Информация для размещения на официальном сайте ГБПОУ «Светлоградский региональный сельскохозяйственный колледж»

Для электронного обучения

Группа	421
Дата	09.11.2021 г
Время	10-10-11-00
Наименование УД/МДК/УП/ПП	МДК 01.02
Ф.И.О. преподавателя	Сахарчук Т.В.
Электронная почта	saharchyk777@mail.ru
Основная литература	1. Интеллектуальные системы и технологии. Учебник и практикум для СПО Станкевич Л. А. Научная школа: Санкт-Петербургский политехнический университет Петра Великого (г. Санкт-Петербург). Год: 2019 / Гриф УМО СПО https://biblio-online.ru/book/intellektualnye-sistemy-i-tehnologii-445852 2. Федорова Г.Н. Разработка и администрирование баз данных (2-е изд., стер.) учебник«Академия»2019 г. 3. Федорова Г.Н. Информационные системы (6-е изд., стер.) учебник «Академия» 2019г 4. Рудаков А.В. Технология разработки программных продуктов (12-е изд.) учебник«Академия»2018 г.
Тема № 57-58	Практическое занятие на тему: Компоновка нескольких файлов в одну программу. Использование включаемых файлов.
Задание	**Использование включаемых файлов** В языке Си++ реализовано удобное решение. Можно поместить объявления классов и функций в отдельный *файл* и включать этот *файл* в начало других файлов с помощью директивы #include. #include "Book.h" . . . Book b; Фактически ***директива*** #include подставляет содержимое файла Book.h в текущий файл перед тем, как начать его компиляцию. Эта *подстановка* осуществляется во время первого прохода компилятора по программе – препроцессора. *Файл* Book.h называется файлом заголовков. В такой же *файл* заголовков можно поместить *прототипы функций* и включать его в другие файлы, там, где функции используются. Таким образом, текст программы на языке Си++ помещается в файлы двух типов – файлы заголовков и файлы программ. В большинстве случаев имеет смысл каждый *класс* помещать в отдельный *файл*, вернее, два файла – *файл* заголовков для объявления класса и *файл* программ для определения класса. *Имя файла* обычно состоит из имени класса. Для файла заголовков к нему добавляется окончание " .h " (иногда, особенно в системе Unix, " .hh " или " .H "). *Имя файла* программы – опять-таки *имя класса* с окончанием " .cpp " (иногда " .cc " или " .C "). Объединять несколько классов в один *файл* стоит лишь в том случае, если они очень тесно связаны и один без другого не

используется.
Включение файлов может быть вложенным, т.е. *файл* заголовков может сам использовать директиву #include. *Файл* Book.h выглядит следующим образом:

```
#ifndef __BOOK_H__
#define __BOOK_H__
// включить файл с объявлением используемого
// здесь базового класса
#include "Item.h"
#include "String.h"
// объявление класса String

// объявление класса Book
class Book : public Item
{
public:
. . .
private:
        String title;
    . . .
}; #endif
```

Обратите внимание на первые две и последнюю строки этого файла. *Директива* #ifndef начинает блок так называемой условной компиляции, который заканчивается директивой #endif. Блок условной компиляции – это кусок текста, который будет компилироваться, только если выполнено определенное условие. В данном случае условие заключается в том, что символ __BOOK_H__ не определен. Если этот символ определен, текст между #ifndef и #endif не будет включен в программу. Первой директивой в блоке условной компиляции стоит *директива* #define, который определяет символ __BOOK_H__ как пустую строку.
Давайте посмотрим, что произойдет, если в какой-либо .cpp - *файл* будет дважды включен *файл* Book.h:

```
#include "Book.h"
. . .
#include "Book.h"
```

Перед началом компиляции текст файла Book.h будет подставлен вместо директивы #include:

```
#ifndef __BOOK_H__
#define __BOOK_H__
. . .
class Book
{
. . .
};
#endif
. . .
#ifndef __BOOK_H__
#define __BOOK_H__
. . .
class Book
```

```
{
. . .
};
#endif
```

В самом начале символ __BOOK_H__ не определен, и блок условной компиляции обрабатывается. В нем определяется символ __BOOK_H__. Теперь условие для второго блока условной компиляции уже не выполняется, и он будет пропущен. Таким образом, объявление класса Book будет вставлено в *файл* только один раз. Разумеется, написание два раза подряд директивы #include с одинаковым аргументом легко поправить. Однако структура заголовков может быть очень сложной. Чтобы избежать необходимости отслеживать все вложенные *заголовки* и искать, почему какой-либо *файл* оказался вставленным дважды, можно применить изложенный выше прием и существенно упростить себе жизнь.

Еще одно замечание по составлению заголовков. Включайте в заголовок как можно меньше других заголовков. Например, в заголовок Book.h необходимо включить *заголовки* Item.h и String.h, поскольку *класс* Book использует их. Однако если используется лишь *имя класса* без упоминания его содержимого, можно обойтись и объявлением этого имени:

```
#include "Item.h"
#include "String.h"
class Annotation;
// Annotation – имя некоего класса
class Book : public Item
{
public:
    Annotation* CreateAnnotation();
private:
    String title;
};
```

Объявление класса Item требуется знать целиком, для того, чтобы обработать объявление класса Book, т.е. компилятору надо знать все методы и атрибуты Item, чтобы включить их в *класс* Book. Объявление класса String также необходимо знать целиком, по крайней мере, для того, чтобы правильно вычислить размер экземпляра класса Book. Что же касается класса Annotation, то ни размер его объектов, ни его методы не важны для определения содержимого объекта класса Book. Единственное, что надо знать, это то, что Annotation есть имя некоего класса, который будет определен в другом месте.

Общее правило таково, что если объявление класса использует *указатель* или ссылку на другой *класс* и не задействует никаких методов или атрибутов этого класса, достаточно объявления имени класса. Разумеется, полное объявление класса Annotation понадобится в определении метода CreateAnnotation.

Компилятор поставляется с набором файлов заголовков, которые описывают все стандартные функции и классы. При *включении*

	стандартных файлов обычно используют немного другой *синтаксис*: #include <string.h>
Контрольный тест	Ответьте на вопросы: 1. С помощью какой директивы можно объявить классы и функции в отдельный файл? 2. Охарактеризуйте директиву #include 3. Охарактеризуйте Файл Book.h 4. Охарактеризуйте директиву #endif 5. Охарактеризуйте объявление класса Item 6. Охарактеризуйте объявление класса String

Дата____09.11.2021 г _____ _____ _____

Подпись Ф.И.О. преподавателя

Информация для размещения на официальном сайте ГБПОУ «Светлоградский региональный сельскохозяйственный колледж»

Для электронного обучения

Группа	421
Дата	09.11.2021 г
Время	11-10-12-00
Наименование УД/МДК/УП/ПП	МДК 01.02
Ф.И.О. преподавателя	Сахарчук Т.В.
Электронная почта	saharchyk777@mail.ru
Основная литература	1. Интеллектуальные системы и технологии. Учебник и практикум для СПО Станкевич Л. А. Научная школа: Санкт-Петербургский политехнический университет Петра Великого (г. Санкт-Петербург). Год: 2019 / Гриф УМО СПО https://biblio-online.ru/book/intellektualnye-sistemy-i-tehnologii-445852 2. Федорова Г.Н. Разработка и администрирование баз данных (2-е изд., стер.) учебник«Академия»2019 г. 3. Федорова Г.Н. Информационные системы (6-е изд., стер.) учебник «Академия» 2019г 4. Рудаков А.В. Технология разработки программных продуктов (12-е изд.) учебник«Академия»2018 г.
Тема № 59-60	Лекция на тему: Файлы и переменные. Общие данные. Глобальные переменные. Повышение надежности обращения к общим данным.
Задание	Файлы и переменные Автоматические переменные определены внутри какой-либо функции или метода класса. Назначение автоматических переменных – хранение каких-либо данных во время выполнения функции или метода. По завершении выполнения этой функции автоматические переменные уничтожаются и данные теряются. С этой точки зрения автоматические переменные представляют собой временные переменные. Иногда временное хранилище данных требуется на более короткое время, чем выполнение всей функции. Во-первых, поскольку в Си++ необязательно, чтобы все используемые переменные были определены в самом начале функции или метода, переменную можно определить непосредственно перед тем, как она будет использоваться. Во-вторых, переменную можно определить внутри блока – группы операторов, заключенных в фигурные скобки. При выходе из блока такая переменная уничтожается еще до окончания выполнения функции. Третьей возможностью временного использования переменной является определение переменной в заголовке цикла for только для итераций этого цикла: *funct(int N, Book[]& bookArray) {* Int X; // автоматическая переменная X *for (int i = 0; i < N; i++) { // переменная i определена только на время выполнения цикла for* *String s; // новая автоматическая переменная создается при каждой итерации цикла заново*

	s.Append(bookArray[i].Title()); *s.Append(bookArray[i].Author());* *cout << s;* *}* *cout << s; // ошибка, переменная s не существует* *}* Если переменную, определенную внутри функции или блока, описать как статическую, она не будет уничтожаться при выходе из этого блока и будет хранить свое значение между вызовами функции. Однако при выходе из соответствующего блока эта переменная станет недоступна, иными словами, невидима для программы. В следующем примере переменная allAuthors накапливает список авторов книг, переданных в качестве аргументов функции funct за все ее вызовы: *funct(int n, Book[]& bookArray){* *for (int i = 0; i < n; i++) {* *static String allAuthors;* *allAuthors.Append(bookArray[i].Author());* *cout << allAuthors; // авторы всех ранее обработанных книг, в том числе в предыдущих вызовах функции* *}* *cout << allAuthros; // ошибка, переменная недоступна* *}* Глобальные переменные Язык Си++ предоставляет возможность определения глобальной переменной. Если переменная определена вне функции, она создается в самом начале выполнения программы (еще до начала выполнения main). Эта переменная доступна во всех функциях того файла, где она определена. Аналогично прототипу функции, имя глобальной переменной можно объявить в других файлах и тем самым предоставить возможность обращаться к ней и в других файлах: *// файл main.cpp* *#include "RandomGenerator.h"* *RandomGenerator rgen; // определение глобальной переменной* *main() {* *rgen.Init(1000);* *}* *Void fun1(void) {* *unsigned long x = rgen.GetNumber();* *. . .* *}* *// файл class.cpp* *#include "RandomGenerator.h"* *extern RandomGenerator rgen; // объявление глобальной переменной, внешней по отношению к данному файлу* *Class1::Class1() {* *. . .* *}* *Void fun2() {* *unsigned long x = rgen.GetNumber();* *. . .*

	} Объявление внешней переменной можно поместить в файл-заголовок. Тогда не нужно будет повторять объявление переменной с описателем extern в каждом файле, который ее использует. Модификацией определения глобальной переменной является добавление описателя static. Для глобальной переменной описатель static означает то, что эта переменная доступна только в одном файле – в том, в котором она определена. (Правда, в данном примере такая модификация недопустима – нам-то как раз нужно, чтобы к глобальной переменной rgen можно было обращаться из разных файлов.)
Контрольный тест	Ответьте на вопросы: 1. Что такое Файлы? 2. Что такое переменные? 3. Охарактеризуйте Глобальные переменные. 4. Охарактеризуйте повышение надежности обращения к общим данным. 5. Охарактеризуйте объявление внешней переменной 6. Что является модификацией определения глобальной переменной

Дата____09.11.2021 г _____ _____ _____

Подпись Ф.И.О. преподавателя

Информация для размещения на официальном сайте ГБПОУ «Светлоградский региональный сельскохозяйственный колледж»

Для электронного обучения

Группа	421
Дата	10.11.2021 г
Время	10-10-11-00
Наименование УД/МДК/УП/ПП	МДК 01.02
Ф.И.О. преподавателя	Сахарчук Т.В.
Электронная почта	saharchyk777@mail.ru
Основная литература	1. Интеллектуальные системы и технологии. Учебник и практикум для СПО Станкевич Л. А. Научная школа: Санкт-Петербургский политехнический университет Петра Великого (г. Санкт-Петербург). Год: 2019 / Гриф УМО СПО https://biblio-online.ru/book/intellektualnye-sistemy-i-tehnologii-445852 2. Федорова Г.Н. Разработка и администрирование баз данных (2-е изд., стер.) учебник«Академия»2019 г. 3. Федорова Г.Н. Информационные системы (6-е изд., стер.) учебник «Академия» 2019г 4. Рудаков А.В. Технология разработки программных продуктов (12-е изд.) учебник«Академия»2018 г.
Тема № 61-62	Лекция на тему: Область видимости имен. Оператор определения контекста namespace.
Задание	Область видимости имен Между именами переменных, функций, типов и т.п. при использовании одного и того же имени в разных частях программы могут возникать конфликты. Для того чтобы эти конфликты можно было разрешать, в языке существует такое понятие как область видимости имени. Минимальной областью видимости имен является блок. Имена, определяемые в блоке, должны быть различны. При попытке объявить две переменные с одним и тем же именем произойдет ошибка. Имена, определенные в блоке, видимы (доступны) в этом блоке после описания и во всех вложенных блоках. Аргументы функции, описанные в ее заголовке, рассматриваются как определенные в теле этой функции. Имена, объявленные в классе, видимы внутри этого класса, т.е. во всех его методах. Для того чтобы обратиться к атрибуту класса, нужно использовать операции " .", " -> " или " ::". Для имен, объявленных вне блоков, областью видимости является весь текст файла, следующий за объявлением. Объявление может перекрывать такое же имя, объявленное во внешней области. int x = 7; class A { public: void foo(int y); int x; };

```
int main()
{
    A a;
    a.foo(x);
  // используется глобальная переменная x
  // и передается значение 7
    cout << x;
    return 1;
}
void
A::foo(int y)
{
    x = y + 1;
    {
        double x = 3.14;

        cout << x;
    }
    cout << x;
}   // x – атрибут объекта типа A
  // новая переменная x перекрывает атрибут класса x
```

В результате выполнения приведенной программы будет напечатано 3.14, 8 и 7.

Несмотря на то, что имя во внутренней области видимости перекрывает имя, объявленное во внешней области, перекрываемая переменная продолжает существовать. В некоторых случаях к ней можно обратиться, явно указав область видимости с помощью квалификатора " ::". Обозначение ::имя говорит о том, что имя относится к глобальной области видимости. (Попробуйте поставить :: перед переменной x в приведенном примере.) Два двоеточия часто употребляют перед именами стандартных функций библиотеки языка Си++, чтобы, во-первых, подчеркнуть, что это глобальные имена, и, во-вторых, избежать возможных конфликтов с именами методов класса, в котором они употребляются.

Если перед квалификатором поставить имя класса, то поиск имени будет производиться в указанном классе. Например, обозначение A::x показало бы, что речь идет об атрибуте класса A. Аналогично можно обращаться к атрибутам структур и объединений. Поскольку определения классов и структур могут быть вложенными, у имени может быть несколько квалификаторов:

```
class Example
{
public:
    enum Color { RED, WHITE, BLUE };
    struct Structure
    {
        static int Flag;
        int x;
    };
    int y;
```

```
    void Method();
};
```
Следующие обращения допустимы извне класса:
```
Example::BLUE
Example::Structure::Flag
```
При реализации метода Method обращения к тем же именам могут быть проще:
```
void
Example::Method()
{
    Color x = BLUE;
    y = Structure::Flag;
}
```
При попытке обратиться извне класса к атрибуту набора BLUE компилятор выдаст ошибку, поскольку имя BLUE определено только в контексте класса.

Отметим одну особенность типа enum. Его атрибуты как бы экспортируются во внешнюю область имен. Несмотря на наличие фигурных скобок, к атрибутам перечисленного типа Color не обязательно (хотя и не воспрещается) обращаться Color::BLUE.

Оператор определения контекста namespace

Несмотря на столь развитую систему областей видимости имен, иногда и ее недостаточно. В больших программах возможность возникновения конфликтов на глобальном уровне достаточно реальна. Имена всех классов верхнего уровня должны быть различны. Хорошо, если вся программа разрабатывается одним человеком. А если группой? Особенно при использовании готовых библиотек классов. Чтобы избежать конфликтов, обычно договариваются о системе имен классов. Договариваться о стиле имен всегда полезно, однако проблема остается, особенно в случае разработки классов, которыми будут пользоваться другие.

Одно из сравнительно поздних добавлений к языку Си++ – контексты, определяемые с помощью оператора namespace. Они позволяют заключить группу объявлений классов, переменных и функций в отдельный контекст со своим именем. Предположим, мы разработали набор классов для вычисления различных математических функций. Все эти классы, константы и функции можно заключить в контекст math для того, чтобы, разрабатывая программу, использующую наши классы, другой программист не должен был бы выбирать имена, обязательно отличные от тех, что мы использовали.
```
namespace math
{
    double const pi = 3.1415;
    double sqrt(double x);
    class Complex
    {
    public:
        . . .
    };
};
```
Теперь к константе pi следует обращаться math::pi.

	Контекст может содержать как объявления, так и определения переменных, функций и классов. Если функция или метод определяется вне контекста, ее имя должно быть полностью квалифицировано double math::sqrt(double x) { . . . } Контексты могут быть вложенными, соответственно, имя должно быть квалифицировано несколько раз: namespace first { int i; namespace second // первый контекст // второй контекст { int i; int whati() { return first::i; } // возвращается значение первого i int anotherwhat() { return i; } // возвращается значение второго i } } first::second::whati(); // вызов функции Если в каком-либо участке программы интенсивно используется определенный контекст, и все имена уникальны по отношению к нему, можно сократить полные имена, объявив контекст текущим с помощью оператора using. double x = pi; // ошибка, надо использовать math::pi using namespace math; double y = pi; // использовать контекст math // теперь правильно
Контрольный тест	Ответьте на вопросы: 1. Охарактеризуйте область видимости имен. 2. Что является минимальной областью видимости имен? 3. Что является для имен, объявленных вне блоков, областью видимости? 4. Охарактеризуйте особенность типа enum 5. Охарактеризуйте оператор определения контекста namespace.

Дата____10.11.2021 г _____ _____ _____

Подпись Ф.И.О. преподавателя