
        object_7
    object_2
        object_4
        object_5
        object_3
    object_3
        object_7
```

2 МЕТОД РЕШЕНИЯ

3 ОПИСАНИЕ АЛГОРИТМОВ

Согласно этапам разработки, после определения необходимого инструментария в разделе «Метод», составляются подробные описания алгоритмов для методов классов и функций.

3.1 Алгоритм функции main

Функционал: основная функция.

Параметры: нет.

Возвращаемое значение: int.

Алгоритм функции представлен в таблице 1.

Таблица 1 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		создание объекта ob_app класса app с параметром nullptr вызов метода build объекта ob_app возврат результата работы метода start_app() для объекта ob_app	∅

3.2 Алгоритм метода printreadystatus класса baseclass

Функционал: вывод иерархии объектов с указанием состояния готовностикаждого объекта..

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 2.

Таблица 2 – Алгоритм метода printreadystatus класса baseclass

№	Предикат	Действия	№ перехода
1		вывод иерархии объектов с указанием состояния готовностикаждого объекта.	2
2	объект активный	вывод в консоль " is ready"	3
		вывод в консоль " is not ready"	3
3	перебор элементов в children указателем child	вызов у child метода printreadystatus	2
			∅

3.3 Алгоритм метода findobjectbycoordinate класса baseclass

Функционал: поиск объекта в иерархии по координате.

Параметры: нет.

Возвращаемое значение: baseclass*.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода findobjectbycoordinate класса baseclass

№	Предикат	Действия	№ перехода
1		поиск объекта в иерархии по координате	∅

3.4 Алгоритм метода removechild класса baseclass

Функционал: вычекркивание из списка одного объекта из дочерних.

Параметры: нет.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода removechild класса baseclass

№	Предикат	Действия	№ перехода
1		вычекркивание из списка одного объекта из дочерних	∅

3.5 Алгоритм метода changenewparent класса baseclass

Функционал: изменение головного объекта на другой.

Параметры: нет.

Возвращаемое значение: bool.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода changenewparent класса baseclass

№	Предикат	Действия	№ перехода
1		изменение головного объекта на другой	∅

3.6 Алгоритм метода build_tree_objects класса app

Функционал: построение иерархии объектов.

Параметры: нет.

Возвращаемое значение: string.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода build_tree_objects класса app

№	Предикат	Действия	№ перехода
1		построение иерархии объектов	∅

3.7 Алгоритм метода exec_app класса app

Функционал: вывод иерархии объектов в консоль.

Параметры: нет.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода exec_app класса app

№	Предикат	Действия	№ перехода
1		вывод иерархии объектов в консоль	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-5.

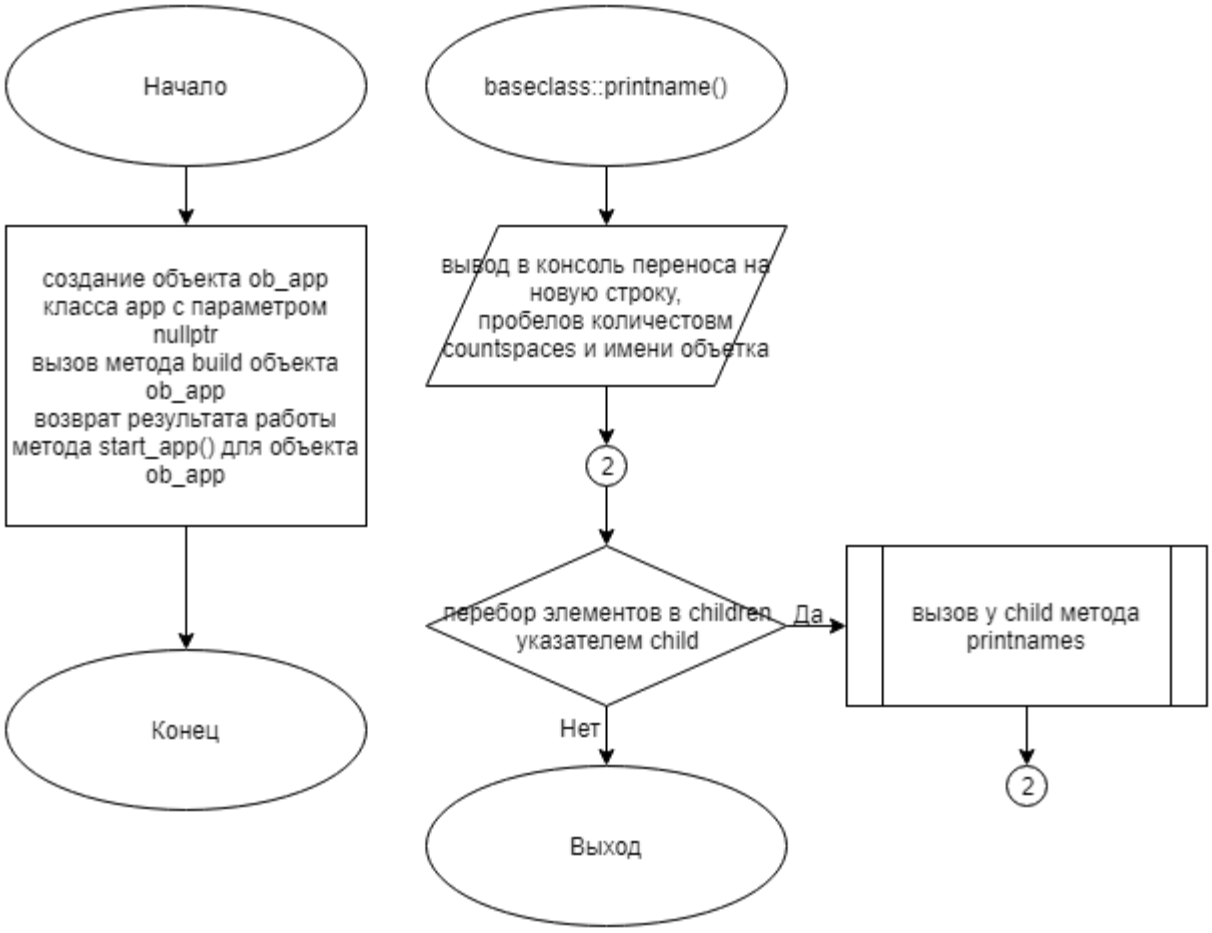

Рисунок 1 – Блок-схема алгоритма

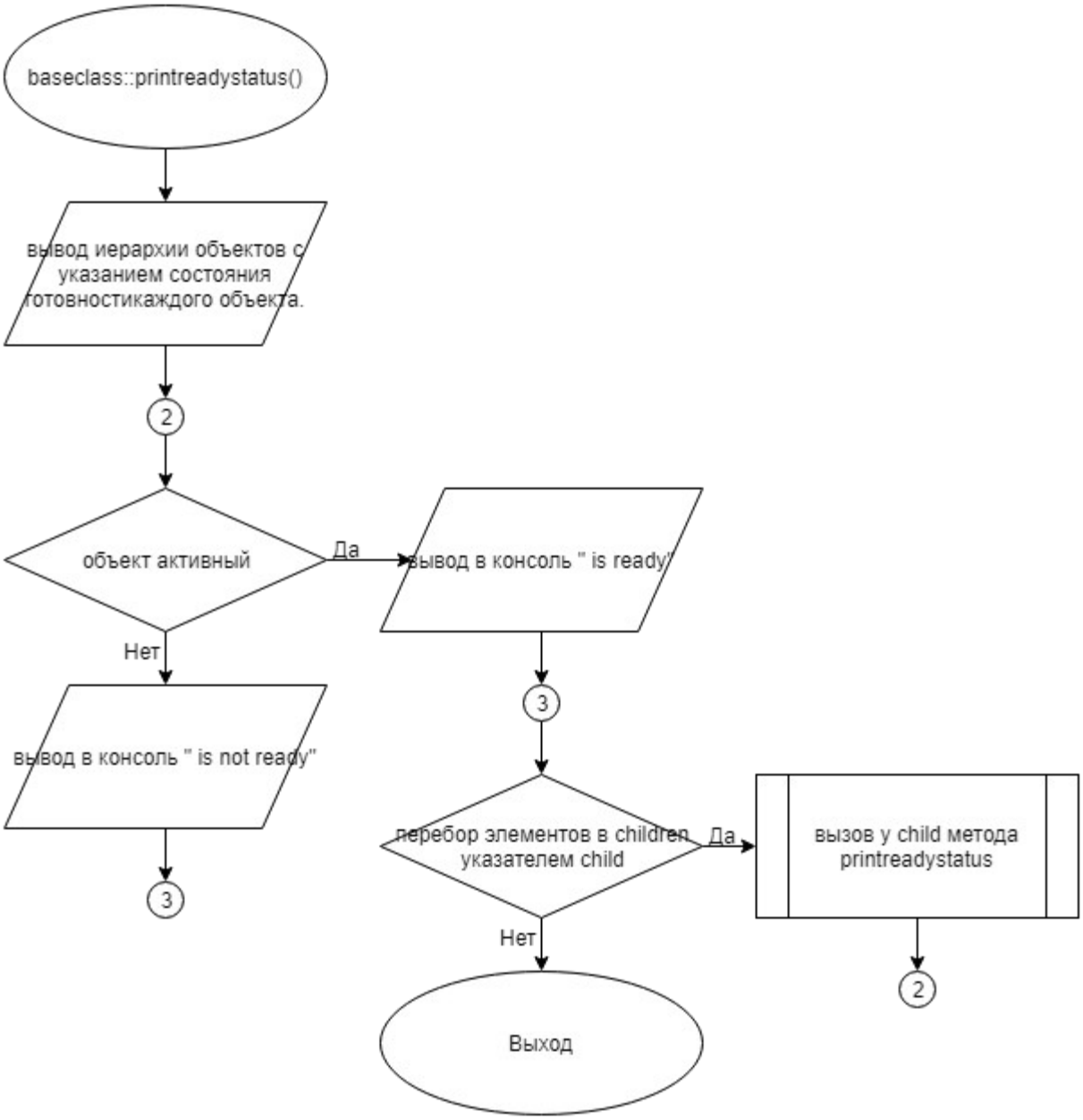

Рисунок 2 – Блок-схема алгоритма

Рисунок 3 – Блок-схема алгоритма

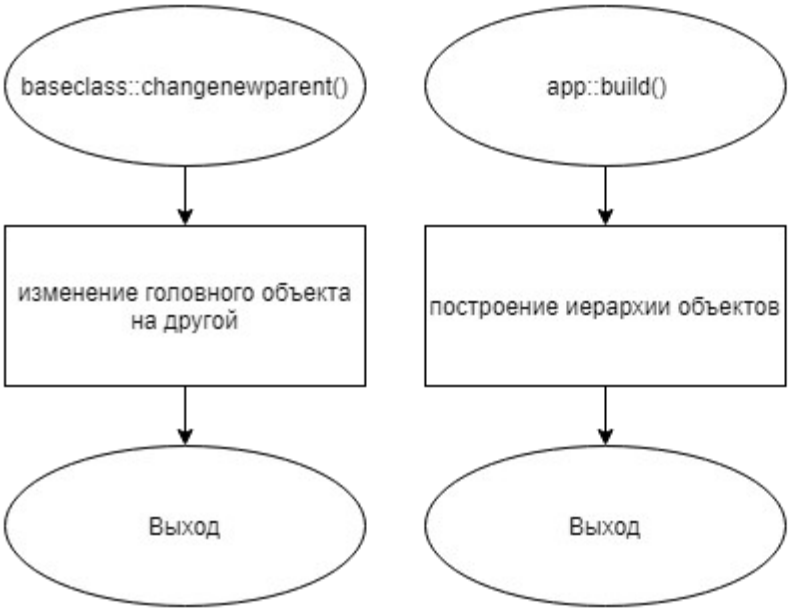

Рисунок 4 – Блок-схема алгоритма

Рисунок 5 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл Class_2.cpp

Листинг 1 – Class_2.cpp

```
#include "Class_2.h"
#include "Class_base.h"
#include <iostream>
using namespace std;
Class_2 :: Class_2(Class_base* b_head, string b_name) :
//-------------------------------------------------------------------------//
// Конструктор класса Class_5 с присвоением свойствам класса Class_5 //
// значения параметров b_head и b_name. Вызов конструктора класса //
// Class_base с аргументами b_head и b_name //
//-------------------------------------------------------------------------//
Class_base(b_head, b_name){}
```

5.2 Файл Class_2.h

Листинг 2 – Class_2.h

```
#ifndef __CLASS_2__H
#define __CLASS_2__H
#include "Class_base.h"
#include <iostream>
using namespace std;
class Class_2 : public Class_base{ // наследование от класса Class_base с
модификатором доступа public
public: // открытый модификатор доступа
   Class_2(Class_base* b_head, string b_name); // конструктор
};
#endif
```

5.3 Файл Class_3.cpp

Листинг 3 – Class_3.cpp

```
#include "Class_3.h"
#include "Class_2.h"
#include <iostream>
using namespace std;
Class_3 :: Class_3(Class_base* b_head, string b_name) :
//-------------------------------------------------------------------//
// Конструктор класса Class_3 с присвоением свойствам класса Class_3 //
// значения параметров b_head и b_name. Вызов конструктора класса //
// Class_base с аргументами b_head и b_name //
//-------------------------------------------------------------------//
Class_base(b_head, b_name){}
```

5.4 Файл Class_3.h

Листинг 4 – Class_3.h

```
#ifndef __CLASS_3__H
#define __CLASS_3__H
#include "Class_2.h"
#include <iostream>
using namespace std;
class Class_3 : public Class_base{ // наследование от класса Class_base с
модификатором доступа public
public: // открытый модификатор доступа
   Class_3(Class_base* b_head, string b_name); // конструктор
};
#endif
```

5.5 Файл Class_4.cpp

Листинг 5 – Class_4.cpp

```
#include "Class_4.h"
#include "Class_base.h"
#include <iostream>
using namespace std;
Class_4 :: Class_4(Class_base* b_head, string b_name) :
//-------------------------------------------------------------------//
// Конструктор класса Class_4 с присвоением свойствам класса Class_4 //
```

```
// значения параметров b_head и b_name. Вызов конструктора класса //
// Class_base с аргументами b_head и b_name //
//-------------------------------------------------------------------//
Class_base(b_head, b_name){}
```

5.6 Файл Class_4.h

Листинг 6 – Class_4.h

```
#ifndef __CLASS_4__H
#define __CLASS_4__H
#include "Class_3.h"
#include <iostream>
using namespace std;
class Class_4 : public Class_base{ // наследование от класса Class_base с
модификатором доступа public
public: // открытый модификатор доступа
   Class_4(Class_base* b_head, string b_name); // конструктор
};
#endif
```

5.7 Файл Class_5.cpp

Листинг 7 – Class_5.cpp

```
#include "Class_5.h"
#include "Class_base.h"
#include <iostream>
using namespace std;
Class_5 :: Class_5(Class_base* b_head, string b_name) :
//-------------------------------------------------------------------//
// Конструктор класса Class_5 с присвоением свойствам класса Class_5 //
// значения параметров b_head и b_name. Вызов конструктора класса //
// Class_base с аргументами b_head и b_name //
//-------------------------------------------------------------------//
Class_base(b_head, b_name){}
```

5.8 Файл Class_5.h

Листинг 8 – Class_5.h

```
#ifndef __CLASS_5__H
#define __CLASS_5__H
#include "Class_4.h"
#include <iostream>
using namespace std;
class Class_5 : public Class_base{ // наследование от класса Class_base с
модификатором доступа public
public: // открытый модификатор доступа
   Class_5(Class_base* b_head, string b_name); // конструктор
};
#endif
```

5.9 Файл Class_6.cpp

Листинг 9 – Class_6.cpp

```
#include "Class_6.h"
#include "Class_base.h"
#include <iostream>
using namespace std;
Class_6 :: Class_6(Class_base* b_head, string b_name) :
//-------------------------------------------------------------------//
// Конструктор класса Class_6 с присвоением свойствам класса Class_6 //
// значения параметров b_head и b_name. Вызов конструктора класса //
// Class_base с аргументами b_head и b_name //
//-------------------------------------------------------------------//
Class_base(b_head, b_name){}
```

5.10 Файл Class_6.h

Листинг 10 – Class_6.h

```
#ifndef __CLASS_6__H
#define __CLASS_6__H
#include "Class_5.h"
#include <iostream>
using namespace std;
class Class_6 : public Class_base{ // наследование от класса Class_base с
модификатором доступа public
```

```
public: // открытый модификатор доступа
   Class_6(Class_base* b_head, string b_name); // конструктор
};
#endif
```

5.11 Файл Class_application.cpp

Листинг 11 – Class_application.cpp

```
#include "Class_application.h"
#include <iostream>
using namespace std;
Class_application :: Class_application(Class_base* p_head) :
//-------------------------------------------------------------------------//
// Конструктор класса Class_application с присвоением свойствам класса //
// Class_application значения параметров b_head и b_name. //
// Вызов конструктора класса Class_base с аргументом b_head //
//-------------------------------------------------------------------------//
Class_base(p_head){}
int flag = 0;
void Class_application :: build_tree_objects(){
   //------------------------------//
   // Метод ввода иерархии объектов //
   //------------------------------//
   string   b_head_name,   b_sub_name,   action;   //   объявление   переменных
строкового типа
   Class_base* current_object = this;
   Class_base* b_head = this;
   int number_class; // объявление целочиссленной переменной - номер класса и
номер состояния
   cin >> b_head_name; // ввод с клавиатуры значения переменной b_head_name
   setName(b_head_name); // вызов метода и установка имени
   while(true){
      cin >> b_head_name; // ввод с клавиатуры именя головного объекта
      if (b_head_name == "endtree") break; // завершить цикл, если головного
объекта равно endtree
         cin >> b_sub_name >> number_class; // ввод с клавиатуры имени ее
подчиненного объекта и номера класса
         b_head = getObjectCoordinate(b_head_name);
      if (!b_head){
         this -> setName('/' + this -> getName() + ' ' + b_head_name);
         break;
      }
      if (b_head){ // если найден
         switch(number_class){ // выбор взависимости от номера класса
            case 2: // номер класса равен двум
               new  Class_2(b_head,  b_sub_name);  //  создание  объекта  класса
Class_2
               break;
            case 3: // номер класса равен трём
               new  Class_3(b_head,  b_sub_name);  //  создание  объекта  класса
```

```
Class_3
                break;
             case 4: // номер класса равен четырем
                new  Class_4(b_head,  b_sub_name);  //  создание  объекта  класса
Class_4
                break;
             case 5: // номер класса равен пяти
                new  Class_5(b_head,  b_sub_name);  //  создание  объекта  класса
Class_5
                break;
             case 6: // номер класса равен шести
                new  Class_6(b_head,  b_sub_name);  //  создание  объекта  класса
Class_6
                break;
          }
       }
    }
    cout << "Object tree" << endl;
    if (this -> getName()[0] == '/'){
       string index = getName().substr(getName().find(' ', 0) + 1);
       this -> setName(getName().substr(1, getName().find(' ', 0) - 1));
       showObjectAndSubObjects();
       cout << endl << "The head object " << index << " is not found";
       flag = 1;
    }
    else{
       showObjectAndSubObjects();
       cout << endl;
       while (true) {
          cin >> action;
          if (action == "END")
             break;
          cin >> b_head_name;
          Class_base*         object         =         current_object         -
>getObjectCoordinate(b_head_name);
          if (action == "SET"){
             if (object){
                current_object = object;
                cout << "Object is set: " << object -> getName() << endl;
             }
             else{
                cout << "The object was not found ar the specified coordinate:
" << b_head_name << endl;
             }
          }
          else if (action == "FIND"){
             if (object)
                cout << b_head_name << "     Object name: " << object ->
                getName() << endl;
             else
                cout << b_head_name << "     Object is not found" << endl;
          }
          else if (action == "MOVE"){
             if (object == nullptr) cout << b_head_name << "      Head object
is not found"<< endl;
                else  if(current_object  ->  changeHead(object))  cout  <<  "New
```

```
head object: " << object -> getName() << endl;
                        else  if  (object  ->  getSubObject(current_object  ->
getName()) != nullptr){
                           cout << b_head_name << "      Dubbing the names of
subordinate objects" << endl;
                        }
            else cout << b_head_name << "       Redefining the head object
failed" << endl;
         }
         else if (action == "DELETE"){
            if (object != nullptr){//если не пустой
               string delete_str;
               Class_base* b_head_temporaty = object -> getHead();
               if    (current_object    ->    search_Object_branch(object    ->
getName())){//есть в подчиненных удолемый объект
                  current_object -> deleteObjectName(object -> getName());
                  while (b_head_temporaty){//объект не пустой
                        if  (b_head_temporaty  ->  getHead())//имеет  головной
объект
                           delete_str = delete_str + "/" + b_head_temporaty ->
getName();
                        b_head_temporaty = b_head_temporaty -> getHead();
                        }
                  cout  <<  "The  object  "  <<  delete_str  +  "/"  +  object->
getName() << " has been deleted" << endl;
               }
            }
         }
      }
   }
}
int Class_application :: exec_app(){
   //--------------------------//
   // Метод запуска приложения //
   //--------------------------//
   if (flag == 1)
      return(0);
   cout << "Current object hierarchy tree" << endl;
   showObjectAndSubObjects();
   return 0;
}
```

5.12 Файл Class_application.h

Листинг 12 – Class_application.h

```
#ifndef __CLASS_APPLICATION__H
#define __CLASS_APPLICATION__H
#include "Class_base.h"
#include "Class_6.h"
```

```
#include <iostream>
using namespace std;
class Class_application : public Class_base{
   // наследование от класса
   // Class_base с модификатором доступа public
public: // открытый модификатор доступа
   Class_application(Class_base* p_head); // конструктор
   void build_tree_objects(); // метод постройки дерева, то есть ввод с
клавиатуры данных
   int exec_app(); // запуск приложения
};
#endif
```

5.13 Файл Class_base.cpp

Листинг 13 – Class_base.cpp

```
#include "Class_base.h"
#include <iostream>
#include <algorithm>
using namespace std;
int count = 0;
Class_base :: Class_base(Class_base* b_head_object, string b_name) :
b_head_object(b_head_object), b_name(b_name) // присвоение свойствам класса
   //Class_base значения из параметра конструктора
   {
   //-----------------------------------------------//
   // Конструктор класса Class_base создающий //
   // ветку подчинённых объектов //
   //-----------------------------------------------//
   if(b_head_object != nullptr) // проверка на нулевой указатель
      b_head_object -> b_sub_objects.push_back(this); // добавляение в конец
   //вектора указателя на объект
   }
Class_base :: ~Class_base(){
   //-----------------------------------------------//
   // Деструктор класса Class_base удаляющий //
   // ветку подчинённых объектов //
   //-----------------------------------------------//
   for(int i = 0; i < b_sub_objects.size(); i++){ // цикл по подчиненным
                                                  //объектам
      delete b_sub_objects[i]; // удаление подчиненного объекта
   }
}
bool Class_base :: setName(string b_new_name){
   //-------------------------------------//
   // Установка нового имени из параметра //
   // если такое имя еще не существует //
   //-------------------------------------//
   if (b_head_object != nullptr){ // проверка на не нулевую ссылку
```

```
      for(int i = 0; i < b_head_object -> b_sub_objects.size(); i++){ // по
   //указателю на головной объект вызываем его вектор с его подчиненными
   //объектами и функцию size
         if   (b_head_object   ->   b_sub_objects[i]   ->   getName()   ==
b_new_name) //проверка на совпадение с именем
           return false; // вовзрат неудачной установки имени
      }
   }
   this -> b_name = b_new_name; // присвоение свойству b_name параметр
b_new_name
   return true; // возврат индикатора успешной установки имени
}
string Class_base :: getName(){
   //--------------------------------//
   // Метод возвращающий имя объекта //
   //--------------------------------//
   return this -> b_name; // возврат имени объекта
}
Class_base* Class_base :: getHead(){
   //--------------------------------//
   // Метод возвращающий указатель //
   // на головной объект //
   //--------------------------------//
   return this -> b_head_object; // возврат указателя на головной объект
}
Class_base* Class_base :: search_Object_branch(string b_name_new){
   //-----------------------------------------------//
   // Метод осуществляет поиск объекта по имени //
   // от корня и возвращает указатель на //
   // первое вхождение объекта с требуемым именем //
   // то есть поиск по дереве //
   //-----------------------------------------------//
   int counter = 0; //счетчик
   Class_base* result = nullptr;
   vector <Class_base*> vecto_r(1,this);
   while(vecto_r.size()>0){
      Class_base* clas = vecto_r.back();
      vecto_r.pop_back();
      if (clas -> b_name == b_name_new){
         counter++;
         result = clas;
      }
      for(int i = 0; i < clas -> b_sub_objects.size(); i++){
         vecto_r.push_back(clas -> b_sub_objects[i]);
         }
      }
   if (counter <= 1)
      return result;
   return nullptr;
}
Class_base* Class_base :: search_object_sub_by_name(string b_name_new){
   //-----------------------------------------------//
   // Метод осуществляет поиск объекта по имени //
   // от текущего и возвращает указатель на //
   // первое вхождение объекта с требуемым именем //
```

```
   // то есть поиск по ветке //
   //-----------------------------------------//
   if (b_head_object != nullptr)
      return b_head_object -> search_object_sub_by_name(b_name_new);
   else
      return search_Object_branch(b_name_new);
}
void Class_base :: setObjectState(int state){
   //--------------------------//
   // Метод установки состояния //
   // объекта //
   //--------------------------//
   if(getHead() && !getHead() -> b_state || state == 0) // если есть головной
                        //айл и он не имеет состояния или его состояние равно
0
      this -> b_state = false; // присвоение состояние not ready
   else // иначе
      this -> b_state = true; // присвоение состояние ready
   if (!state) // если состояние falst то есть notReady
      for (auto b_sub_object : b_sub_objects) // цикл по подчиненным объектам
         b_sub_object -> setObjectState(state); // все подчиненные объекты
получают состояние false то есть not ready
}
void Class_base :: showObjectAndSubObjects(int spaces){
   //--------------------------------//
   // Метод вывода иерархии объектов //
   // (дерева или ветки) от текущего //
   // объекта //
   //--------------------------------//
   cout << this -> getName(); // вывод текущего имени объекта
   if (!b_sub_objects.empty()){ // если массив с подчиненными объектами не
пустой
      for ( auto b_sub_object : b_sub_objects){ // цикл по подчиненным
объектам
         cout << endl; // переход на новую строку
         for(int i = 0; i < spaces; i++) // цикл пробелов
            cout << " "; // вывод пробела на экран
         b_sub_object -> showObjectAndSubObjects(spaces + 4); // вызов метода
   //showObjectAndSubObjects с параметром пробелов + 4 по ссылке
      }
   }
}
void Class_base :: showObjectsState(int spaces){
   //-------------------------------------------------//
   // Метод вывода иерархии объектов (дерева или ветки) //
   // и отметок их готовности от текущего объекта //
   //-------------------------------------------------//
   cout << this -> getName(); // вывод текущего имени объекта
   cout << (this -> b_state ? " is ready" : " is not ready"); // если
   //состояние объекта true - вывод is ready, если false - вывод is not ready
   if (!b_sub_objects.empty()){ // если массив с подчиненными объектами не
пустой
      for ( auto b_sub_object : b_sub_objects){ // цикл по подчиненным
объектам
         cout << endl; // переход на новую строку
```

```
         for(int i = 0; i < spaces; i++) // цикл попробелам
            cout << " "; // вывод пробела на экран
      b_sub_object -> showObjectsState(spaces + 4); // вызов метода
         //showObjectsState с параметром пробелов + 4 по ссылке
      }
   }
}
bool Class_base :: changeHead(Class_base* new_b_head_object)
{
   //-------------------------------------------------------------------------/
/
   // Метод переопределения головного объекта для текущего в дереве иерархии.
   //
   // Метод должен иметь один параметр, указатель на объект базового класса,
   //
   //  содержащий  указатель  на  новый  головной  объект.  Переопределение
головного //
   // объект для корневого объекта недопустимо. Недопустимо создать второй
   //
   // корневой объект. Недопустимо при переопределении, чтобы у нового
   //
   // головного появились два подчиненных объекта с одинаковым наименованием.
   //  Новый  головной  объект  не  должен  принадлежать  к  объектам  из  ветки
текущего. //
   // Если переопределение выполнено, метод возвращает значение «истина»,
   // иначе «ложь»
   //
   //-------------------------------------------------------------------------
   if(new_b_head_object){
      Class_base* test_object = new_b_head_object;
      while (test_object)
      {
         test_object = test_object -> getHead();
         if (this == test_object)
            return false;
      }
         if (!new_b_head_object -> getSubObject(getName()) && b_head_object)
         {
            b_head_object    ->    b_sub_objects.erase(find(b_head_object    -
>b_sub_objects.begin(), b_head_object->b_sub_objects.end(), this));
            new_b_head_object -> b_sub_objects.push_back(this);
            b_head_object = new_b_head_object;
            return true;
         }
      }
   return false;
   }
void Class_base :: deleteObjectName(string delete_b_name){
   //-----------------------------------------------------//
   // Метод удаления подчиненного объекта по наименованию. //
   // Если объект не найден, то метод завершает работу. //
   // Один параметр строкового типа, содержит наименование //
   // удаляемого подчиненного объекта //
   //-----------------------------------------------------//
   if (delete_b_name.empty()) // проверка на не пустую строчку
```

```
      return; // завершение метода
   for( int i = 0; i < b_sub_objects.size(); i++) // цикл по подчиненным
объектам
      if (b_sub_objects[i] -> getName() == delete_b_name) // проверка что
объект по индексу совпадает с удаляемым объектом
         b_sub_objects.erase(b_sub_objects.begin()   +   i);   //   удаление
подчиненных объектов объекта delete_b_name
}
Class_base* Class_base :: getObjectCoordinate(string b_name_coordinate)
{
   //-------------------------------------------------------------------------
//
      // Метод получения указателя на любой объект в составе дерева иерархии
      //объектов согласно пути (координаты).
      //
      // В качестве параметра методу передать путь (координату) объекта.
      //Координата задаться в следующем виде:
   //
   // / - корневой объект;
   //
   // //«имя объекта» - поиск объекта по уникальной имени от корневого (для
   //однозначности уникальность требуется в рамках дерева);
   //
   // . - текущий объект;

      // .«имя объекта» - поиск объекта по уникальной имени от текущего (для
   // однозначности уникальность требуется в рамках ветви дерева от текущего
      //объекта); //
   // «имя объекта 1»[/«имя объекта 2»] . . . - относительная координата от
   //текущего объекта, «имя объекта 1» подчиненный текущего;
   //
   // /«имя объекта 1»[/«имя объекта 2»] . . . - абсолютная координата от
   //корневого объекта.
      //
      //-------------------------------------------------------------------------
---

   string b_name_this;
   int index = 0;
   Class_base* b_root = this;
   if (b_name_coordinate[0] == '/')
   {
      while (b_root -> getHead())
         b_root = b_root -> getHead();
   }
   //метод получения на любой объект в сосаве дерева иерархии объектов
согласно пути
   if (b_name_coordinate.length() == 1 && (b_name_coordinate[0] == '/' ||
                                    b_name_coordinate[0] == '.'))
      return b_root;
   if (b_name_coordinate[0] == '/' || b_name_coordinate[0] == '.')
      index = 1;
   if (b_name_coordinate[0] == '/' && b_name_coordinate[1] == '/')
   {
      b_name_this = b_name_coordinate.substr(2);
```

```
            return b_root->search_Object_branch( b_name_this);
        }
        while (index < b_name_coordinate.length())
        {
            if (b_name_coordinate[index] == '/')
            {
                b_root = b_root -> getSubObject(b_name_this);
                if (b_root == nullptr)
                    return nullptr;
                b_name_this = "";
            }
            else
                b_name_this += b_name_coordinate[index];
            index++;
        }
        if (b_name_coordinate[0] == '.')
            return b_root->search_Object_branch(b_name_this);
        else
            return b_root->getSubObject(b_name_this);
    }
    Class_base* Class_base :: getSubObject(string s_name){
        for(int i = 0; i < b_sub_objects.size(); i++)
            if (b_sub_objects[i] -> getName() == s_name)
                return b_sub_objects[i];
        return nullptr;
    }
```

5.14 Файл Class_base.h

Листинг 14 – Class_base.h

```
    #ifndef __CLASS_BASE__H
    #define __CLASS_BASE__H
    #include <iostream>
    #include <vector>
    using namespace std;
    // базовый класс
    class Class_base{
    private:
        string b_name;
        Class_base* b_head_object; // указатель на головной объект для каждого
    объекта
        vector <Class_base*> b_sub_objects; // динамический массив хранящий
    подчиненные объекты головного объекта
        int b_state = 0; // состояние объекта
    public:
        Class_base(Class_base* b_head_object, string b_name = "Base Object"); //
    конструктор
        ~Class_base(); // деструктор
        bool setName(string b_new_name); // установка имени из аргумента
```

```
   string getName(); // возврат свойства имя
   Class_base* getHead(); // возврат указателя на головной объекта
   Class_base* search_object_sub_by_name(string b_name);
   Class_base*  search_Object_branch(string  b_name);  //  поиск  объекта  с
заданным именем на дереве
   void setObjectState(int state); // установка готовности объекта, в
   //        качестве параметра передается переменная целого типа, содержит
номер
   //    состояния.
   void showObjectAndSubObjects(int = 4);  // метод вывода иерархии объектов
(дерева или ветки) от текущего объекта с параметром пробелов заданным по
умолчанию
   void showObjectsState(int = 4); // метод вывода иерархии объектов (дерева
или ветки) и отметок их готовности от текущего объекта
   Class_base* getSubObject(string s_name);
   bool changeHead(Class_base* new_b_head_object);
            /* метод
            переопределения головного объекта для текущего в дереве иерархии.
Метод
            должен иметь один параметр, указатель на объект базового класса,
содержащий
            указатель на новый головной объект. Переопределение головного
объект для
            корневого  объекта  недопустимо.  Недопустимо  создать  второй
корневой объект.
            Недопустимо  при  переопределении,  чтобы  у  нового  головного
появились два
            подчиненных объекта с одинаковым наименованием. Новый головной
объект не
            должен  принадлежать  к  объектам  из  ветки  текущего.  Если
переопределение
            выполнено, метод возвращает значение «истина», иначе «ложь»*/
   void deleteObjectName(string delete_b_name); /* метод удаления
            подчиненного объекта по наименованию. Если объект не найден, то
метод
            завершает  работу.  Один  параметр  строкового  типа,  содержит
наименование
            удаляемого подчиненного объекта*/
   Class_base* getObjectCoordinate(string b_name_coordinate); // метод
   //получения указателя на любой объект в составе дерева иерархии объектов
   //согласно пути (координаты)
};
#endif
```

5.15 Файл main.cpp

Листинг 15 – main.cpp

```
#include <stdlib.h>
#include <stdio.h>
```

```
#include "Class_application.h"
#include <iostream>
using namespace std;
int main()
{
   Class_application ob_cl_application ( nullptr ); // создание объекта
приложение
   ob_cl_application.build_tree_objects (); // конструирование системы,
   return ob_cl_application.exec_app (); // запуск системы
}
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 8.

Таблица 8 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные

rootela
/ object_1 3
/ object_2 2
/object_2 object_4 3
/object_2 object_5 4
/ object_3 3
/object_2 object_3 6
/object_1 object_7 5
/object_2/object_4
object_7 3
endtree
FIND
object_2/object_4
SET /object_2
FIND //object_7
FIND
object_4/object_7
FIND .
FIND .object_7
FIND
object_4/object_7
MOVE .object_7
SET
object_4/object_7
MOVE //object_1
MOVE /object_3
END
	Object tree
	rootela
	object_1
	object_7
	object_2
	object_4
	object_7
	object_5
	object_3
	object_3
	object_2/object_4
	Object name:
	object_4
	Object is set:
	object_2
	//object_7
	Object is not found
	object_4/object_7
	Object name:
	object_7
	. Object name:
	object_2
	.object_7 Object
	name: object_7
	object_4/object_7
	Object name:
	object_7
	.object_7
	Redefining the head
	object failed
	Object is set:
	object_7
	//object_1
	Dubbing the names of
	subordinate objects
	New head object:
	object_3
	Current object
	hierarchy tree
	rootela
	object_1
	object_7
	object_2
	object_4
	object_5
		Object tree
		rootela
		object_1
		object_7
		object_2
		object_4
		object_7
		object_5
		object_3
		object_3
		object_2/object_4
		Object name:
		object_4
		Object is set:
		object_2
		//object_7
		Object is not found
		object_4/object_7
		Object name:
		object_7
		. Object name:
		object_2
		.object_7 Object
		name: object_7
		object_4/object_7
		Object name:
		object_7
		.object_7
		Redefining the head
		object failed
		Object is set:
		object_7
		//object_1
		Dubbing the names of
		subordinate objects
		New head object:
		object_3
		Current object
		hierarchy tree
		rootela
		object_1
		object_7
		object_2
		object_4
		object_5

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
	object_3 object_3 object_7	object_3 object_3 object_7

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 19 Единая система программной документации.

2. Методическое пособие студента для выполнения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [Электронный ресурс] – URL: https://mirea.aco-avrora.ru/student/files/methodichescoe_posobie_dlya_laboratornyh_rabot_3.pdf (дата обращения 05.05.2021).

3. Приложение к методическому пособию студента по выполнению заданий в рамках курса «Объектно-ориентированное программирование» [Электронный ресурс]. URL: https://mirea.aco-avrora.ru/student/files/Prilozheniye_k_methodichke.pdf (дата обращения 05.05.2021).

4. Шилдт Г. C++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2019. — 624 с.

5. Видео лекции по курсу «Объектно-ориентированное программирование» [Электронный ресурс]. АСО «Аврора».

6. Антик М.И. Дискретная математика [Электронный ресурс]: Учебное пособие /Антик М.И., Казанцева Л.В. — М.: МИРЭА — Российский технологический университет, 2018 — 1 электрон. опт. диск (CD-ROM).