Лабораторная работа № 5

Тема: Наследование в C++. Пример использования.

Цель работы. Изучить принципы применения наследования

Практическая часть

Наследование является одним из самых мощных механизмов объектно-ориентированного программирования и, прежде всего, позволяет избежать лишнего дублирования программного кода. Рассмотрим принципы наследования классов на примере задачи хранения информации о студентах и преподавателях учебного заведения. Очевидно, что представить все данные о студентах и преподавателях в одном классе не удобно получится, поскольку данные для них в некоторой части различаются. Что-то необходимо хранить только для студентов, что-то — только для преподавателей.

Создадим базовый класс human, который будет описывать модель человека и в нем будут храниться имя, фамилия и отчество. Создайте новый проект и добавьте в него файл `human.h`:

```
// human.h
#ifndef HUMAN_H_INCLUDED
#define HUMAN_H_INCLUDED

#include <string>
#include <sstream>

class human {
   public:
      // Конструктор класса human
      human(std::string last_name, std::string name, std::string second_name)
      {
         this->last_name = last_name;
         this->name = name;
         this->second_name = second_name;
      }

      // Получение ФИО человека
      std::string get_full_name()
      {
         std::ostringstream full_name;
         full_name << this->last_name << " "
            << this->name << " "
            << this->second_name;
         return full_name.str();
      }

   private:
      std::string name; // имя
      std::string last_name; // фамилия
      std::string second_name; // отчество
};

#endif // HUMAN_H_INCLUDED
```

Теперь создаем новый класс `student`, который будет наследником класса `human`. Для этого в проект добавляем еще один файл - `student.h`.

```
// student.h
#ifndef STUDENT_H_INCLUDED
#define STUDENT_H_INCLUDED

#include "human.h"
#include <string>
#include <vector>

class student : public human {

   public:

      // Конструктор класса Student
       student(
          std::string last_name,
          std::string name,
          std::string second_name,
          std::vector<int> scores
       ) : human(
          last_name,
          name,
          second_name
       ) {
          this->scores = scores;
       }

      // Получение среднего балла студента
       float get_average_score()
       {
         // Общее количество оценок
          unsigned int count_scores = this->scores.size();
         // Сумма всех оценок студента
          unsigned int sum_scores = 0;
         // Средний балл
          float average_score;

          for (unsigned int i = 0; i < count_scores; ++i) {
             sum_scores += this->scores[i];
          }

          average_score = (float) sum_scores / (float) count_scores;
          return average_score;
       }

   private:
      // Оценки студента
       std::vector<int> scores;
};
#endif // STUDENT_H_INCLUDED
```

Функция get_average_score вычисляет среднее арифметическое всех оценок студента. Все публичные свойства и методы класса human будут доступны в классе student.
Конструктор класса `student` использует конструктор класса `human` и передает в него фамилию, имя и отчество человека, которые сохраняются в экземпляре класса. Для класса `student`, также необходимо задать еще и список оценок студента. Поэтому конструктор `student` принимает все аргументы конструктора базового класса, а также дополнительные аргументы для хранения оценок. Список оценок студента хранится в векторе. Вектор — это специально разработанная структура данных, предназначенная для хранения однотипных данных. Основным ее отличием от массива является автоматическое выделение памяти для хранения данных, а также использование,

связанных с ней операций. Например, `push_back()` добавляет элемент в конец вектора, `size()` позволяет определить количество элементов в векторе, `pop_back()` удаляет последний элемент из вектора, `empty()` проверяет вектор на пустоту.
Ниже приведен пример использования класса `student.` Выполните коррекцию файла main.cpp в проекте, в соответствии со следующим

```
// main.cpp

#include <iostream>
#include <vector>

#include "human.h"
#include "student.h"

int main(int argc, char* argv[ ])
{

   // Оценки студента будут храниться в структуре данных типа «вектор»
   std::vector<int> scores;

   // Добавление оценок студента в вектор
   scores.push_back(5);
   scores.push_back(3);
   scores.push_back(2);
   scores.push_back(2);
   scores.push_back(5);
   scores.push_back(3);
   scores.push_back(3);
   scores.push_back(3);
   scores.push_back(3);

   // Создание объекта класса student
   student *stud = new student("Петров", "Иван", "Алексеевич", scores);

   // Вывод полного имени студента (используется унаследованный метод класса human)
   std::cout << stud->get_full_name() << std::endl;
   // Вывод среднего балла студента
   std::cout << "Средний балл: " << stud->get_average_score() << std::endl;

   return 0;
}
```

Вывод полного имени студента выполняется с помощью функции get_full_name, которая унаследована от базового класса human. Для вычисления среднего балла успеваемости студента используется функция `get_average_score`, которая описана внутри класса `student`. Для выполнения проект должен включать 3 файла, как показано на рис.1

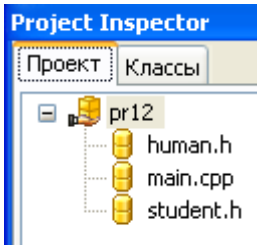

Рис.1.

Создание класса-наследника teacher

Рассмотрим создание еще одного класса-наследника (`teacher`), в котором будут храниться данные преподавателей. Для хранение имени, фамилии и отчества, а также получения значений этих атрибутов этот класс будет наследовать методы и атрибуты от класса `human`. Для размещения класса создадим в проекте еще один заголовочный файл `teacher.h`:

```
// teacher.h
#ifndef TEACHER_H_INCLUDED
#define TEACHER_H_INCLUDED

#include "human.h"
#include <string>

class teacher : public human {
   // Конструктор класса teacher
   public:
      teacher(
         std::string last_name,
         std::string name,
         std::string second_name,
        // Количество учебных часов за семетр у преподавателя
         unsigned int work_time
      ) : human(
         last_name,
         name,
         second_name
      ) {
         this->work_time = work_time;
      }

      // Получение количества учебных часов
      unsigned int get_work_time()
      {
         return this->work_time;
      }

   private:
      // Учебные часы
      unsigned int work_time;
};

#endif // TEACHER_H_INCLUDED
```

В классе teacher добавлен новый атрибут work_time (с видимостью private) — количество учебных часов, которые проводит преподаватель в течении одного семестра. Значение атрибута задается при создании нового объекта при помощи конструктора. Для получения значения атрибута в классе имеется функция get_work_time(). Все остальные атрибуты и методы наследуются от базового класса human.
Ниже приведен пример использования класса teacher. Выполните коррекцию файла main.cpp в проекте, в соответствии со следующим

```
#include <iostream>

#include "human.h"
#include "teacher.h"

int main(int argc, char* argv[])
{
```

```
    // Количество учебных часов преподавателя
    unsigned int teacher_work_time = 40;

    teacher *tch = new teacher("Васильков", "Петр", "Сергеевич", teacher_work_time);

    std::cout << tch->get_full_name() << std::endl;
    std::cout << "Количество часов: " << tch->get_work_time() << std::endl;

    return 0;
}
```

Задания к работе

1. Изучите приведенные примеры. В отчет к работе предоставьте скриншоты результатов работы программ.
2. Внесите изменения в класс `student` такие, чтобы в нем присутствовали атрибуты адреса проживания студента, а также названия изучаемых дисциплин. Предоставьте скриншоты результатов работы программы.
3. Внесите изменения в класс teacher такие, чтобы в нем присутствовали атрибуты названий преподаваемых дисциплин и времени проведения преподавателем консультаций (день недели, часы, минуты). Предоставьте скриншоты результатов работы программы.
4. Составьте отчет, в который включите код программ и скриншоты результатов их работы.