

Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №6.
"Наследование и виртуальные функции"

Выполнил:
Ст. 2 курса гр. АС-53
Анискин Д.В.
Проверила:
Давидюк Ю. И.

Брест, 2020

1. **Цель.** Получить практические навыки создания иерархии классов и использования статических компонентов класса.

2. Постановка задачи (Вариант 3)

Написать программу, в которой создается иерархия классов. Включить полиморфные объекты в связанный список, используя статические компоненты класса. Показать использование виртуальных функций.

служащий, персона, рабочий, инженер;

Классы: Employee, Person, Worker, Engineer

Конструкторы:

- Пустой
- С параметрами
- Копирования

Деструктор.

Виртуальные функции:

- Добавления в список
- Вывода информации

3. Иерархия классов в виде графа:

- Персона
 - Рабочий
 - Инженер
 - Служащий

4. Определение пользовательских классов с комментариями.

```
//Базовый класс
class person // базовый класс
{
public:
    static person* begin; //указатель на начало списка
    person* next = NULL;
    static void ShowList() //список
    {
        person* p = begin;
        while (p)
        {
            p->show();
            p = p->next;
        }
    }
    person() //без параметров
    {
        name = new char[81];
    }
    person(const char* NAME, int age1) //с параметрами
    {
        // выделение памяти для name. размер выделяемой памяти = длина строки NAME
        name = new char[strlen(NAME) + 1];
        strcpy(name, NAME);
    }
};
```

```

        age = age1;
    }
    ~person() // деструктор
    {
        cout << "Person object deleted" << endl;
    }
    virtual void show() = 0; //Чистая виртуальная функция
    virtual void input() = 0;

protected:
    char* name;
    int age;
};

class worker :public person // производный класс
{
public:
    worker() : person() {} //без параметров
    worker(const char* NAME, int age1, int EXPERIENCE, string POSITION, bool AddToList = false)
    :person(NAME, age1) //с параметрами
    {
        if (AddToList)
        {
            person* p = begin;
            while (p->next)
            {
                p = p->next;
            }
            p->next = this;
        }
        experience = EXPERIENCE;
        position = POSITION;
    }
    void show()
    {
        cout << "\nКласс: Рабочий";
        cout << "\nИмя: " << name;
        cout << "\nВозраст: " << age;
        cout << "\nДолжность: " << position;
        cout << "\nОпыт работы: " << experience;
        cout << "\n";
    }
    void input()
    {
        cout << "\nИмя Рабочего: ";
        cin >> name;
        cout << "\nВозраст: ";
        cin >> age;
        cout << "\nДолжность: ";
        cin >> position;
        cout << "\nОпыт Работы: ";
        cin >> experience;
        cout << "\n";
    }
    ~worker() // деструктор
    {
        cout << "worker object deleted" << endl;
    }

protected:
    int experience;
    string position;
};

class engineer :public worker // производный класс
{
public:
    engineer() : worker() {}

```

```

    engineer(const char* NAME, int age1, int EXPERIENCE, string POSITION, string QUALIFICATION,
    bool AddToList = false) :worker(NAME, age1, EXPERIENCE, POSITION)
    {
        if (AddToList)
        {
            person* p = begin;
            while (p->next)
            {
                p = p->next;
            }
            p->next = this;
        }
        qualification = QUALIFICATION;
    }
    void show()
    {
        cout << "\nКласс: Инженер";
        cout << "\nИмя: " << name;
        cout << "\nВозраст: " << age;
        cout << "\nДолжность: " << position;
        cout << "\nОпыт работы: " << experience;
        cout << "\nКвалификация: " << qualification;
        cout << "\n";
    }
    void input()
    {
        cout << "\nИмя Инженера: ";
        cin >> name;
        cout << "\nВозраст:";
        cin >> age;
        cout << "\nДолжность: ";
        cin >> position;
        cout << "\nОпыт Работы: ";
        cin >> experience;
        cout << "\nКвалификация: ";
        cin >> qualification;
        cout << "\n";
    }
    ~engineer() // деструктор
    {
        cout << "engineer object deleted" << endl;
    }
protected:
    string qualification;
};
class employee :public person // производный класс
{
public:
    employee() : person() {}

    employee(const char* NAME, int age1, string RANK ,bool AddToList = false) :person(NAME, age1)
    {
        if (AddToList)
        {
            person* p = begin;
            while (p->next)
            {
                p = p->next;
            }
            p->next = this;
        }
        rank = RANK;
    }
    void show()

```

```

{
    cout << "\nКласс: Служащий";
    cout << "\nИмя: " << name;
    cout << "\nВозраст: " << age;
    cout << "\nЗвание: " << rank;
    cout << "\n";
}
void input()
{
    cout << "\nИмя Служащего: ";
    cin >> name;
    cout << "\nВозраст: ";
    cin >> age;
    cout << "\nЗвание: ";
    cin >> rank;
    cout << "\n";
}
~employee() // деструктор
{
    cout << "employee object deleted" << endl;
}
protected:
    string rank;
};

```

Реализация
 конструкторов
 с
 параметрами

и
деструктора.

- Для класса Person:

```
person() //без параметров
{
    name = new char[81];
}
person(const char* NAME, int age1) //с параметрами
{
    // выделение памяти для name. размер выделяемой памяти = длина строки NAME
    name = new char[strlen(NAME) + 1];
    strcpy(name, NAME);
    age = age1;
```

- }Для класса Worker:

```
worker() : person() {} //без параметров
worker(const char* NAME, int age1, int EXPERIENCE, string POSITION, bool AddToList = false)
:person(NAME, age1) //с параметрами
{
    if (AddToList)
    {
        person* p = begin;
        while (p->next)
        {
            p = p->next;
        }
        p->next = this;
    }
    experience = EXPERIENCE;
    position = POSITION;
```

- }Для класса Employee:

```
employee() : person() {}

employee(const char* NAME, int age1, string RANK, bool AddToList = false) :person(NAME, age1)
{
    if (AddToList)
    {
        person* p = begin;
        while (p->next)
        {
            p = p->next;
        }
        p->next = this;
    }
    rank = RANK;
```

- }Для класса Engeneer:

```

engineer() : worker() {}

engineer(const char* NAME, int age1, int EXPERIENCE, string POSITION, string QUALIFICATION,
bool AddToList = false) :worker(NAME, age1, EXPERIENCE, POSITION)
{
    if (AddToList)
    {
        person* p = begin;
        while (p->next)
        {
            p = p->next;
        }
        p->next = this;
    }
    qualification = QUALIFICATION;
}

```

5. Реализация метода для просмотра списка.

```

void show()
{
    cout << "\nКласс: Служащий";
    cout << "\nИмя: " << name;
    cout << "\nВозраст:" << age;
    cout << "\n";
}

```

6. Листинг демонстрационной программы.

```

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    worker* a1;
    employee* a2;
    engineer* a3;
    a1 = new worker;
    a2 = new employee;
    a3 = new engineer;
    a1->input();
    a2->input();
    a3->input();
    cout << " ----- \n";
    person::begin = a1;
    a1->next = a2;
    a2->next = a3;
    engineer* x4 = new engineer("Dimon-patron", 20, 5, "electric",
    "3 grade", true); // Создание объекта класса person::ShowList();

    return 0;
}

```

8. Вывод программы:

Имя Рабочего: ваня

Возраст: 22

Должность: банкир

Опыт Работы: 2

Имя Служащего: виталья

Возраст: 23

Звание: полковник

Имя Инженера: вася

Возраст: 24

Должность: электрик

Опыт Работы: 3

Квалификация: 3 разряд

Класс: Рабочий

Имя: ваня

Возраст: 22

Должность: банкир

Опыт работы: 2

Класс: Служащий

Имя: виталья

Возраст: 23

Звание: полковник

Класс: Инженер

Имя: вася

Возраст: 24

Должность: электрик

Опыт работы: 3

Квалификация: 3

Класс: Инженер

Имя: Dimon-patron

Возраст: 20

Должность: electric

Опыт работы: 5

Квалификация: 3 grade

Объяснение необходимости виртуальных функций. Следует показать, какие

результаты будут в случае виртуальных и не виртуальных функций.

При наследовании бывает необходимо, чтобы поведение некоторых методов базового класса и классов-наследников различалось, именно для этого и требуется наличие виртуальных функций `virtual void Show() = 0;` `virtual void Add() = 0;`

В данном коде, в случае отсутствия виртуальной функции нельзя будет переопределить поведение.

9. Вывод:

Получил практические навыки реализации классов на C++.