ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ОРЕНБУРГСКИЙ ГОСУДАРСТВЕННЫЙ АГРАРНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ДЛЯ ОБУЧАЮЩИХСЯ ПО ОСВОЕНИЮ ДИСЦИПЛИНЫ

Б1.Б.32 Программирование на языках высокого уровня

Направление подготовки *10.03.01 Информационная безопасность*

Профиль образовательной программы *Безопасность автоматизированных систем*

Форма обучения *очная*

СОДЕРЖАНИЕ

1. Конспект лекций ……..…	3
1.1. Лекция № 1 Общие принципы построения и использования языков высокого уровня	3
1.2. Лекция № 2 Структура программы на языке высокого уровня ……	3
1.3. Лекция № 3 Система программирования языка высокого уровня ……	4
1.4. Лекция № 4 Описание данных. Типы данных и переменные ……	4
1.5. Лекция № 5 Обработка данных. Операции ввода-вывода ……	10
1.6. Лекция № 6 Алгоритмические механизмы. Представления основных структур программирования ……	12
1.7. Лекция № 7 Модульные программы ……	13
1.8. Лекция № 8 Проектирование программ ……	15
1.9. Лекция № 9 Объекты и классы ……	16
1.10. Лекция № 10 Взаимодействие объектов в программе ……	17
2. Методические указания по проведению практических занятий……	21
2.1. **Практическое занятие № 1 (ПЗ-1).** Общие принципы построения и использования языков высокого уровня……	21
2.2. **Практическое занятие № 2 (ПЗ-2).** Структура программы на языке высокого уровня	21
2.3. **Практическое занятие №3, 4 (ПЗ-3, 4).** Система программирования языка высокого уровня……	22
2.4. **Практическое занятие №5 (ПЗ-5).** Описание данных. Типы данных и переменные…	22
2.5. **Практическое занятие №6 (ПЗ-6).** Обработка данных. Операции ввода-вывода……	23
2.6. **Практическое занятие №7, 8 (ПЗ-7, 8).** Алгоритмические механизмы. Представления основных структур программирования……	23
2.7. **Практическое занятие № 9, 10 (ПЗ-9,10).** Модульные программы……	24
2.8. **Практическое занятие № 11, 12 (ПЗ-11, 12).** Проектирование программ……	24
2.9. **Практическое занятие №13, 14, 15 (ПЗ-13, 14, 15).** Объекты и классы……	25
2.10. **Практическое занятие №16, 17 (ПЗ-16, 17).** Взаимодействие объектов в программе	25

1. КОНСПЕКТ ЛЕКЦИЙ

1.1. Лекция № 1 (1 час)

Тема: «Общие принципы построения и использования языков высокого уровня»

1.1.1. Вопросы лекции:

1) Виды языков программирования.

2) Алгоритмические языки.

1.1.2. Краткое содержание вопросов:

Алгоритм должен быть составлен таким образом, чтобы исполнитель, в расчете на которого он создан, мог однозначно и точно следовать командам алгоритма и эффективно получать определенный результат. Это накладывает на записи алгоритмов ряд обязательных требований, суть которых вытекает, вообще говоря, из приведенного выше неформального толкования понятия алгоритма. Сформулируем эти требования в виде перечня свойств, которым должны удовлетворять алгоритмы, адресуемые заданному исполнителю.

1. Одно из первых требований, которое предъявляется к алгоритму, состоит в том, что описываемый процесс должен быть разбит на последовательность отдельных шагов. Рассмотренное свойство алгоритмов называют дискретностью.

2. Используемые на практике алгоритмы составляются с ориентацией на определенного исполнителя. Чтобы составить для него алгоритм, нужно знать, какие команды этот исполнитель может понять и исполнить, а какие - не может. Это свойство алгоритмов будем называть понятностью.

3. Будучи понятным, алгоритм не должен содержать предписаний, смысл которых может восприниматься неоднозначно, т.е. одна и та же команда, будучи понятна разным исполнителям, после исполнения каждым из них должна давать одинаковый результат. Отмеченное свойства алгоритмов называют определенностью или детерминированностью.

4. Обязательное требование к алгоритмам -результативность. Смысл этого требования состоит в том, что при точном исполнении всех предписаний алгоритма процесс должен прекратиться за конечное число шагов и при этом должен получиться определенный результат. Вывод о том, что решения не существует - тоже результат.

5. Наиболее распространены алгоритмы, обеспечивающие решение не одной конкретной задачи, а некоторого класса задач данного типа. Это свойство алгоритма называют массовостью. В простейшем случае массовость обеспечивает возможность использования различных исходных данных.

1.2. Лекция № 2 (1 час)

Тема: «Структура программы на языке высокого уровня»

1.2.1. Вопросы лекции:

1) Главная функция.

2) Заголовочные файлы.

1.2.2. Краткое содержание вопросов:

Структура программы

Программа на языке С++ состоит из функций, описаний и директив препроцессора. Одна из функций должна иметь имя main. Выполнение программы начинается с первого оператора этой функции. Простейшее определение функции имеет следующий формат:

```
тип возвращаемого значения имя ([ параметры ]){
операторы, составляющие тело функции
}
```

Как правило, функция используется для вычисления какого-либо значения, поэтому перед именем функции указывается его тип. Ниже приведены самые необходимые сведения о функциях:

если функция не должна возвращать значение, указывается тип void:
тело функции является блоком и, следовательно, заключается в фигурные скобки;
функции не могут быть вложенными;
каждый оператор заканчивается точкой с запятой (кроме составного оператора).

1.3. Лекция № 3 (2 часа)

Тема: «Система программирования языка высокого уровня»

1.3.1. Вопросы лекции:

1) Понятие системы программирования.

2) Возможности среды программирования Microsoft Visual Studio.

1.3.2. Краткое содержание вопросов:

Система программирования (СП) — совокупность программных средств, облегчающих написание, отладку диалоговой программы и автоматизирующих её многоэтапное преобразование в исполняемую программу и загрузку в память для выполнения. Ныне СП трансформировались в интегрированные среды разработки программ (Integrated Development Environment, IDE), позволяющие визуально разрабатывать пользовательский интерфейс и организовывать связь с базами данных.

Создание сложного программного средства осуществляется в среде программной инженерии. Согласно ГОСТ Р ИСО/МЭК 14764-2002 она представляет собой «набор автоматических инструментальных средств, программно-аппаратных и технических средств, необходимых для выполнения объёма работ по программной инженерии». К автоматизированным инструментальным средствам относятся, в частности, компиляторы, компоновщики загрузочных операционных систем, отладчики, средства моделирования, средства документирования и системы управления базами данных.

Система программирования освобождает проблемного пользователя или прикладного программиста от необходимости написания программ решения своих задач на неудобном для него языке машинных команд и предоставляют им возможность использовать специальные языки более высокого уровня. Для каждого из таких языков, называемых входными или исходными, система программирования имеет программу, осуществляющую автоматический перевод (трансляцию) текстов программы с входного языка на язык машины. Обычно система программирования содержит описания применяемых языков программирования, программы-трансляторы с этих языков, а также развитую библиотеку стандартных подпрограмм. Важно различать язык программирования и реализацию языка.

Язык – это набор правил, определяющих систему записей, составляющих программу, синтаксис и семантику используемых грамматических конструкций.

Реализация языка – это системная программа, которая переводит (преобразует) записи на языке высокого уровня в последовательность машинных команд.

По набору входных языков различают системы программирования одно- и многоязыковые. Отличительная черта многоязыковых систем состоит в том, что отдельные части программы можно составлять на разных языках и помощью специальных обрабатывающих программ объединять их в готовую для исполнения на ЭВМ программу.

Для построения языков программирования используется совокупность общепринятых символов и правил, позволяющих описывать алгоритмы решаемых задач и однозначно истолковывать смысл созданного написания. Основной тенденцией в развитии языков программирования является повышение их семантического уровня с целью облегчения процесса разработки программ и увеличения производительности труда их составителей.

По структуре, уровню формализации входного языка и целевому назначению различают системы программирования машинно-ориентированные и машинно-независимые.

1.4. Лекция № 4 (1 час)

Тема: «Описание данных. Типы данных и переменные».

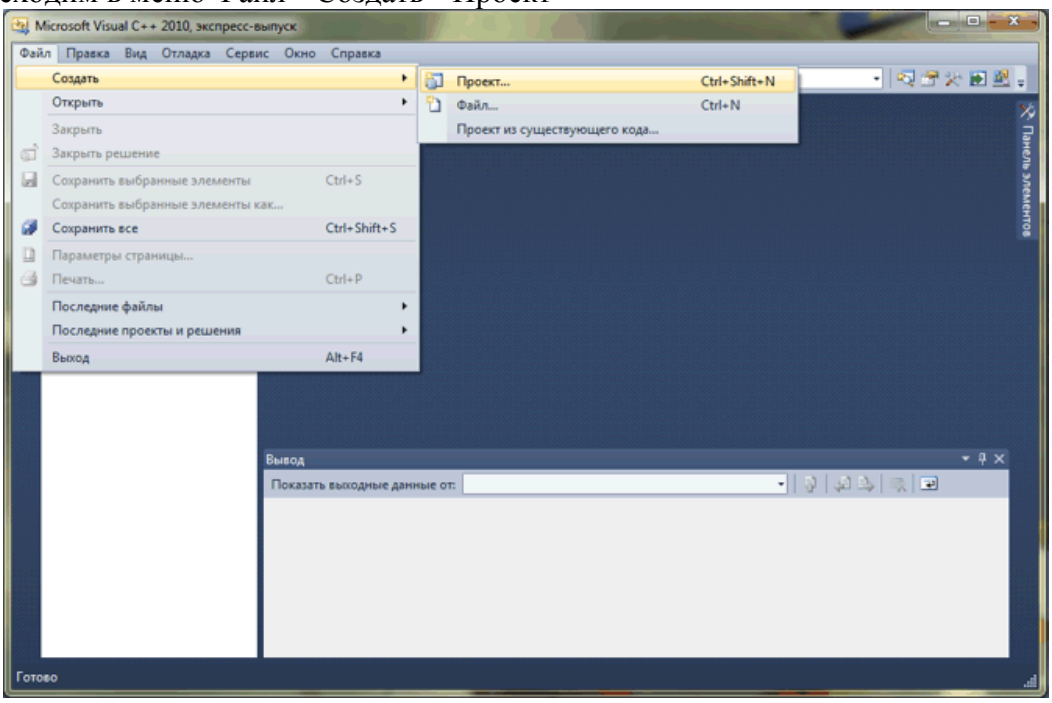
1.4.1. Вопросы лекции:

1) Понятие консольного приложения.
2) Возможности консольного приложения.
3) Стандартный заголовок.

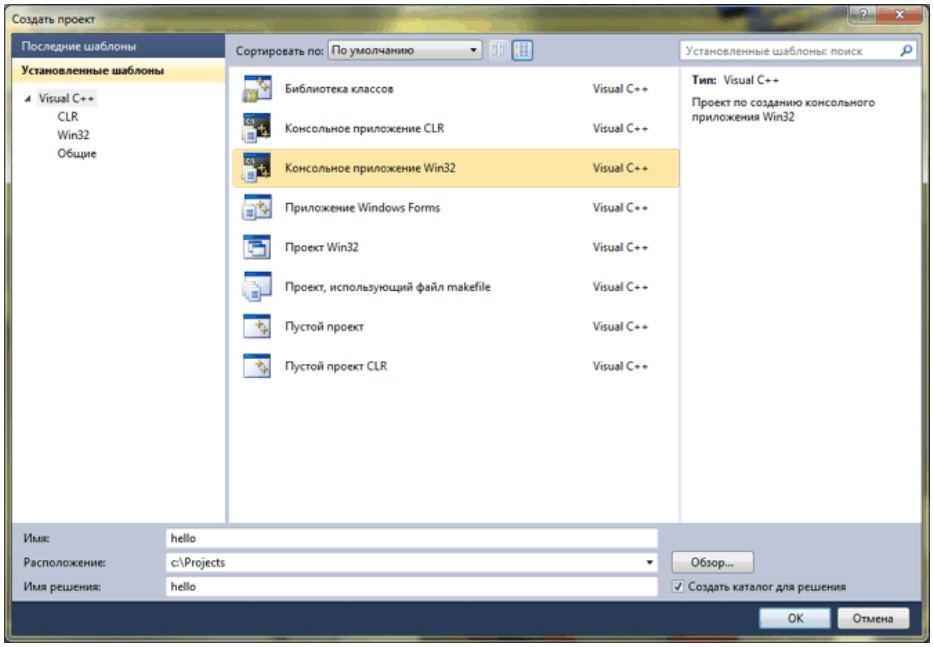
1.4.2. Краткое содержание вопросов:

Для создания Windows-приложений на C++ можно использовать среду разработки Microsoft Visual Studio 2010 Express с пакетом обновления SP1.

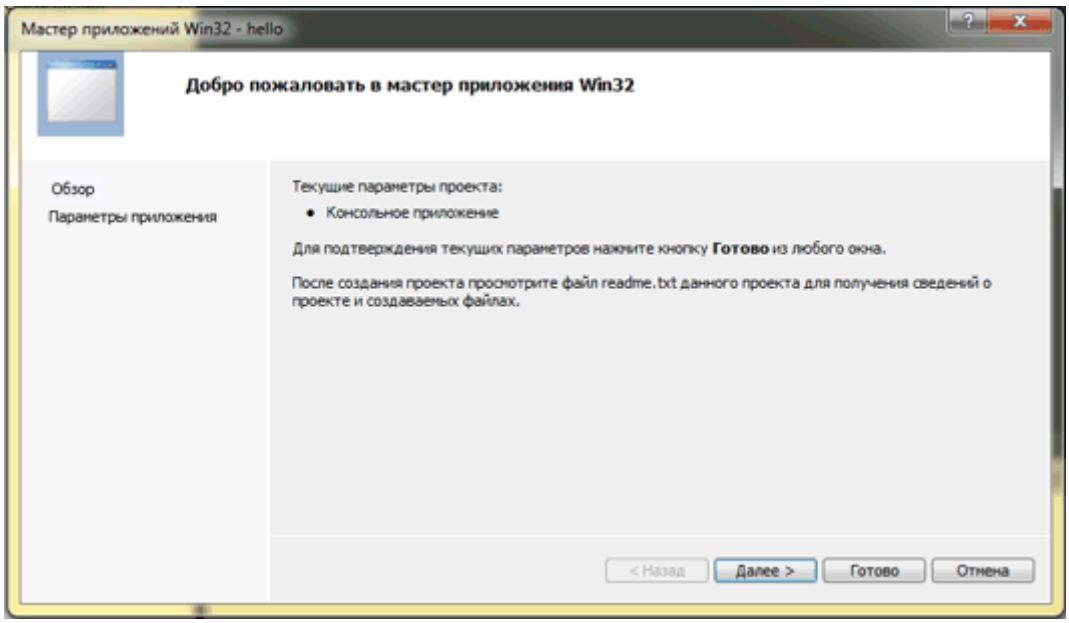
Для создания нового консольного приложения запускаем Microsoft Visual Studio 2010 Express и переходим в меню Файл->Создать->Проект



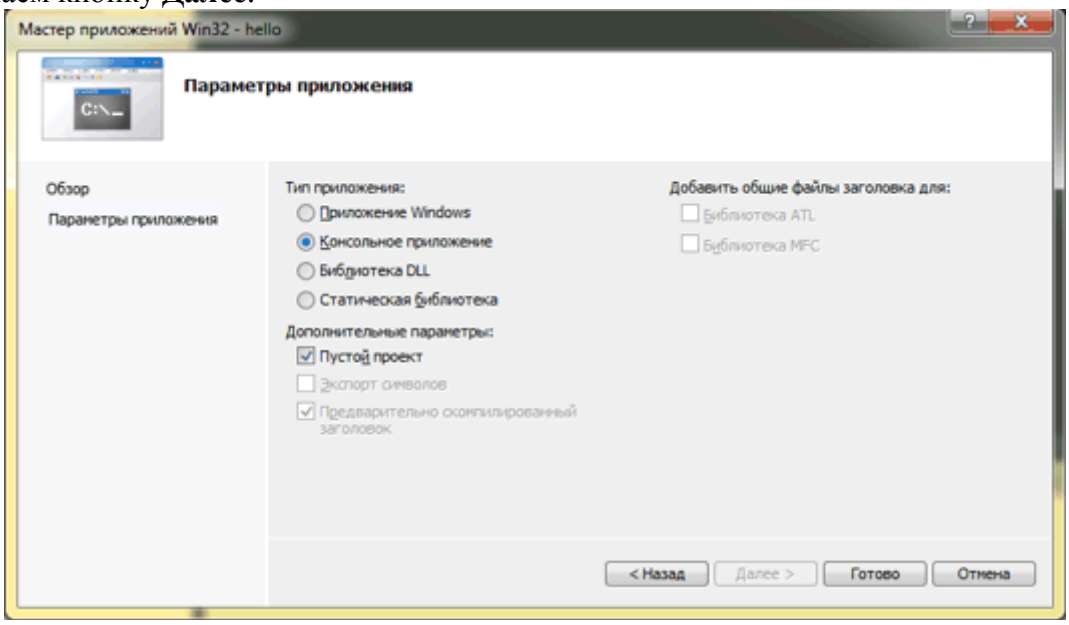
В появившемся окне выбираем Консольное приложение Win32 и задаем имя проекта и нажимаем кнопку OK.



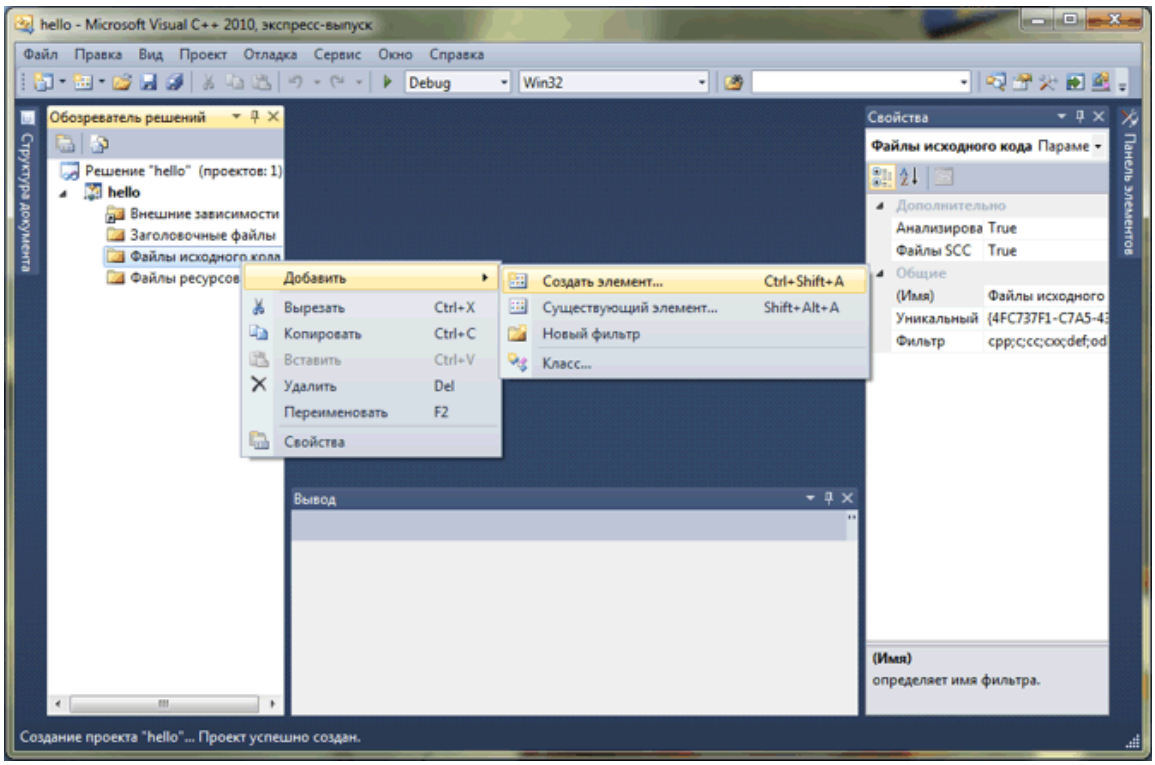
В появившемся окне нажимаем кнопку Далее.



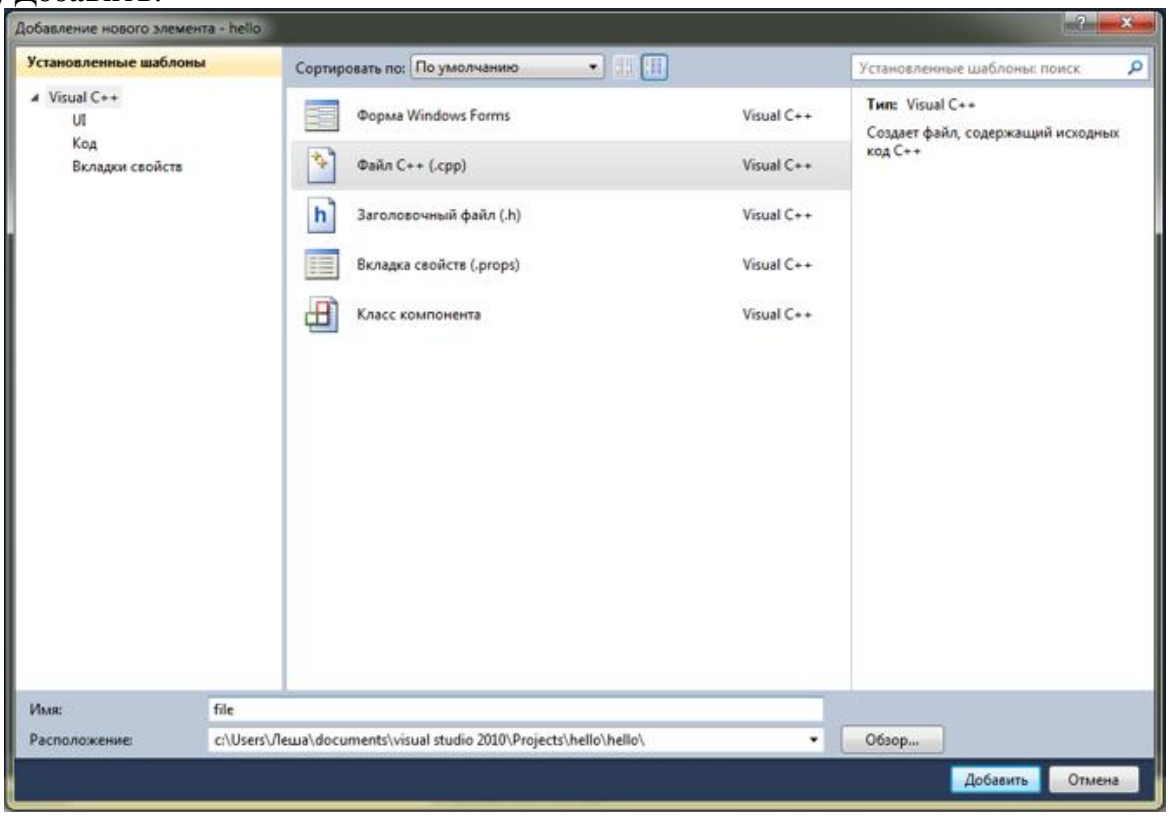
В следующем окне отмечаем галочку **Дополнительные параметры: Пустой проект** и нажимаем кнопку **Далее**.



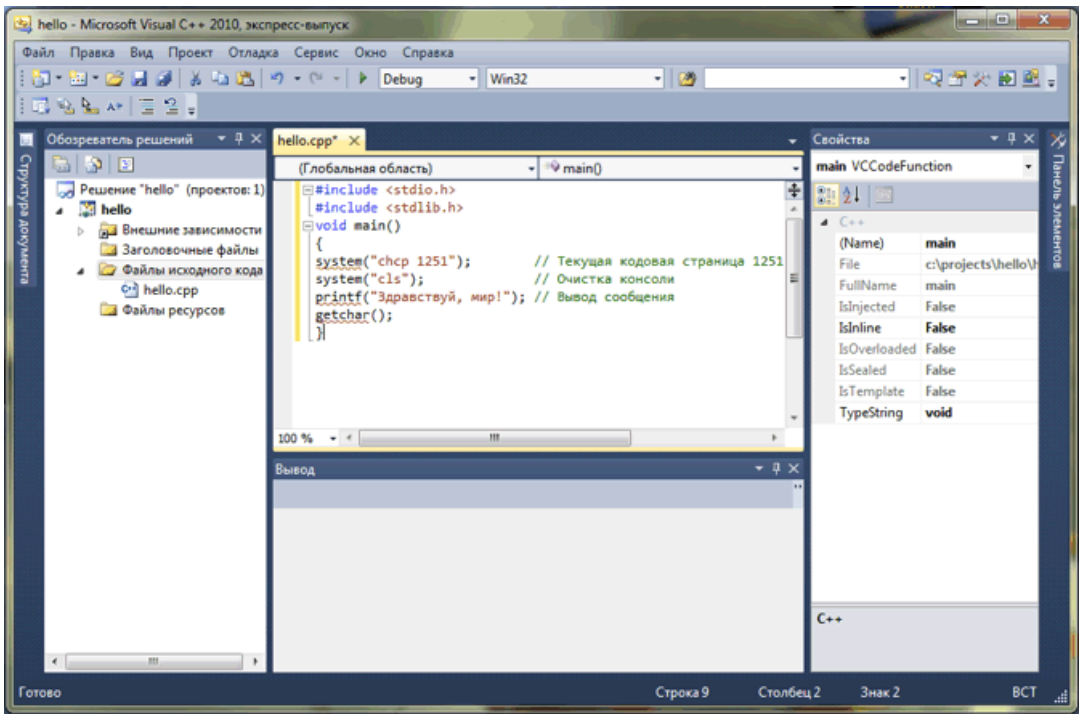
В левой части появившегося окна отображается **Обозреватель решений**. Для добавления нового файла программы в проект выбираем по правой кнопке мыши на папке **Файлы исходного кода**меню **Добавить->Создать элемент**.



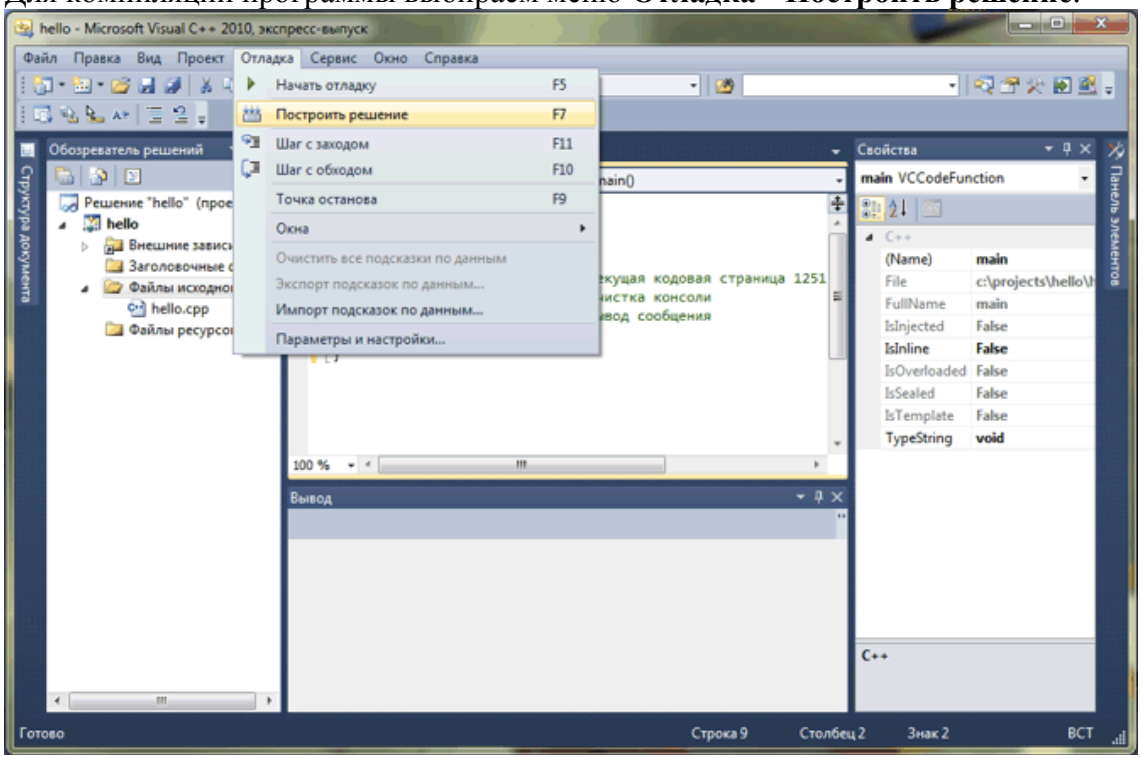
В появившемся окне выбираем **Файл C++ (.cpp)**, задаем имя файла и нажимаем кнопку **Добавить**.



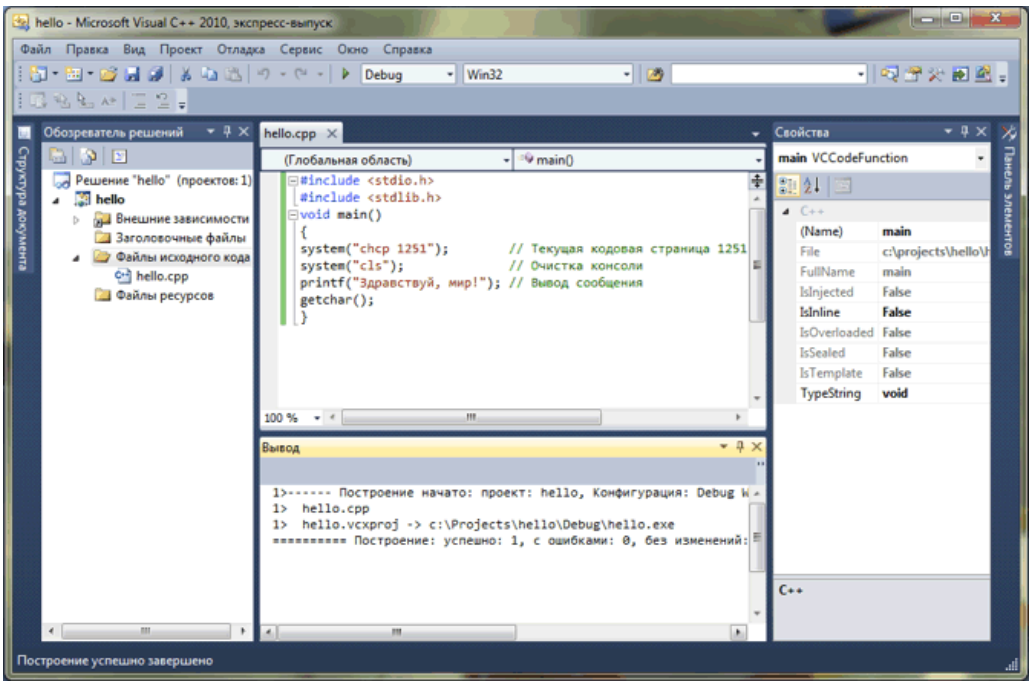
В появившемся окне набираем текст программы. В качестве примера можно использовать текст программы "Здравствуй, мир!" из раздела Структура программы на языке Си



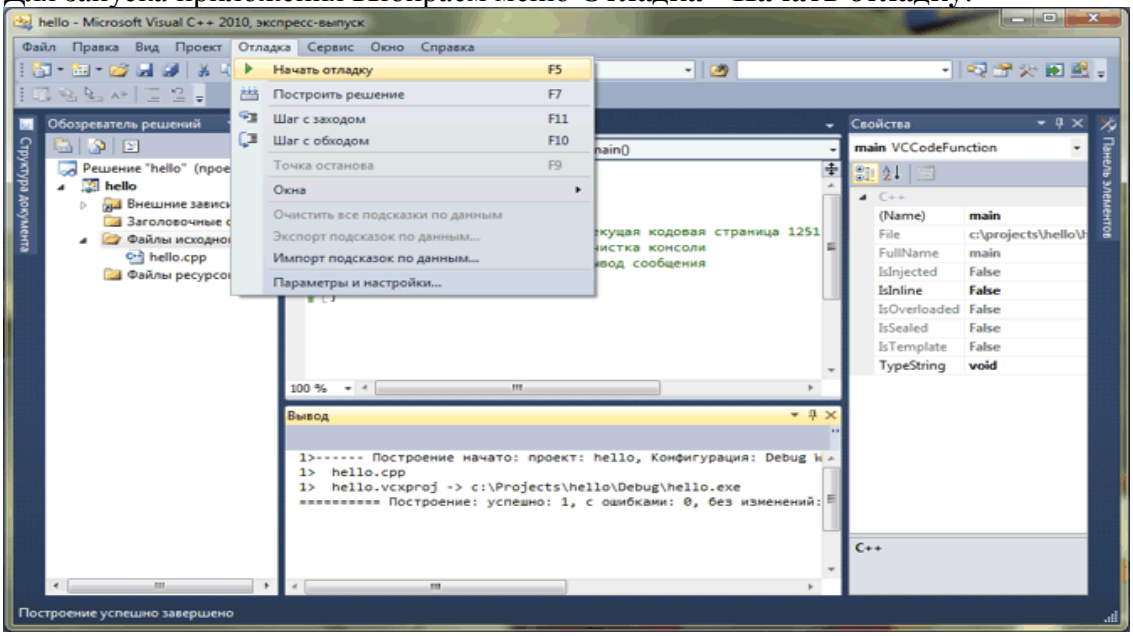
Для компиляции программы выбираем меню **Отладка->Построить решение**.



В случае успешного построения в нижней части окна отображается **Построение: успешно 1**.



Для запуска приложения выбираем меню **Отладка->Начать отладку**.



Результат выполнения программы:

1.5. Лекция № 5 (1 час)

Тема: «Обработка данных. Операции ввода-вывода».

1.5.1. Вопросы лекции:

1) Оператор присваивания.
2) Оператор условного перехода.
3) Логические операторы. Таблицы истинности.
4) Цикл с параметрами.
5) Цикл с предыдущим условием.
6) Цикл с последующим условием.

1.5.2. Краткое содержание вопросов:

естественно, что основными механизмами любого ЯП являются механизмы управления данными и процессом выполнения операторов. Рассмотрим последние.

Давно доказано, что для реализации алгоритма любой сложности достаточно совсем небольшого набора средств (к примеру условного оператора и оператора перехода или оператора цикла с условием и оператора переходов). Однако, ограничить язык наличием какого бы то ни было "алгоритмического минимума" нецелесообразно с точки зрения эффективности реализации большинства алгоритмов (каждый может проверить это, попытавшись нарисовать блок-схему некоторого циклического алгоритма с использованием только условных вершин для ветвления, даже не пытаясь реализовать его на каком либо ЯП). Даже максимально редуцированные языки - ассемблеры - обладают помимо условных операторов операторами параметрических циклов. Поэтому в большинство ЯП включается примерно следующий джентльменский набор операторов изменения порядка следования операторов. Каждый оператор будем сопровождать примером из какого-либо языка.

Оператор условия

Необходимость его присутствия в языке очевидна. Предоставляет возможность выполнить то или иное действие в зависимости от истинности некоторого логического условия (ЛУ). Практически во всех языках называется if.

Условный оператор практически всегда имеет две формы - усеченную, когда указывается лишь действие, выполняемое при истинности ЛУ, и полную когда указываются действия выполняемые при обоих значениях ЛУ. Типичный пример

```
if (a=4) a=a+1;
else a=b-2; //C++
```

Оператор множественного выбора

Некоторые алгоритмы требуют выполнения более двух различных действий в зависимости от некоторого условия (всегда не логического; к примеру от значения некоторой переменной), то есть ветвления более чем на 2 ветки. Реализация такого ветвления с помощью условного оператора нецелесообразна (кто не верит - пусть нарисует). Как следствие в большинство ЯП включен оператор множественного выбора.

Суть его, как правило, в следующем. Некоторое выражение (как правило это просто переменная), которое к моменту выполнения оператора может принимать несколько различных значений, сравнивается с одним из заранее определенных программистом значений, и в случае совпадения выполняется соответствующий оператор.

```
case c of
3: c: =8;
4: c:= b-3;
else b:= 4;
end; {Pascal}
```

Параметрический цикл

Часто некоторые действия необходимо повторять более одного раза, причем количество повторений можно заранее вычислить. К примеру - обработать последовательно элементы некоторого массива. Для эффективной реализации такого рода алгоритмов в ЯП включают циклы с параметром.

Наиболее часто такие циклы имеют следующий смысл. Некоторой переменной, называемой параметром присваивается некоторое начальное значение. Затем выполняются повторяемые операторы, называемые телом цикла, после чего параметр изменяется на некоторое значение, называемое шагом. Новое значение параметра сравнивается с конечным значением, и, если значение параметра не превысило конечное значение, все (выполнение тела цикла и изменение параметра) повторяется; в противном случае цикл считается завершенным.

```
Do 100 I=3,8,1
*операторы тела цикла
100 Continue
* Fortran IV
```

Операторы цикла с условием

Однако не всякий алгоритм позволяет заранее предсказать количество повторений тех или иных действий: довольно часто некоторые действия следует повторять вплоть до достижения некоторой цели (читай - выполнения некоторого условия). Соответственно, для предоставления программисту возможности простой и эффективной реализации такого рода алгоритмов в ЯП высокого уровня включают циклы с условием окончания.

Суть их в следующем. Повторяемые операторы, называемые все так же телом цикла, повторяются до тех пор пока выполняется некоторое логическое условие.

Здесь следует отметить, что в зависимости от того, когда производится проверка ЛУ - до или после выполнения тела цикла - различают циклы с предпроверкой и постпроверкой условия. Небольшие размышления позволяют сделать вывод о некоторых свойствах такого рода циклов.

цикл с предпроверкой условия может ни разу не выполниться, в том случае, если условие продолжения цикла не выполняется даже при первой проверке;

цикл с постпроверкой выполнится как минимум один раз вне зависимости от значения условия продолжения.

Следует отметить, что всегда существует возможность сэмулировать один цикл с условием с помощью условного оператора и другого цикла с условием. Однако это почти всегда неэффективно, а значит каждый уважающий себя программист должен одинаково свободно владеть обоими и применять каждый в подходящем для него случае.

```
Do While Not EOF()
&& Тело цикла с предпроверкой
EndDo && FoxPro
```

Repeat
{Тело цикла с постпроверкой}
Until i=4; {Pascal}

Оператор безусловного перехода

И, наконец, практически ни один из ЯП, за исключением, пожалуй, только FoxPro, не обходится без оператора безусловного перехода, полезность которого ставится под сомнение большинством программистов. Суть его общеизвестна и заключается в следующем: как только данный оператор встречается в программе, производится переход по адресу в нем указанному. Этот адрес как правило называется меткой и представляет из себя либо введенную программистом лексему, либо номер строки, как, например, в следующем примере

157 GoTo 342 (BASIC)

Собственно, этим набором и ограничивается весь список операторов, реализующих алгоритмические механизмы в алгоритмических ЯП высокого уровня.

1.6. Лекция № 6 (2 часа)

Тема: «Алгоритмические механизмы. Представления основных структур программирования»

1.6.1. Вопросы лекции:

1) Тип данных массив.
2) Тип данных строка.
3) Тип данных структура.

1.6.2. Краткое содержание вопросов:

Массив - это упорядоченное, ограниченное множество однотипных элементов, объединенных общим именем. Тип компонент (элементов) называется базовым и м.б. любым кроме файлового. Для обозначения элементов массива используется переменная с индексами.

Например: A[25], B[5, 8], X[i], Y[i, j], R[n+1].

Размер массива (длина) – число элементов массива.

Размерность массива – число индексов в массиве.

Индекс определяет положение элемента в массиве. В качестве индекса м.б. использовано выражение (индексное), переменная или константа (частный случай выражения). В качестве индексов м.б. любой порядковый тип кроме Longint. Тип индексов д.б. скалярным, упорядоченным и конечным. Чаще всего используются индексы интервального типа, при этом они м.б. отрицательными.

Все компоненты массива одинаково доступны и могут выбираться в произвольном порядке.

В программе массивы д.б. описаны одним из двух способов:

С использованием раздела описания типов:

```
Type T = array[T1] of T2;
Var  A, B, C        :T;
```

где:

T - имя типа; {регулярный тип}

T1 - тип индекса;
T2 - тип компонент. {базовый тип}

Тип, определенный с помощью конструкции array . . of называется регулярным.

Например:

```
Type
    vec_r = array[1. .n] of real;
    vec1_i = array[1. .n] of integer;
    vec1_l = array[False. .True] of Boolean;
    vec1_S = array['A'1. . 'Z'] of char;
Var
    A, B      :vec_r;
    N, K      :vec_i;
    V, W      :vec_l;
    R, S      :vec_s;
```

Без использования раздела описания типов:

```
Var A, B, C : = array[T1] of T2;
```

Например:

```
Var
    A, B      : array[1. .n] of real;
    N, K      : array[1. .n] of integer;
    V, W      : array[False. .True] of Boolean;
    R, S      : array['A'1. . 'Z'] of char;
```

Первый способ является более предпочтительным, хотя он и удлиняет размер описаний, но программа становится более ясной логически. Использование раздела Type свидетельствует о хорошем стиле программирования

К компонентам массива применимы операции, процедуры и функции, допустимые для данного базового типа.

Все действия с массивами выполняются в цикле, за исключением присваивания целиком элементов одного массива, другому.

1.7. Лекция № 7 (2 часа)

Тема: «Модульные программы».

1.7.1. Вопросы лекции:

1) Понятие динамической переменной.

2) Указатели.

3) Динамические массивы.

1.7.2. Краткое содержание вопросов:

При написании программ часто возникает необходимость динамического выделения и освобождения памяти на этапе выполнения программы. Это позволяет использовать освобожденное пространство для других целей.

Для динамических переменных характерны следующие особенности:

1) у них нет имен, поэтому вся работа с ними осуществляется только с помощью указателя;

2) в отличие от остальных переменных, которые создаются и уничтожаются автоматически, динамические переменные создаются с помощью операции new на этапе выполнения программы и должны быть уничтожены операцией delete до окончания работы программы, иначе память компьютера останется занятой;

3) поскольку размер динамической памяти ограничен, то запрос на выделение памяти не всегда удовлетворяется и программист обязан за этим следить.

Оператор new выделяет память требуемого размера и возвращает адрес начала выделенного участка памяти, который присваивается указателю соответствующего типа. Если память не может быть выделена, то возвращается нулевой адрес. Поэтому перед началом работы с динамической памятью проверяется значение указателя. Если в нем хранится нулевой адрес, то работа соответствующей функции в программе завершается.

Пример создания и работы с динамической переменной:

```
double *Ptr; // объявление указателя
Ptr = new double; // выделение памяти и инициализация указателя
if (Ptr == NULL) // если память не выделена

{

printf("\n Ошибка выделения памяти!");

getch();

return; // выход из функции, которая

} // использует динамическую память

*Ptr = 1.23456; // работа с динамической переменной
```

По окончании работы с динамической переменной она должна быть удалена с помощью операции delete.

```
delete Ptr; // освободить память, адрес которой хранится в указателе Ptr
```

Удаление динамической переменной не означает, что память очищается, это означает только то, что ранее занятая память становится доступной для повторного выделения.

В операторе new можно использовать как стандартные типы, так и типы данных, определенные пользователем. Динамически выделять память для переменных стандартных типов невыгодно, т.к. память расходуется не только на переменные, но и на информацию о выделении памяти. Поэтому она выделяется только на большие массивы стандартных или пользовательских типов, размер которых заранее неизвестен.

Динамический одномерный массив создается следующим образом:

```
double *Ptr; // объявление указателя

int N;

printf( "\n Введите размер массива ");

scanf("%d", &N);



Ptr = new double [N]; // создание динамического массива
```

```
if (Ptr == NULL)

{

printf( “\n Ошибка выделения памяти!”);

getch();

return;

}
for (int i = 0; i < N; i++)
Ptr[ i ] = i * 5; // работа с динамическим массивом
```

Для удаления динамического массива необходимо записать:
delete [ ] Ptr;
однако, возможен и такой вариант удаления:
delete Ptr;
т.к. информация о размере выделенной памяти известна.

1.8. Лекция № 8 (2 часа)
Тема: «Проектирование программ».
1.8.1. Вопросы лекции:
1) Определение объекта.
2) Определение класса.
3) Свойства объекта.
1.8.2. Краткое содержание вопросов:

Технологическое определение класса. Технология ООП, прежде всего, накладывает ограничения на способы представления данных в программе и их взаимодействие с алгоритмической компонентой (функциями). Любая программа отображает в своих данных состояние **внешних объектов программирования.** Это могут быть как физические объекты внешней среды, так и логические программные сущности (например, файлы). Для этого можно использовать различные способы, можно «размазать» свойства объекта по различным структурам данных. Можно исходить из того, что каждому объекту будет соответствовать своя собственная структура данных, в которой содержатся все элементы описания свойств внешнего объекта программирования. Такую структуру данных можно аналогично назвать **объектом.** Функции, работающие с объектом и получающие в качестве обязательного параметра указатель на структуру данных, называются **методами.** Совокупность описания объектов одного типа и методов работы с ними называется **классом.**

Объект – структура данных, содержащая описание свойств внешнего объекта программирования. **Метод** – функция, работающая с объектом. **Класс** – описание структуры объекта и методов работы с ним.

Строго говоря, реализовать идеи ООП можно в классической среде программирования, соблюдая дух, а не букву технологии. Например, **библиотека функций,** работающая на общую структуру данных, может в первом приближении считаться классом.

Синтаксическое определение объекта и класса - структура со встроенными функциями. Структурированная переменная со встроенными в нее функциями (см. **5.3)** идеально подходит под определение объекта. Не нужно пытаться изобретать велосипед, достаточно назвать вещи своими именами. Структурированная переменная – это объект,

функции, встроенные в структурированный тип – это методы класса, а само определение структурированного типа (включающее и определение встроенных функций) – есть класс.

Синтаксическое определение класса – тип данных, определяемый программистом. Класс представляет собой описание структуры объектов одного вида с набором методов их обработки. Аналогия с типом данных здесь напрашивается сама собой. Тип данных – это форма представления данных с набором операций. Отличие состоит в том, что тип данных либо уже определен в языке, либо формально составляется из уже существующих (но без внутреннего программирования). Значит, класс можно определить как тип данных, определяемый программистом. Тогда объект – это переменная класса. Эти трактовка закреплена в языке: синтаксис определения переменных и объектов почти полностью идентичен.

Класс – тип данных, определяемый программистом, **объект –**переменная класса

Прописные истины объектно-ориентированного подхода

Объектно-ориентированный подход не ограничен синтаксисом. Следует соблюдать не только букву, но и дух ООП. Но даже в самой реализации понятий класса и объекта в языке программирования имеется много очевидных, но не всегда упоминаемых вещей, которые следует помнить. Попробуем их здесь перечислить.

Контекст класса (текущего объекта). Понятие контекста мы уже вводили применительно к функции (**1.6**). Контекст (окружение) – это набор имен (объектов языка), которые можно использовать непосредственно, без указания «пути доступа». Контекстом функции являются ее формальные параметры и локальные переменные. Каждый класс определяет свой контекст – это имена элементов данных и методов класса. Этот контекст возникает, поскольку программные компоненты класса имеют дело с текущим объектом, умолчание касается именно его. Любой элемент контекста может адресоваться как просто по имени, так и через указатель **this,** например **n** и **this->n.**

Классы и порождение объектов. Класс – это описание объектов, объект – это инсталляция (отображение) класса в памяти. Можно, конечно, это понимать буквально, как создание экземпляра класса со всеми его «потрохами» - данными и методами. В реальности же проекция класса на традиционную компьютерную архитектуру выглядит таким образом:

для каждого объекта создается экземпляр данных;

методы класса, с которыми работает объект, представляют собой единственный экземпляр программного кода в сегменте команд, который одинаково выполняется для всех объектов (разделяется ими);

при вызове метода объект, для которого он выполняется, идентифицируется указателем текущего объекта **this,** задающим контекст текущего объекта.

1.9. Лекция № 9 (2 часа)

Тема: «Объекты и классы».

1.9.1. Вопросы лекции:

1) Понятие иерархии объектов.

2) Понятие наследования.

3) Права доступа при наследовании.

1.9.2. Краткое содержание вопросов:

Иерархия объектов

До сих пор мы рассматривали в качестве содержимого объектов только переменные. При использовании на их месте объектов возникает иерархия вложенности. В ней самой нет ничего

необычного, вложенные объекты можно применять как обычные данные, вызывать для них собственные методы и т.д..

Однако технология ООП требует «распространения» по иерархии объектов сверху вниз ряда специфичных для нее действий над объектами, таких как конструирование (в т.ч. конструктор копирования), присваивание, ввод и вывод в стандартные потоки. За это отвечает внешний класс, включающий в себя объекты: при выполнении в нем перечисленных действий он должен инициировать аналогичные действия в объектах, определенных внутри него. Рассмотрим эту проблему с общих позиций.

Класс user в качестве элементов данных включает в себя объекты классов string и date. Если объект не содержит других объектов и динамических данных (класс date), то переопределение присваивания и конструктора копирования не требуется. Для класса string, содержащего внешний динамический массив символов, требуется корректное определение конструкторов (пустого и копирования), деструктора и присваивания.

```
class string{      char *str;
public:   string(){ str=strdup("");}
          string(char *s){ str=strdup(s); }
          string(string &T){ str=strdup(T.str); }        // КК
          ~string(){ delete []str; }
          string &operator=(string &T){                  // Присваивание
                    delete []str;                        // разрушить старое
                    str=strdup(T.str);                   // копия ДМ источника
                    return *this; }                      // возврат ссылки на текущий
          friend ostream&operator<<(ostream &O, string &D){
                    O << D.str << endl; return O; }    // вывод в поток
};
class date{ int dd,mm,yy;
public:   date(int d=1,int m=1,int y=2000) { dd=d; mm=m; yy=y; }
          // КК и = переопределять не требуется
          friend ostream&operator<<(ostream &O, date &D){
                    O << D.dd << " " << D.mm << " " << D.yy << endl; return O; }
};
```

В классе user первый вопрос возникает в связи с конструированием. Здравый смысл подсказывает, что если в конструкторе внешнего класса не содержится информации о конструировании вложенных объектов, то по умолчанию для них возможен только вызов их собственных конструкторов без параметров, причем перед вызовом конструктора нового класса. Поскольку он во время своей работы должен получить внутренние объекты «в готовом виде». Из тех же соображений транслятор вызывает деструкторы вложенных объектов после вызова деструктора для основного класса.

Если все-таки требуется использовать конструкторы внутренних объектов с параметрами, то в заголовке конструктора внешнего класса их необходимо явно перечислить. Их параметры могут быть любыми выражениями, включающими формальные параметры конструктора нового класса. Имена конструкторов в этом варианте совпадают с именами внутренних объектов.

1.10. Лекция № 10 (2 часа)

Тема: «Взаимодействие объектов в программе».

1.10.1. Вопросы лекции:

1) Технология структурного программирования.

2) Особенности функции как модуля.

3) Модульное проектирование.

1.10.2. Краткое содержание вопросов:

Модульная технология обучения обретала статус самостоятельной дидактической системы постепенно, опираясь на ряд сущностных моментов программированного обучения: индивидуализированный темп учебно-познавательной деятельности, постоянное подкрепление

обучающимся собственных действий по самоконтролю, последовательность и логичность этих действий.

В модульном обучении (МО) интегрированы теоретико-практические наработки и обобщения проблемного обучения, принципов индивидуализации и дифференциации обучения. Особенности рефлексивного подхода во многом способствовали созданию основ МО, определения принципов и правил его построения, методов и форм его реализации.

Модульные технологии, дидактические системы, отдельные курсы на основе принципов модульного обучения, созданы и функционируют во многих колледжах и гимназиях, университетах США и Западной Европы. Они получают распространение в Казахстане: в общеобразовательной школе, в системах начального, среднего и высшего профессионального образования, внедряются в образовательные системы обучения взрослых – при подготовке и переподготовке специалистов и при повышении квалификации, [1].

Модуль – от латинского слова «modulies» – «мера», «способ». Разработчиками проблем модульной технологии подчеркивается соотнесение его дидактического определения с пониманием модуля в точных науках, в технике: это – некая целостная функциональная система, ограниченная определенными рамками, которая обеспечивает выполнение какой-то конкретной функции от начала до конца. То есть – это функционально и конструктивно независимая единица, которая может быть относительно самостоятельной частью – объектом в составе другого более сложного объекта или в виде индивидуального изделия, агрегата, объекта.

Модуль – это целевой функциональный узел, в котором учебное содержание, технология овладения им система контроля и коррекции объединены в систему высокого уровня целостности. Исследователи утверждают, что модуль можно рассматривать как программу обучения, индивидуализированную по содержанию, методам обучения, уровню самостоятельности, темпу учебно-познавательной деятельности обучающихся. Каждый модуль имеет свою дидактическую цель. Ей должна соответствовать достаточная полнота учебного материала. Это означает:

- в модуле излагается принципиально важное содержание учебной информации;
- дается разъяснение к этой информации;
- определяются условия погружения в информацию (с помощью средств ТСО, конкретных литературных источников, методов добывания информации);
- приводятся теоретические задания и рекомендации к ним;
- указаны практические задания;
- дается система самостоятельного и внешнего контроля.

Модульную технологию можно использовать в любой системе обучения, в том числе в экстернате: четкое дозирование учебного материала, информационно-методическое обеспечение с программой логически последовательных действий для обучающегося, возможность осваивать материал в удобное для него время, – все это помогает улучшить качество и эффективность образовательного процесса в целом.

Основным средством модульной технологии, кроме модуля как части программного материала учебной дисциплины, является сформированная на основе модулей модульная программа.

Модульная программа – это система средств, приемов, с помощью и посредством которых достигается интегрирующая дидактическая цель в совокупности всех модулей конкретной учебной дисциплины. Она разрабатывается преподавателем на основе определения основных идей курса. Каждой такой идее соответствует разработанный преподавателем модуль. Их совокупность обеспечивает реализацию основной цели изучения всей учебной дисциплины. Исследователи рекомендуют начинать каждый модуль: 1) с входного контроля знаний и умений (для определения уровня готовности обучаемых к предстоящей самостоятельной работе); 2) с выдачи индивидуального задания, основанного на таком анализе. Заданием может быть: например, реферат по результатам анализа знаний, расчетно-графические задания, контрольная работа, тесты, письменные опросы и т.п. Модуль всегда должен заканчиваться контрольной

проверкой знаний. Контролем промежуточным и выходным проверяется уровень усвоения знаний и выработки умений в рамках одного модуля или нескольких модулей. Затем – соответствующая доработка, корректировка, установка на следующий «виток», т.е. последующий модуль.

Ценность модульной системы обучения в том, что она, воспитывая умение самостоятельно учиться, развивает рефлексивные способности. Существенно, что при модульной системе, когда учебная деятельность структурируется на: учебные ситуации, контроль и оценку, актуализируются аналитические, исследовательские умения специалистов.

2. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ПРОВЕДЕНИЮ ПРАКТИЧЕСКИХ (СЕМИНАРСКИХ) ЗАНЯТИЙ

2.1. Практическое занятие №1 (2 часа).

Тема: «Общие принципы построения и использования языков высокого уровня».

2.1.1. Вопросы к занятию:

1) Понятия оператора, переменной, подпрограммы.

2.1.2. Краткое описание проводимого занятия:

2.1.2.1. Ответы на вопросы семинарского (практического) занятия.

2.1.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

1. Какие языки называют языками программирования высокого уровня
+ a) имитирующие естественные языки
   b) имитирующие машинный код
   c) имитирующие английский язык
   d) имитирующие высокоуровневый язык
2. Что такое оператор
+ a) фраза языка, однозначно определяющая трактуемый этап обработки данных
   b) ключевое слово языка, однозначно определяющее трактуемый этап обработки данных
   c) подпрограмма, однозначно определяющая трактуемый этап обработки данных
   d) имя переменной или идентификатор переменной
3. Что означает идентификатор языка программирования
+ a) имя программного объекта
   b) порядковый номер программного объекта
   c) принцип работы программного объекта
   d) принцип работы функции

2.2. Практическое занятие № 2 (2 часа).

Тема: «Структура программы на языке высокого уровня».

2.2.1. Вопросы к занятию:

1) Главная функция.

2) Заголовочные файлы.

2.2.2. Краткое описание проводимого занятия:

2.2.2.1. Ответы на вопросы семинарского (практического) занятия.

2.2.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

1. Языки высокого уровня делятся на:
+1) процедурные, логические и объектно-ориентированные;
  2) машинно-зависимые и машинно-независимые;
  3) проблемно-ориентированные и универсальные;
  4) алгоритмические и неалгоритмические.
2. Языками объектно-ориентированного программирования являются:
  1) Ada;
+2) C++;
  3) Fortran;
+4) Object Pascal;
  5) HTML;
  6) Pascal.
3. Системами программирования являются:
  1) MS DOS;

2) Adobe PhotoShop;
+3) Borland Delphi;
4) Turbo Pascal;
+5) Visual C++;
+6) Java.

2.3. Практическое занятие № 3, 4 (4 часа).
Тема: «Система программирования языка высокого уровня».
2.3.1. Вопросы к занятию:
1) Создание проекта.
2) Обозреватель решений.
2.3.2. Краткое описание проводимого занятия:
2.3.2.1. Ответы на вопросы семинарского (практического) занятия.
2.3.2.2. Проведение текущего контроля успеваемости
Задания для проведения текущего контроля успеваемости
1. Укажите рекомендации, относящиеся к рациональному стилю программирования.
+1) логически завершенные последовательности операторов целесообразно оформлять в виде подпрограмм;
+2) не следует создавать большие текстовые файлы;
3) метки лучше записывать цифрами;
4) программные модули следует создавать объемными.
2. Эффективная работа пользователя с диалоговой программой в основном достигается
+1) подготовкой пользователя для работы в диалоге;
2) знанием программистом специфики профессиональной деятельности пользователя;
3) знанием пользователем языка программирования;
4) наличием исчерпывающего комментария в программе.
3. Данные, при обработке которых правильными алгоритмами системы происходит сбой – это?
+1) выброс;
2) ошибка;
3) данные-тесты;
4) контрольные данные.

2.4. Практическое занятие № 5 (2 часа).
Тема: «Описание данных. Типы данных и переменные».
2.4.1. Вопросы к занятию:
1) Виды приложений.
2.4.2. Краткое описание проводимого занятия:
2.4.2.1. Ответы на вопросы семинарского (практического) занятия.
2.4.2.2. Проведение текущего контроля успеваемости
Задания для проведения текущего контроля успеваемости
1. Программа является монолитной, если она не содержит
+1) подпрограмм;
2) составных операторов;
3) пустых строк;
4) комментариев.
2. Подход к разработке программного комплекса, при котором он разбивается на программные модули (программы), образующие многоуровневую структуру, - это
+1) нисходящая разработка;
2) структурное программирование;
3) сквозной контроль;
4) макетирование.

3. Программирование взаимодействия между пользователем и выполняемой программой - это программирование
+1) интерфейса;
2) дружественности;
3) модифицируемости;
4) надежности.

2.5. Практическое занятие № 6 (2 часа).
Тема: «Обработка данных. Операции ввода-вывода».
2.5.1. Вопросы к занятию:
1) Описание переменных.
2) Типы данных.
2.5.2. Краткое описание проводимого занятия:
2.5.2.1. Ответы на вопросы семинарского (практического) занятия.
2.5.2.2. Проведение текущего контроля успеваемости
Задания для проведения текущего контроля успеваемости
1. Переменные, объявленные внутри процедуры или функции, называются
+1) локальными;
2) процедурными;
3) частными;
4) внутренними.
2. Переменная в программировании полностью характеризуется:
1) именем;
+2) именем, значением и типом;
3) именем и типом;
4) именем и значением;
5) значением.
3. Что такое синтаксис языка программирования
+ a) система правил, определяющих допустимые конструкции языка программирования из букв алфавита
b) система правил, определяющих допустимые конструкции языка программирования из английских букв
c) система правил, определяющих допустимые ключевые слова
d) система правил, определяющих допустимые имена переменных

2.6. Практическое занятие № 7, 8 (4 часа).
Тема: «Алгоритмические механизмы. Представления основных структур программирования».
2.6.1. Вопросы к занятию:
1) Операторы ввода вывода.
2) Математические операции.
2.6.2. Краткое описание проводимого занятия:
2.6.2.1. Ответы на вопросы семинарского (практического) занятия.
2.6.2.2. Проведение текущего контроля успеваемости
Задания для проведения текущего контроля успеваемости
1. Какая последовательность операций представлена в нарушении приоритета?
1) вызов функции; деление нацело; сложение;
2) выражение, заключенное в скобки; возведение в степень;
3) остаток от деления нацело; вычитание;
+4) деление нацело; умножение.
2. Какие типы данных занимают память менее 4 байт?
+1) Byte

+2) Boolean
+3) Integer
4) Single
5) Double
6) Long Integer

3. Какая функция возвращает число, содержащееся в строке string, как числовое значение?

+1) Val (String)
2) Str (number)
3) CStr (Выражение)
4) CInt (Выражение)

2.7. Практическое занятие № 9, 10 (4 часа).

Тема: «Модульные программы».

2.7.1. Вопросы к занятию:

1) Оператор условного перехода.
2) Логические операторы. Таблицы истинности.
3) Циклы.

2.7.2. Краткое описание проводимого занятия:

2.7.2.1. Ответы на вопросы семинарского (практического) занятия.
2.7.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

1. Какой из знаков обозначает "не равно"
+ a) <>
b) ><
c) =>
d) <=

2. Какое из слов обозначает логическое "ИЛИ"
+ a) OR
b) XOR
c) AND
d) NOT

3. Какое из слов обозначает логическое исключающее "ИЛИ"
+ a) XOR
b) OR
c) AND
d) NOT

2.8. Практическое занятие № 11, 12 (4 часа).

Тема: «Проектирование программ».

2.8.1. Вопросы к занятию:

1) Одномерные и многомерные массивы.
2) Текстовые файлы.
3) Двоичные файлы.

2.8.2. Краткое описание проводимого занятия:

2.8.2.1. Ответы на вопросы семинарского (практического) занятия.
2.8.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

1. В чем основное отличие файла от массива
+ a) число элементов, называемое длиной файла, не фиксировано
b) в файле могут быть только двоичные числа
c) в файле могут быть только данные типа byte

d) число элементов, называемое длиной массива, не фиксировано

2. Как правильно указывается имя элемента двумерного массива

+ a) A[3,5]
b) A(3,5)
c) A[3.5]
d) A(3.5)

3. В каком виде сортировки массивов элементы попарно сравниваются с соседними и при необходимости меняются местами

+ a) Пузырьковая сортировка
b) Классическая сортировка
c) Сортировка включениями
d) Быстрая сортировка

2.9. Практическое занятие № 13, 14, 15 (6 часов).

Тема: «Объекты и классы».

2.9.1. Вопросы к занятию:

1) Тип данных строка.
2) Тип данных структура.

2.9.2. Краткое описание проводимого занятия:

2.9.2.1. Ответы на вопросы семинарского (практического) занятия.
2.9.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

1. Переменная какого типа представляет собой один символ

+ a) char
b) string
c) byte
d) word

2. Переменная какого типа содержит в себе строку текста

+ a) string
b) char
c) byte
d) word

3. Переменная какого типа содержит в себе "истину" или "ложь"

+ a) boolean
b) char
c) string
d) byte

2.10. Практическое занятие № 16, 17 (4 часа).

Тема: «Взаимодействие объектов в программе».

2.10.1. Вопросы к занятию:

1) Динамические переменные.
2) Динамические массивы.

2.10.2. Краткое описание проводимого занятия:

2.10.2.1. Ответы на вопросы семинарского (практического) занятия.
2.10.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

1. Массив является динамическим, когда в качестве параметра переменной используется:

+1) массив без указания размера;
2) переменная без указания типа;
3) подпрограмма без списка параметров;

4) массив фиксированного размера.

2. Из чего состоят адреса байтов памяти

+ a) сегмент и смещение
  b) сектор и смещение
  c) сдвиг и смещение
  d) сдвиг и сектор

3. Указатель - это переменная, которая …

+ a) в качестве своего значения содержит адрес байта памяти
  b) в качестве своего значения содержит адрес файла
  c) ссылается на значение другой переменной
  d) указывает куда надо перейти при работе оператора безусловного перехода