Постановка задачи

Для организации иерархического построения объектов необходимо разработать базовый класс, который содержит функционал и свойства для построения иерархии объектов. В последующем, в приложениях использовать этот класс как базовый для всех создаваемых классов. Это позволит включать любой объект в состав дерева иерархии объектов.

Создать базовый класс со следующими элементами:

Свойства:

- наименование объекта (строкового типа);
- указатель на головной объект для текущего объекта (для корневого объекта значение указателя равно 0);
- список указателей на объекты подчиненных к текущему объекту в дереве иерархии.

Функционал:

- параметризированный конструктор с параметром указателя на головной объект в дереве иерархии;
- параметризированный конструктор с параметром указателя на головной объект в дереве иерархии и наименованием объекта;
- метод задания имени объекта;
- метод получения имени объекта;
- метод вывода наименований объектов в дереве иерархии слева направо и сверху вниз.

Написать программу, которая последовательно строит дерево иерархии объектов, слева направо и сверху вниз. Переход на новый уровень происходит от правого (последнего) объекта предыдущего уровня. Для построения дерева использовать объекты двух производных классов, наследуемых от базового. Каждый объект имеет уникальное имя.

Построчно, по уровням вывести наименования объектов построенного иерархического дерева.

Описание входных данных

Первая строка:
«имя корневого объекта»
Создается корневой объект.
Вторая строка и последующие строки:
«имя головного объекта»«имя подчиненного объекта»
Создается подчиненный объект и добавляется в иерархическое дерево.

Если «имя головного объекта» равняется «имени подчиненного объекта», то новый объект не создается и построение дерева объектов завершается.

Примерввода

Object_root
Object_root Object_1
Object_root Object_2
Object_root Object_3
Object_3 Object_4
Object_3 Object_5
Object_6 Object_6

Дерево объектов, которое будет построено по данному примеру:

Object_root
Object_1
Object_2
Object_3
Object_4
Object_5

Описание выходных данных

Первая строка:
«имя корневого объекта»
Вторая строка и последующие строки имена головного и подчиненных объектов

очередного уровня разделенных двумя пробелами.
«имя головного объекта»«имя подчиненного объекта»[[«имя подчиненного объекта»] …….]

Пример вывода

Object_root
Object_root Object_1 Object_2 Object_3 Object_3 Object_4 Object_5

Метод решения

Используются стандартные потоки ввода и вывод cin и cout библиотеки iostream

Используется стандартный шаблон std::vector библиотеки vector

Используются объекты классов cl_base, Child1, Child2

Описание класса cl_base:

Свойства:

- name - наименование объекта (строковый тип)
- *parent - указатель на головоной объект для текущего объекта
- pChildList[] - список указателей на объекты подчиненных к текущему объекту в дереве иерархии

Функционал:

- cl_base(parent) - конструктор с параметром указателя на головной объект иерархии
- cl_base(name, parent) - конструктор с параметром указателя на головной объект в дереве и имя объекта
- setName - метод задания имени объекта
- getName - метод получения имени объекта
- findItem - метод поиска объекта в дереве
- printChildList - метод вывода наименований объектов в дереве

Описание класса Child1 публично насследованного от cl_base

Функционал:

- Child1 - конструктор принимающий имя объекта и указатель на головной объект иерархии и передает параметры конструктору родительского класса

Описание класса Child2 публично насследованного от cl_base

Функционал:

- Child2 - конструктор принимающий имя объекта и указатель на головной объект иерархии и передает параметры конструктору родительского класса

Описание алгоритма

№ шага	предикат	действие	№ перехо
1	-	string name1, name2	2
2	-	cin >> name1	3
3	-	Child1 Root(name1)	4
4	-	cin >> name1 >> name2	5
5	name1 != name2	-	6
	name1 == name2	cout << Root.getName()	9
6	name1 == Root.getName()	new Child1 (&Root, name2)	8
	name1 != Root.getName()	ThreeItem *parent = Root.findItem(name1);	7
7	parent != nullptr	new Child2(parent, name2)	8
	parent == nullptr	-	8
8	-	cin >> name1 >> name2	5
9	-	cout << Root.getName()	10
10	-	Root.printChildList()	0

* Синим цветом для удобства обозначен цикл while

Алгоритм cl_base::cl_base(cl_base*parent)

№ шага	предикат	действие	№ перехода
1	-	this->parent = parent	2
2	parent != nulltpre	parent->addChild(this)	0
	parent == nulltpre	-	0

Алгоритм cl_base::cl_base(string name, cl_base* parent)

№ шага	предикат	действие	№ перехода
1	-	this->parent = parent	2
2	-	this->name = name	2
3	parent != nulltpre	parent->addChild(this)	0
	parent == nulltpre	-	0

Алгоритм cl_base::setName(string name)

№ шага	предикат	действие	№ перехода
1	-	this->name = name	0

Алгоритм cl_base::addChild(cl_base*child)

№ шага	предикат	действие	№ перехода
1	-	pChildList.push_back(child)	0

Алгоритм cl_base::findItem(string name, cl_base* &parent)

№ шага	предикат	действие	№ перехода
1	child : pChildList	-	2
	child == pChildList.end()	-	0
2	child->getName() == name	parent = child	1
	child->getName() != name	child->findItem(name, parent)	1

Алгоритм cl_base::pringChildList()

№ шага	предикат	действие	№ перехода
1	pChildList.size() < 1	-	0
	pchildList() >= 1	-	2
2	-	cout << endl << name	3
3	child : pChildList	cout << " " << child->getName()	3
	child == pChildList.end()	-	4
4	child : pChildList	child->printChildList()	4
	child == pChildList.end()	-	0

Алгоритм cl_base::~cl_base()

№ шага	предикат	действие	№ перехода
1	child : pChildList	delete Child	1
	child = pChildList.end()	-	0

Алгоритм Child1::Child1(parent, name):cl_base(name, parent)

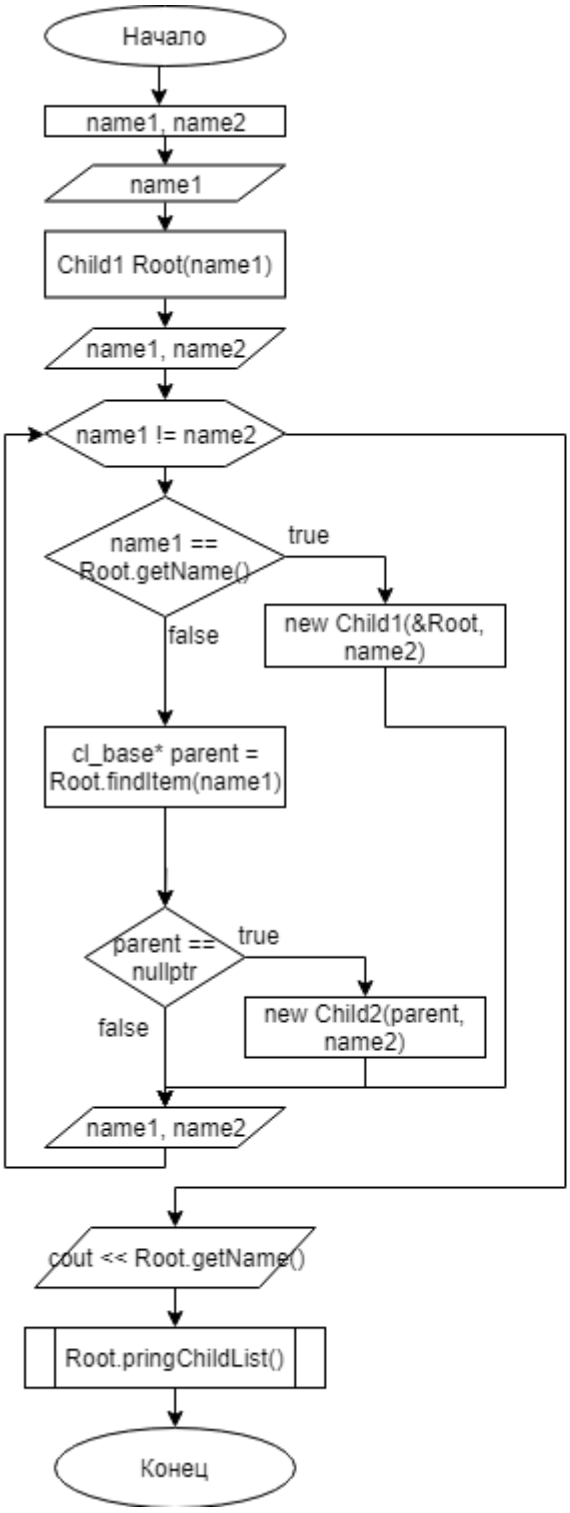
№ шага	предикат	действие	№ перехода

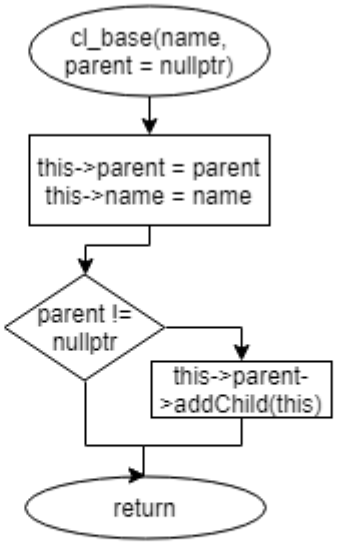
1	-	parent->AddChild	0

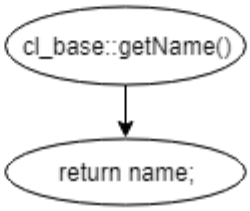
Алгоритм Child2::Child2(parent, name):cl_base(parent, name)

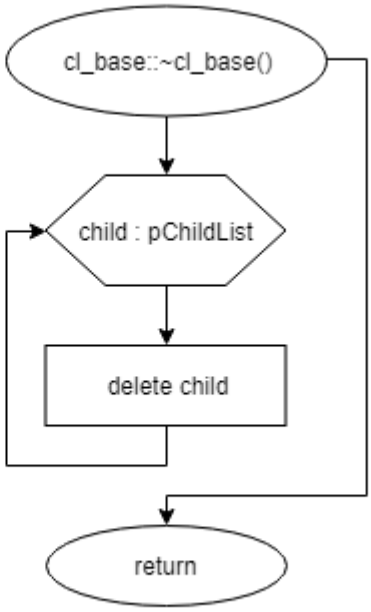
№ шага	предикат	действие	№ перехода
1	-	parent->AddChild	0

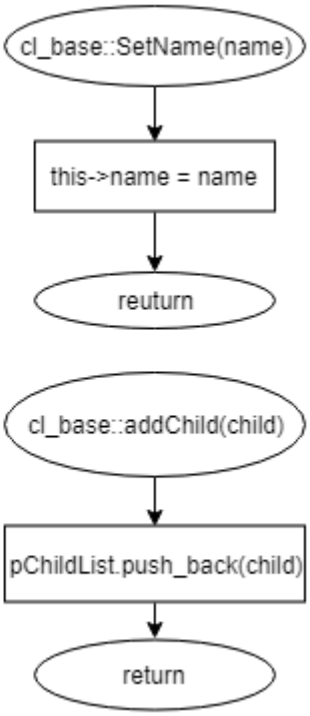
Блок-схема алгоритма

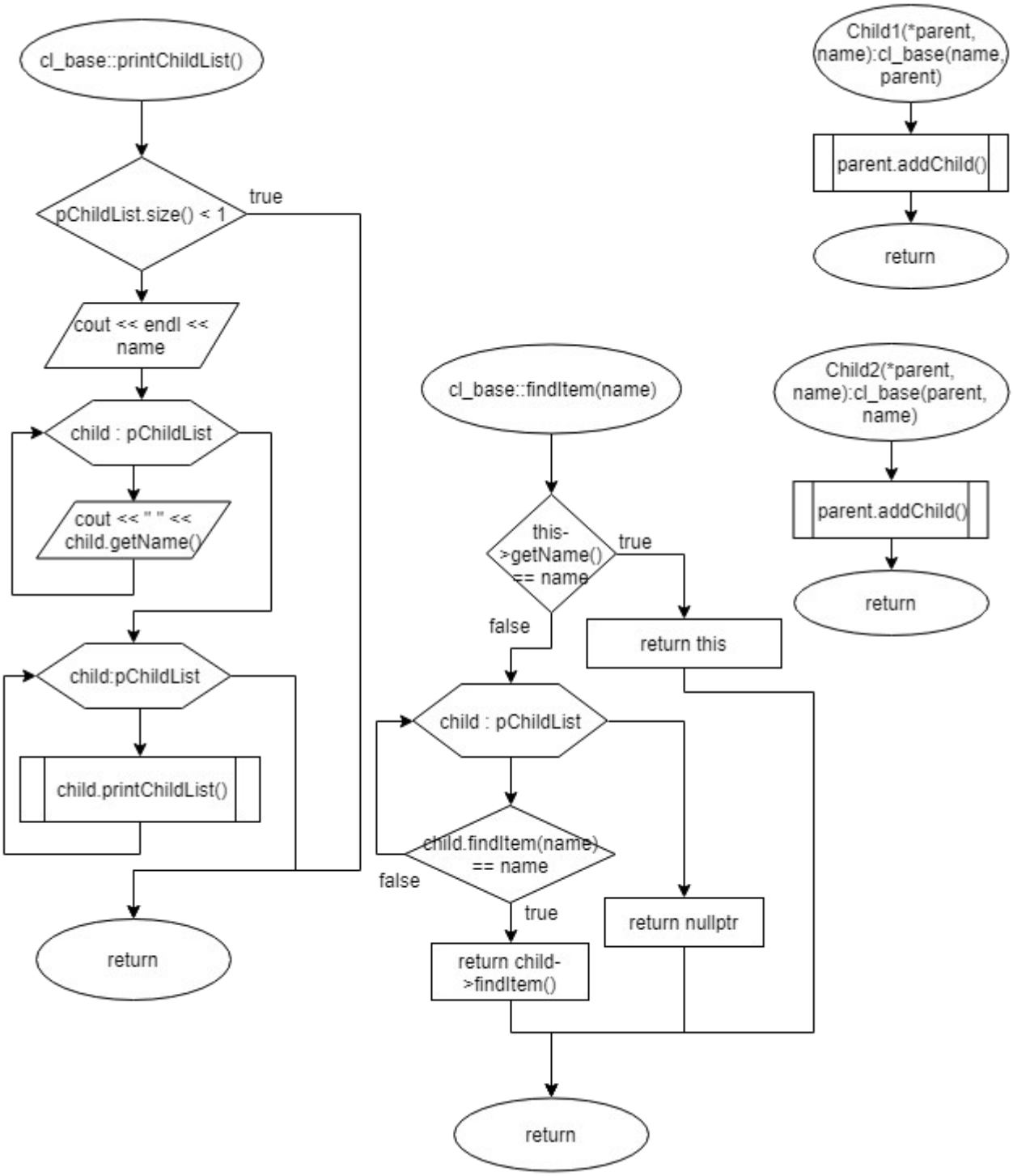

Начало
name1, name2
name1
Child1 Root(name1)
name1, name2
name1 != name2
name1 == Root.getName()
true
new Child1(&Root, name2)
false
cl_base* parent = Root.findItem(name1)
parent == nullptr
true
new Child2(parent, name2)
false
name1, name2
cout << Root.getName()
Root.pringChildList()
Конец


cl_base(name, parent = nullptr)
this->parent = parent
this->name = name
parent != nullptr
this->parent->addChild(this)
return


cl_base::getName()
return name;


cl_base::~cl_base()
child : pChildList
delete child
return


cl_base::SetName(name)
this->name = name
reuturn
cl_base::addChild(child)
pChildList.push_back(child)
return



Код программы

Файл Child1.cpp

```
#include "Child1.h"

Child1::Child1(string name, cl_base* parent):cl_base(name, parent){
}
```

Файл Child1.h

```
#ifndef __CHILD1_H
#define __CHILD1_H
#include "cl_base.h"

class Child1 : public cl_base{
public:
        Child1(string name, cl_base* parent = nullptr);

};

#endif
```

Файл Child2.cpp

```
#include "Child2.h"

Child2::Child2(cl_base* parent, string name):cl_base(name, parent){
}
```

Файл Child2.h

```
#ifndef __CHILD2_H
#define __CHILD2_H
#include "cl_base.h"

class Child2 : public cl_base{
public:
        Child2(cl_base* parent, string name);

};
```

```
#endif
```

Файл cl_base.cpp

```
#include "cl_base.h"

cl_base::cl_base(cl_base* parent){
        this->parent = parent;
        if (parent != nullptr)
                this->parent->addChild(this);
}

cl_base::cl_base(string name, cl_base* parent){
        this->parent = parent;
        this->name = name;
        if (parent != nullptr)
                this->parent->addChild(this);
}


void cl_base::setName(string name){this->name = name;}
string cl_base::getName(){return name;}

void cl_base::addChild(cl_base* child){
                pChildList.push_back(child);
}

cl_base* cl_base::findItem(string name){
        if (this->getName() == name) return this;

        for (auto child : pChildList){
                if (child->findItem(name) != nullptr){
                        return child->findItem(name);
                }
        }

        return nullptr;
}

void cl_base::printChildList(){
        if (pChildList.size() < 1)
                return;
        else{
                cout << endl << name;

                for (auto child : pChildList)
                        cout << "  " << child->getName();

                for (auto child : pChildList)
```

```
                        child->printChildList();
        }
}

cl_base::~cl_base(){
        for (auto child : pChildList)
                delete child;
}
```

Файл cl_base.h

```
#ifndef __PARENT_H
#define __PARENT_H

#include <iostream>
#include <vector>
#include <string>

using namespace std;

class cl_base{
        string name;
        cl_base* parent;
        vector<cl_base*> pChildList;
public:
        cl_base(cl_base* parent = nullptr);
        cl_base( string name, cl_base* parent = nullptr);

        void setName(string name);
        string getName();

        void addChild(cl_base* child);

        cl_base* findItem(string name);
        void printChildList();

        ~cl_base();
};

#endif
```

Файл main.cpp

```
//#include "cl_base.h"
#include "Child1.h"
#include "Child2.h"
```

```
int main(){
        string name1, name2;

        cin >> name1;

        Child1 Root(name1);

        cin >> name1 >> name2;
        while(name1 != name2){
                if (name1 == Root.getName())
                        new Child1(name2, &Root);
                else {
                        cl_base *parent = Root.findItem(name1);

                        if (parent != nullptr)
                                new Child2(parent, name2);
                }
                cin >> name1 >> name2;
        }

        cout << Root.getName();
        Root.printChildList();
        return 0;
}
```

Тестирование

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
Root Obj1 Obj1	Root	Root
Root Root Obj1 Root Obj2 Obj1 Obj3 Obj2 Obj4 6 6	Root Root Obj1 Obj2 Obj1 Obj3 Obj2 Obj4	Root Root Obj1 Obj2 Obj1 Obj3 Obj2 Obj4

Root Root Obj1 Root Obj2 Obj1 Obj3 6 6	Root Root Obj1 Obj2 Obj1 Obj3	Root Root Obj1 Obj2 Obj1 Obj3
root root obj1 obj1 obj2 obj2 obj3 1 1	root root obj1 obj1 obj2 obj2 obj3	root root obj1 obj1 obj2 obj2 obj3