# Постановка задачи

**Проверка готовности объектов к работе**

Фрагмент методического указания.

Создание объектов и построение исходного иерархического дерева объектов. Система собирается из объектов, принадлежащих определенным классам. В тексте постановки задачи классу соответствует уникальный номер. Относительно номера класса определяются требования (свойства, функциональность). Первоначальная сборка системы (дерева иерархии объектов, программы) осуществляется исходя из входных данных. Данные вводятся построчно.

Первая строка содержит имя корневого объекта (объект приложение). Номер класса корневого объекта 1. Корневой объект объявляется в основной программе (main). Исходное состояние корневого объекта соответствует его функционированию. Далее, каждая строка входных данных определяет очередной объект, задает его характеристики и расположение на дереве иерархии.

Структура данных в строке:

«Наименование головного объекта»«Наименование очередного объекта»«Номер класса принадлежности очередного объекта»«Номер исходного состояния очередного объекта»

Ввод иерархического дерева завершается, если наименование головного объекта равно « endtree » (в данной строке ввода больше ничего не указывается).

Готовность объекта характеризуется значением его состояния. Значение состояния - целое число.

Определены правила для значения состояния: 0 – объект выключен; Отрицательное – объект включен, но не функционирует, обнаружена неисправность.

Значение классифицирует характер неисправности. Положительное – объект включен, функционирует в штатном режиме. Значение определяет текущее состояние объекта.

Подчиненные объекты располагаются слева на право относительно головного, согласно их следованию в исходных данных. Исходные данные подготовлены таким образом, что любой головной объект предварительно добавлен в качестве подчиненного.

Подразумевается, что все объекты имеют уникальные имена. Для организации исходя из входных данных создания экземпляров объектов и формирования иерархического дерева, необходимо: 1. В базовом классе реализовать метод поиска объекта на дереве объектов по его наименованию и возврата указателя на него. Если объект не найден, то вернуть нулевой указатель.

2. В корневом объекте (объект приложения) реализовать метод чтения исходных данных, создания объектов и построения исходного дерева иерархии.

Пример Ввод

app\_root

app\_root object\_1 3 1

app\_root object\_2 2 1

object\_2 object\_4 3 -1

object\_2 object\_5 3 1

app\_root object\_3 3 1

object\_2 object\_6 2 1

object\_1 object\_7 2 1

endtree

Построенное дерево

app\_root

object\_1

object\_7

object\_2

object\_4 object\_5

object\_6

object\_3

Вывод списка готовности объектов

The object app\_root is ready

The object object\_1 is ready

The object object\_7 is ready

The object object\_2 is ready

The object object\_4 is not ready

The object object\_5 is ready

The object object\_6 is ready

The object object\_3 is ready

**Постановка задачи** Все сложные электронные, технические средства разного назначения в момент включения выполняют опрос готовности к работе составных элементов, индицируя соответствующую информацию на табло, панели или иным образом.

Построить модель иерархической системы. Реализовать задачу опроса готовности каждого объекта из ее состава и вывести соответствующее сообщение на консоль.

Объект считается готовым к работе: 1. Создан и размешен в составе системы (на дереве иерархии объектов) согласно схеме архитектуры;

1. Имеет свое уникальное наименование;
2. Свойство, определяющее его готовность к работе, имеет целочисленное положительное значение.

В результате решения задачи опроса готовности объектов, относительно каждого объекта системы на консоль надо вывести соответствующую информацию:

Если свойство определяющее готовность объекта имеет положительное значение:

The object «наименование объекта» is ready иначе

The object « наименованиеобъекта » is not ready

Система содержит объекты трех классов, не считая корневого. Номера классов: 2,3,4.

## Описание входных данных

Множество объектов, их характеристики и расположение на дереве иерархии.

Структура данных для ввода согласно изложенному в фрагменте методического указания.

## Описание выходных данных

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **В** |  | **первой** |  |  | **строке** | вывести |
| Test |  |  |  |  |  | result |

**Далее, построчно**, согласно следованию объектов на дереве иерархии слева на право и сверху вниз, относительно каждого объекта в зависимости от состояния готовности выводиться, если объект готов к работе:

The object «наименование объекта» is ready

Если не готов, то:

The object «наименование объекта» is not ready

# Метод решения

Класс С, наследуемый от Base с модификатором public

Изменения в классе Base: добавлено закрытое свойство status - состояние текущей вершины изменен конкструктор класса Base(Base \* parent, string name, int status) добавлен метод print\_status\_tree - выводит статусы вершины поддерева текущей вершины

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Имя класса | Класс-наследник | Модификатор доступа при наследовании | Описание |
| 1 | Base |  |  | Основа для в |
| App | public |  |
| A | public |  |
| B | public |  |
| C | public |  |
| 2 | App |  |  | Класс прило |
| 3 | A |  |  |  |
| 4 | B |  |  |  |
| 5 | C |  |  |  |

Изменения в классе App: изменена функция build\_tree() измененеа функция run()

# Описание алгоритма

Класс объекта: Base

Модификатор доступа: public

Метод: Base

Функционал: Конструктор класса

Параметры: Base \* parent - указатель на родителя, string name - имя вершины, int status - её статус

Возвращаемое значение: отсутствует

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** | **Комментарий** |
| 1 |  | Присваивание закрытому свойству parent указателя аргумента parent | 2 |  |
| 2 |  | Присваивание закрытому свойству name значения аргумента name | 3 |  |
| 3 |  | Присваивание закрытому свойству status значениея аргумента status | ∅ |  |

Класс объекта: Base

Модификатор доступа: public

Метод: print\_status\_tree

Функционал: Выводит статусы вершин поддерева из данной вершины

Параметры: отсутствуют

Возвращаемое значение: отсутствует

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** | **Комментарий** |
| 1 |  | Вывод "The object " и значения закрытого свойства name | 2 |  |
| 2 | Значение закрытого свойства status больше 0 | Вывод " is ready" | 3 |  |
| Значение закрытого свойства status меньше 1 | Вывод " is not ready" | 3 |  |
| 3 | Результат работы метода size объекта закрытого  свойства children не равен нулю | Вывод "\n" | 4 |  |
| Результат работы метода size объекта закрытого  свойства children равен нулю |  | 4 |  |
| 4 |  | Инициализация переменной i со значением 0 | 5 |  |
| 5 | i меньше размера объекта закрытого свойства | Вызов метода print\_status\_tree у i-ого элемента закрытого | 6 |  |
|  | children | свойства children |  |  |
| i больше либо равно размеру объекта закрытого свойства children |  | 7 |  |
| 6 | i не равно размеру закрытого свойства children минус 1 | Вывод "\n" | 7 |  |
| i равно размеру закрытого свойства children минус 1 |  | 7 |  |
| 7 |  | Увеличить i на единицу | 5 |  |

Класс объекта: App

Модификатор доступа: public

Метод: build\_tree

Функционал: Строит дерево по данным из входящго потока

Параметры: отсутствуют

Возвращаемое значение: отсутствует

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** | **Комментарий** |
| 1 |  | Инициализация двух переменных строкого типа s1, s2 | 2 |  |
| 2 |  | Инициализация двух переменных целочисленного типа status и class\_type | 3 |  |
| 3 |  | Считывание s1 | 4 |  |
| 4 |  | Вызов метода set\_name у данного объекта с аргументом s1 | 5 |  |
| 5 | Всегда | Считывание s1 | 6 |  |
| Никогда |  | ∅ |  |
| 6 | s1 равно "endtree" |  | ∅ |  |
| s1 не равно "endtree" |  | 7 |  |
| 7 |  | Считывание s2, class\_type, status | 8 |  |
| 8 |  | Вызов метода get\_by\_name у данного объекта с аргументом s1 | 9 |  |
| 9 |  | Присваивание результата работы get\_by\_name переменной b | 10 |  |
| 10 | class\_type равно 2 | Инициализация объекта класса A с аргументами b, s2, status и вызов метода add\_child у объекта b с аргументом созданным объектом класса A | 5 |  |
|  | class\_type равно 3 | Инициализация объекта класса B с аргументами b, s2, status и вызов метода add\_child у объекта b с аргументом созданным объектом класса B | 5 |  |
| class\_type равно 4 | Инициализация объекта класса C с аргументами b, s2, status и вызов метода add\_child у объекта b с аргументом созданным объектом класса C | 5 |  |
| class\_type не равно 2, 3, 4 |  | 5 |  |

Класс объекта: App

Модификатор доступа: public

Метод: run

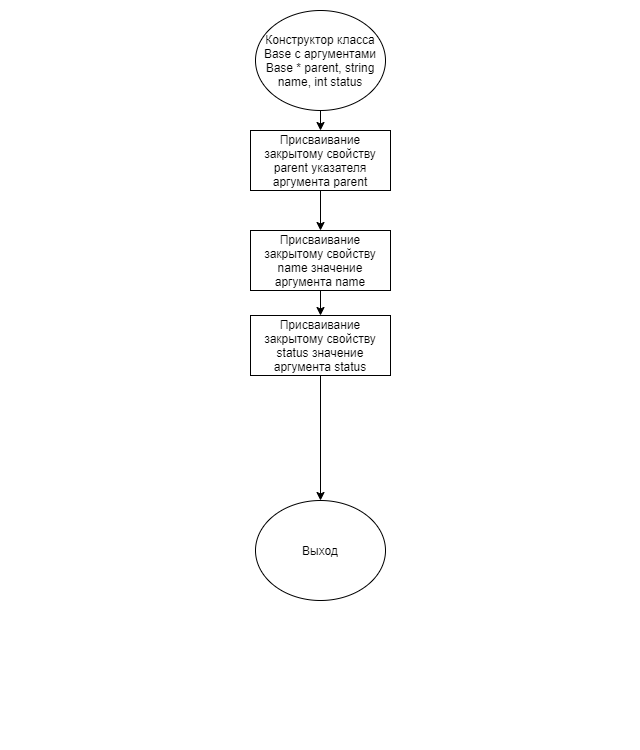
Функционал: Запускает программу действий с созданным деревом

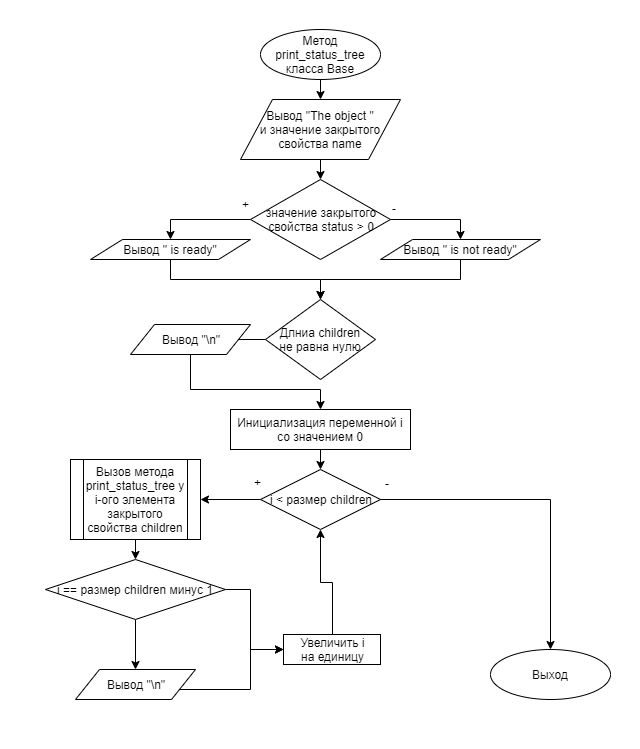
Параметры: отсутствуют

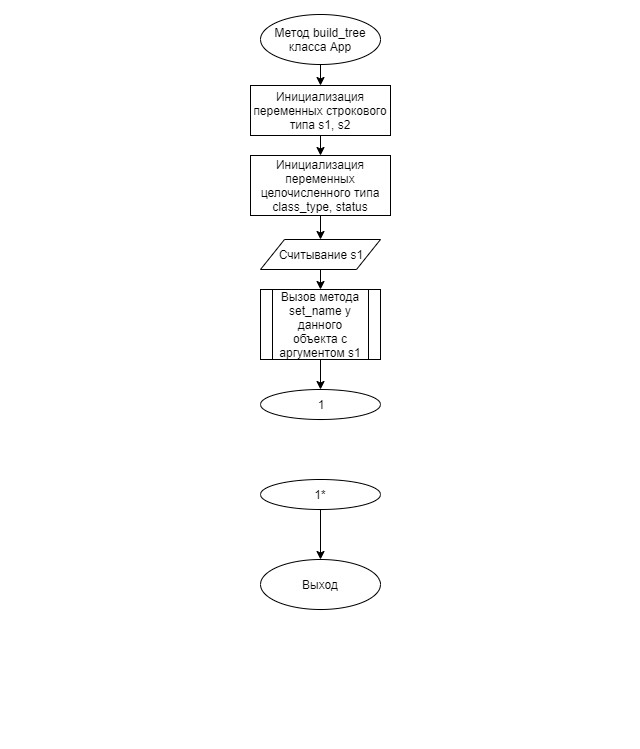
Возвращаемое значение: отсутствует

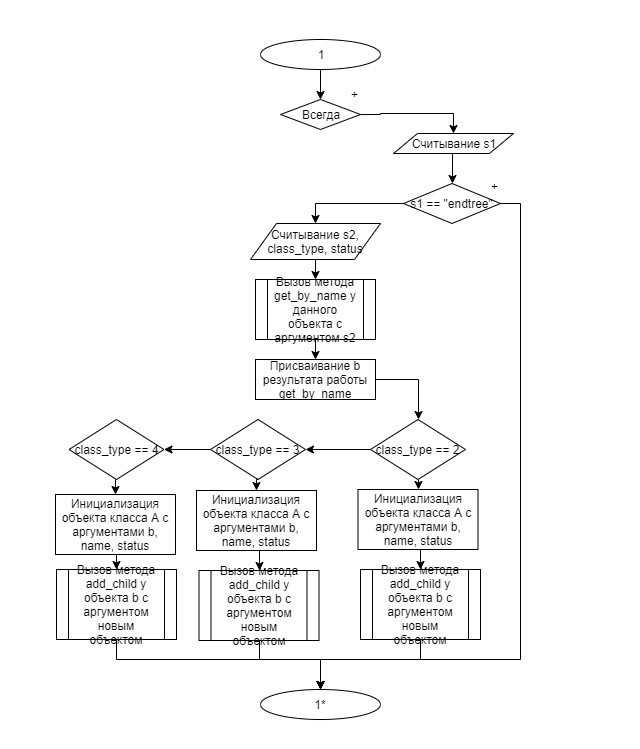
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Предикат** | **Действия** | **№ перехода** | **Комментарий** |
| 1 |  | Вывод "Test result" и перенос строки | 2 |  |
| 2 |  | Вызов метода print\_status\_tree у данного объекта | ∅ |  |

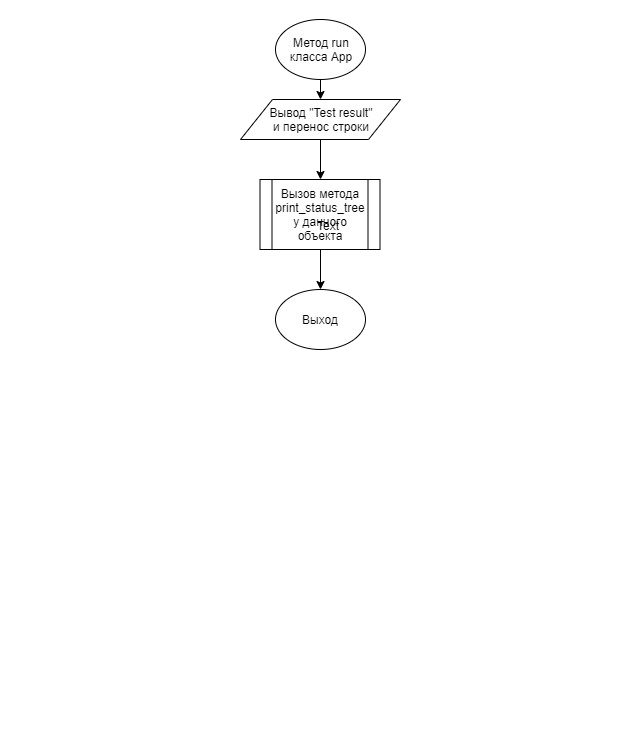
# Блок-схема алгоритма











# Код программы

**Файл a.cpp**

#include <string>

#include "a.h"

#include "base.h"

A::A(Base \* parent, std::string name, int status): Base(parent, name, status)

{};

## Файл a.h

#ifndef AH

#define AH

#include <string>

#include "base.h"

class A: public Base{ public:

A(Base \* parent, std::string name, int status);

};

#endif

## Файл app.cpp

#include <iostream>

#include <string>

#include "app.h"

#include "a.h"

#include "b.h" #include "c.h" using namespace std; App::App(Base \* parent): Base(parent, "", 1){};

void App::build\_tree(){ string s1, s2;

int class\_type = 0, status;

// Считываем имя корня cin >> s1;

// Задаем имя корня

this->set\_name(s1);

// Начинаем считывание while (true){

// Считывание двух строк, разделенных пробелом

cin >> s1;

// Проверяем, что не конец ввода if (s1 == "endtree"){ break;

}

cin >> s2 >> class\_type >> status;

// Находим элемент с именем s1

Base \* b = this->get\_by\_name(s1);

// Добавляем к найденой вершине элемент с именем s2 if (class\_type == 2){

b->add\_child(new A(b, s2, status));

}else if (class\_type == 3){

b->add\_child(new B(b, s2, status));

}else if (class\_type == 4){

b->add\_child(new C(b, s2, status));

}

}

}

void App::run(){ cout << "Test result\n"; this->print\_status\_tree(); }

## Файл app.h

#ifndef APPH

#define APPH #include "base.h"

class App : public Base{ public:

App(Base \* parent); void build\_tree(); void run(); };

#endif

## Файл base.cpp

#include "base.h"

#include <iostream>

#include <vector> #include <string> using namespace std;

void Base::set\_name(string new\_name){ this->name = new\_name;

return;

}

string Base::get\_name(){ return this->name;

}

void Base::set\_parent(Base \* new\_parent){ this->parent = new\_parent;

return; }

Base::Base(Base \* parent, string name, int status=0){

this->parent = parent; this->name = name; this->status = status;

}

Base \* Base::get\_parent(){ return this->parent;

}

void Base::print\_tree(){ if (!this->children.size()) return;

// Выводим имя данной вершины cout << this->name;

// Выводим имена детей, раздлеяя их двумя пробелами for (int i = 0; i < this->children.size(); ++i){ cout << " ";

cout << this->children[i]->get\_name();

}

// Вызываем аналогичную функцию у детей

bool w = false;

for (int i = 0; i < this->children.size(); ++i){ if (this->children[i]->children.size() && !w){ cout << "\n"; w = true;

}

this->children[i]->print\_tree();

} return;

}

void Base::print\_status\_tree(){

cout << "The object " << this->name;

// Проверяем, готов ли объект, и выводим соответствующую информацию if (this->status > 0){ cout << " is ready";

}else{

cout << " is not ready";

}

if (this->children.size()){ cout << "\n";

}

// Выводим имена детей, раздлеяя их двумя пробелами for (int i = 0; i < this->children.size(); ++i){ this->children[i]->print\_status\_tree(); if (i != this->children.size() - 1) cout << "\n";

} return; }

void Base::add\_child(Base \* h){ this->children.push\_back(h); return;

}

Base \* Base::get\_by\_name(string name){ if (this->name == name){ return this;

}

for (int i = 0; i < this->children.size(); ++i){

Base \* b = children[i]->get\_by\_name(name); if (b != NULL) return b;

}

return nullptr; }

## Файл base.h

#ifndef BASEH

#define BASEH

#include <vector> #include <string> class Base;

class Base{ private: std::string name; Base \* parent; int status; protected:

std::vector<Base \*> children; public:

Base(Base \* parent, std::string name, int status);

void set\_name(std::string new\_name); std::string get\_name();

void set\_parent(Base \* new\_parent);

Base \* get\_parent();

void print\_tree(); void print\_status\_tree();

void add\_child(Base \* h);

Base \* get\_by\_name(std::string name);

};

#endif

## Файл b.cpp

#include <string>

#include "b.h"

#include "base.h"

B::B(Base \* parent, std::string name, int status): Base(parent, name, status)

{};

## Файл b.h

#ifndef BH

#define BH

#include <string>

#include "base.h"

class B: public Base{ public:

B(Base \* parent, std::string name, int status);

};

#endif

## Файл c.cpp

#include <string>

#include "c.h"

#include "base.h"

C::C(Base \* parent, std::string name, int status): Base(parent, name, status)

{};

## Файл c.h

#ifndef CH

#define CH

#include <string>

#include "base.h"

class C: public Base{ public:

C(Base \* parent, std::string name, int status);

};

#endif

**Файл main.cpp**

#include "app.h"

int main(){

App app(NULL); app.build\_tree(); app.run(); }

# Тестирование

|  |  |  |
| --- | --- | --- |
| Входные данные | Ожидаемые выходные данные | Фактические выходные данные |
| root root abc 2 -1 root abd 3 1 abd abh 4 1 abh abg 2 -1 endtree | Test result The object root is ready The object abc is not ready The object abd is ready The object abh is ready The object abg is not ready | Test result The object root is ready The object abc is not ready The object abd is ready The object abh is ready The object abg is not ready |
| app\_root app\_root object\_1 3 1 app\_root object\_2 2 1 object\_2 object\_4 3 -1 object\_2 object\_5  3 1 app\_root object\_3 3 1 | Test result The object app\_root is ready The object object\_1 is ready The object object\_7 is ready The object object\_2 is | Test result The object app\_root is ready The object object\_1 is ready The object object\_7 is ready The object object\_2 is |

|  |  |  |
| --- | --- | --- |
| object\_2 object\_6 2 1 object\_1 object\_7 2 1 endtree | ready The object object\_4 is not ready The object object\_5 is ready The object object\_6 is ready The object object\_3 is ready | ready The object object\_4 is not ready The object object\_5 is ready The object object\_6 is ready The object object\_3 is ready |