

# Постановка задачи

## Вывод иерархического дерева объектов на консоль

Внутренняя архитектура (вид иерархического дерева объектов) в большинстве реализованных программах динамически меняется в процессе отработки алгоритма. Вывод текущего дерева объектов является важной задачей, существенно помогая разработчику, особенно на этапе тестирования и отладки программы.

Построить модель иерархической системы. Реализовать вывод на консоль иерархического дерева объектов в следующем виде:

```
root
    ob_1
    ob_2
    ob_3
    ob_4
    ob
    _5
    ob
    _6
    ob
    _7
```

где: root - наименование корневого объекта (приложения).

Состав и иерархия объектов строиться посредством ввода исходных данных. Ввод организован как в контрольной работе № 1.

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

## Описание входных данных

Множество объектов, их характеристики и расположение на дереве иерархии. Структура данных для ввода согласно изложенному в фрагменте методического указания в контрольной работе № 1.

## Описание выходных данных

Вывести Object	иерархию	объектов	в	следующем	виде: tree
«Наименование		корневого		объекта»	
«Наименование		объекта		1»	
«Наименование		объекта		2»	
«Наименование		объекта		3»	
.	.	.	.	.	.

Отступ каждого уровня иерархии 4 позиции.

## Метод решения

### Класс Base

методы:

- void setName() - устанавливает имя
- модификатор доступа public

- `string getName()` - возвращает имя  
модификатор доступа `public`
- `void setParent()` \- устанавливает родителя  
модификатор доступа `public`
- `Base* getParent()` - возвращает указатель на родителя  
модификатор доступа `public`
- `void setState` - устанавливает статус  
модификатор доступа `public`
- `int getState` - возвращает статус  
модификатор доступа `public`
- `PrintState()` - выводит информацию на экран  
модификатор доступа `public`

Поля:

- `string name`  
имя объекта  
модификатор доступа `private`
- `int state`  
статус объекта  
модификатор доступа `private`
- `base* parent`  
указатель на родителя  
модификатор доступа `private`
- `vector <base*> children`  
список с объектами  
модификатор доступа `private`

## **Класс cl\_application наследуется от Base**

методы :

- конструктор
- void bildtree() - строит дерево объектов
- int execute() - функция вызывает функцию вывода всей информации

Поля:

- Base\* temp\_parent  
указатель на родителя  
модификатор доступа private
- Base\* temp\_child  
указатель на ребенка  
модификатор доступа private
- Base\* root\_parent  
указатель на корневой объект  
модификатор доступа private

## **Класс cl\_1 наследуется от Base**

методы:

- конструктор

## **Класс cl\_2 наследуется от Base**

методы:

- конструктор

## **Класс cl\_3 наследуется от Base**

методы:

- конструктор

## Код программы

### Файл cl\_1.cpp

```
#include "cl_1.h"

#include <string>

using namespace std;

Derived::Derived(Base* parent,int state, string name) :
Base(parent,state,name) {}
```

### Файл cl\_1.h

```
#ifndef __CL_1_H
#define __CL_1_H
#include "cl_base.h"

class Derived : public Base {
public:
    Derived(Base*,int, string);
};

#endif
```

### Файл cl\_2.cpp

```
#include "cl_2.h"

#include <string>

using namespace std;

Derived1::Derived1(Base* parent,int state, string name) :
Base(parent,state,name) {}
```

### Файл cl\_2.h

```

#ifndef __CL_2_H
#define __CL_2_H
#include "cl_base.h"

class Derived1 : public Base {

public:

    Derived1(Base*,int, string);

};

#endif

```

### Файл cl\_3.cpp

```

#include "cl_3.h"

#include <string>

using namespace std;

Derived2::Derived2(Base* parent,int state, string name) :
Base(parent,state,name) {}

```

### Файл cl\_3.h

```

#ifndef __CL_3_H
#define __CL_3_H
#include "cl_base.h"

class Derived2 : public Base {

public:

    Derived2(Base*,int, string);

};

#endif

```

### Файл cl\_4.cpp

```
#include "cl_4.h"

#include <string>

using namespace std;

Derived3::Derived3(Base* parent,int state, string name) :
Base(parent,state,name) {}
```

### Файл cl\_4.h

```
#ifndef __CL_4_H
#define __CL_4_H
#include "cl_base.h"

class Derived3 : public Base {
public:
    Derived3(Base*,int, string);
};

#endif
```

### Файл cl\_5.cpp

```
#include "cl_5.h"

#include <string>

using namespace std;

Derived4::Derived4(Base* parent,int state, string name) :
Base(parent,state,name) {}
```

## Файл cl\_5.h

```
#ifndef __CL_5_H
#define __CL_5_H
#include "cl_base.h"

class Derived4 : public Base {

public:

    Derived4(Base*,int, string);

};

#endif
```

## Файл cl\_application.cpp

```
#include "cl_application.h"
#include "cl_1.h"
#include "cl_2.h"
#include "cl_3.h"
#include "cl_4.h"
#include "cl_5.h"
#include <iostream>

using namespace std;

Application::Application(Base* parent,int state) : Base(parent, state) {

}

Application::Application(Base* parent, int state,string name) :
Base(parent,state,name) {

}

void Application::buildTree() {
    string parent_name, child_name;
    int classNum, state;
    cin >> parent_name;
    setName(parent_name);
    temp_parent = this;
    root_parent = this;
    do {
        cin >> parent_name;
        if (parent_name == "endtree") return;
```



```

        if (parent_name != temp_parent->getName()) {
            temp_parent = find(parent_name);
        }
        cin >> child_name >> classNum >> state;
        switch(classNum)
        {
            case 2:
                temp_child = new Derived(temp_parent,
state,child_name);
                break;
            case 3:
                temp_child = new Derived1(temp_parent,
state,child_name);
                break;
            case 4:
                temp_child = new
Derived2(temp_parent,state,child_name);
                break;
            case 5:
                temp_child = new
Derived3(temp_parent,state,child_name);
                break;
            case 6:
                temp_child = new
Derived4(temp_parent,state,child_name);
                break;
        }
    } while (true);
}

int Application::execute() {
    cout<<"Object tree"<<endl;
    printNames(4);
    return 0;
}

```

### Файл cl\_application.h

```

#ifndef APPLICATION_H
#define APPLICATION_H
#include "cl_base.h"

class Application : public Base {
    Base* temp_parent;
    Base* temp_child;
    Base* root_parent;

public:
    Application(Base* parent = 0,int state=1 );
    Application(Base* parent,int state,string name);
    void buildTree();
    int execute();
}

```

```
};  
#endif
```

## Файл cl\_base.cpp

```
#include "cl_base.h"  
#include <iostream>  
using namespace std;  
  
Base::Base(Base* parent, int state, string name) {  
    setParent(parent);  
    setName(name);  
    setState(state);  
}  
  
void Base::setName(string name) {  
    this->name = name;  
}  
  
void Base::setState(int state)  
{  
    this->state = state;  
}  
  
string Base::getName() {  
    return this->name;  
}  
  
int Base::getState()  
{  
    return this->state;  
}  
  
void Base::setParent(Base* parent) {  
    this->parent = parent;  
    if (parent) {  
        parent->children.push_back(this);  
    }  
}
```

```

void Base::printNames(int k) {
    if (this->parent==0)
    {
        cout << this->name;
    }
    if (children.empty()) return;
    children_iterator = children.begin();
    while (children_iterator != children.end()) {
        cout<<endl;
        for(int i =0;i<k;i++) cout<<" ";
        cout << (*children_iterator)->getName();
        (*children_iterator)->printNames(k+4);
        children_iterator++;
    }
}

/*

void Base::printState() {
    if (this->parent==0)
    {
        cout << "The object " << this->name << " is ready";
    }
    if (children.empty()) return;
    children_iterator = children.begin();
    while (children_iterator != children.end()) {
        if ((*children_iterator)->getState()>0)
        {
            cout<<endl << "The object " << (*children_iterator)-
            >getName()<<" is ready";
            (*children_iterator)->printState();
        }
        else
        {
            cout<< endl << "The object " << (*children_iterator)-
            >getName() << " is not ready" ;
            (*children_iterator)->printState();
        }
    }
}

```

```

        children_iterator++;
    }children_iterator--;
}*/

Base* Base::find(string parentName)
{
    Base* search = nullptr;
    if (this->name == parentName)
    {
        search=this;
        return search;
    }
    else
    {
        if (!children.empty()) {
            children_iterator = children.begin();
            while (children_iterator != children.end())
            {
                if (search != 0) return search;
                search = (*children_iterator)-
>find(parentName);
                ++children_iterator;
            }--children_iterator;
        }
        return(search);
    }
}
}

```

## Файл cl\_base.h

```

#ifndef BASE_H
#define BASE_H
#include <vector>
#include <string>

using namespace std;

class Base {
    string name;
    int state;
    Base* parent;
    vector <Base*> children;
    vector <Base*>::iterator children_iterator;
public:
    Base(Base*, int, string = "base");

```

```

        void setName(string);
        string getName();
        void setParent(Base*);
        Base* getParent();
        void printNames(int);
        Base* find(string);
        void setState(int);
        int getState();
};

#endif

```

### Файл main.cpp

```

#include "main.h"

int main() {
    Application app;
    app.buildTree();
    return app.execute();
}

```

### Файл main.h

```

#ifndef MAIN_H
#define MAIN_H
#include "cl_application.h"

int main();

#endif

```

## Тестирование

Входные данные	Ожидаемые выходные	Фактические выходные
----------------	--------------------	----------------------

	данные	данные
app app ob1 3 1 app ob2 3 1 endtree	Object tree app ob1 ob2	Object tree app ob1 ob2