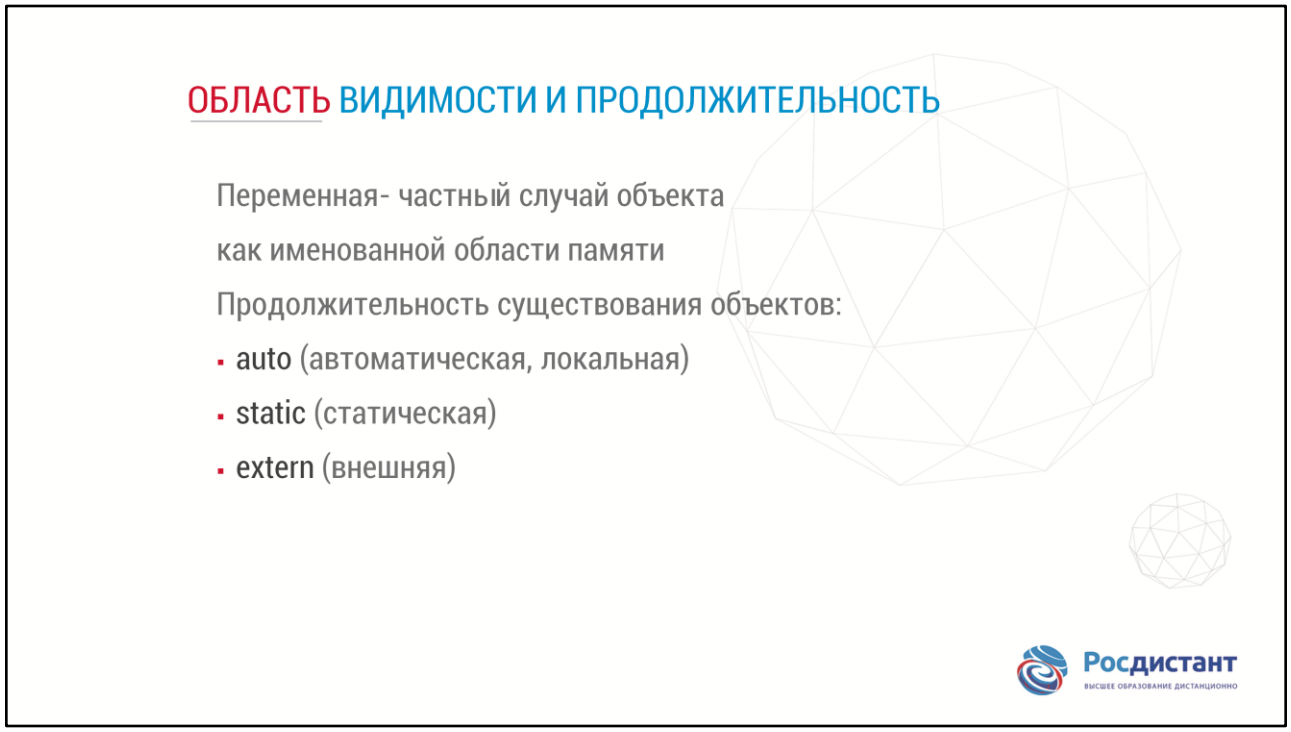

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО
ОСНОВЫ
ПРОГРАММИРОВАНИЯ
4 ЧАСТЬ



Слайд 89

Тема 4. Указатели и адреса объектов

Одним из основных понятий языка **C++** является понятие объекта как некоторой области памяти. Тип переменной определяет совокупность значений, который может принимать переменная, а также размер памяти в байтах, выделенной под хранение значений переменной.

При выполнении инициализации переменной, ей автоматически присваивается свободный адрес памяти, и, любое значение, которое мы присваиваем переменной, сохраняется по этому адресу памяти.

Прежде всего, нужно разобраться с двумя терминами: область видимости и продолжительность**.** Область видимости определяет, где можно использовать переменную. Продолжительность, или как еще называют - время жизни определяет, где переменная создаётся и где уничтожается. Эти понятия тесно связаны между собой.

Переменные, определённые внутри блока, называются локальными переменными. Локальные переменные имеют автоматическую продолжительность и локальную область видимости. Они создаются и инициализируются в точке определения и уничтожаются при выходе из блока.

Для автоматических переменных память выделяется динамически: во время выполнения программы при входе в блок или функцию и освобождается при

выходе из блока или функции. Ключевое слово **auto** присваивается по умолчанию, можно явно не указывать.

Статические переменные хранятся в памяти, могут быть локальными и внешними. Статические переменные отличаются от автоматических переменных тем, что при выходе из блока, или функции, в которых они были объявлены, значения статических переменных не теряются, выделенная память не освобождается. Они не доступны вне блока, или функции, в котором объявлены. Это внутренний тип компоновки и статическая продолжительность существования.

ПРИМЕР ИСПОЛЬЗОВАНИЯ СТАТИЧЕСКИХ ПЕРЕМЕННЫХ

Фрагмент программы	Результаты на экране

for (int i=1; i<=3; i++) // i – локальная
{ cout<"\nЛок= "<<i;
{static int i=3; // i – статическая
cout<<"\tСтат="<<i--;}
}
	Лок = 1 Стат =3
	Лок = 2 Стат =2
	Лок = 3 Стат =1

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Слайд 90

«Внешние переменные – это внешний тип компоновки и статическая продолжительность существования.

Переменная, имеющая внешние связи, называется внешней переменной. Она может использоваться как в файле, в котором определена, так и в других файлах.

Если вы хотите сделать глобальную переменную внутренней, которую можно использовать только внутри одного файла, используйте ключевое слово **static** . Аналогично, если вы хотите сделать глобальную переменную внешней, которую можно использовать в любом файле, используйте ключевое слово **extern** .

Ключевое слово **static** можно использовать для объявления переменных и функций в глобальной области видимости, области пространства имен и области класса. Статические переменные также могут быть объявлены в локальной области видимости.

На слайде приведен пример использования статических переменных.

В данном примере используется оператор цикла с параметром. В структуре этого оператора описана и определена переменная, что означает, что этот оператор является блоком, назовем его внешним блоком.

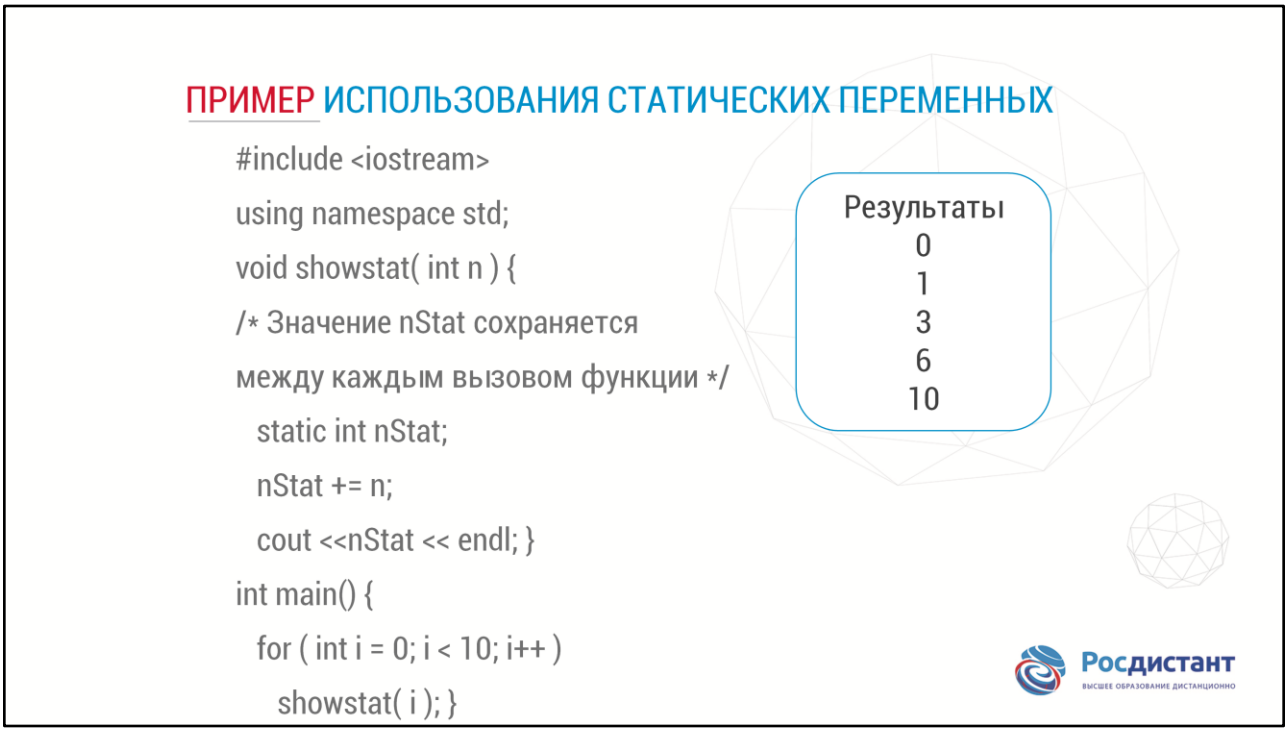
Первая переменная **i ,** описанная в структуре оператора **for** является локальной

переменной и сфера ее видимости – от начала и до конца работы этого оператора.

Вторая переменная **i** описана внутри еще одного блока, назовем его внутренним блоком, образованного фигурными скобками в теле цикла. Эта переменная описана как статическая и сохраняет свою продолжительность до конца работы внешнего блока.

Приведенные результаты демонстрируют, что статическая переменная, описанная во внутреннем блоке, сохраняет значение до конца работы внешнего блока.

Если в данном фрагменте убрать во внутреннем блоке статическое описание переменной, то ее значение в блоке будет равным числу три при каждом прохождении через цикл.



Слайд 91

«Статическая длительность означает, что объект или переменная выделяется при запуске программы и освобождается при ее завершении. Внешняя компоновка означает, что имя переменной видно за пределами файла, в котором эта переменная объявлена. Внутренняя компоновка означает, что имя не видно за пределами файла, в котором объявлена переменная. По умолчанию объект или переменная, определенные в глобальном пространстве имен, имеют статическую длительность и внешнюю компоновку.

Ключевое слово **static** можно использовать в следующих ситуациях:

- При объявлении **static** переменной или функции в области видимости файла ключевое слово указывает, что переменная или функция имеет внутреннюю компоновку. При объявлении переменной она имеет статическую длительность, и компилятор инициализирует ее со значением ноль, если не указано другое значение.
- При объявлении **static** переменной в функции ключевое слово указывает, что переменная удерживает свое состояние между вызовами этой функции.
- Объявление членов объединения как статических невозможно. Однако глобально объявленное анонимное объединение должно быть явно объявлено **static** .»

В этом примере показано, как статическая переменная **nStat ,** объявленная в

функции, удерживает свое состояние между вызовами этой функции. На слайде указаны результаты вычислений с использованием статических переменных.

Локальные автоматически создаваемые объекты или переменные инициализируются каждый раз, когда поток элемента управления достигает их определения. Локальные статические объекты или переменные инициализируются, когда поток элемента управления достигает их определения в первый раз.

УКАЗАТЕЛИ И АДРЕСА ОБЪЕКТОВ

Описание указателей:

<тип> *<имя - указатели>;

Примеры	Пояснения
int *p1, *p2 ;	описание указателей
char c = 'd' ; char *pc =&c;	описание переменной инициализация указателя на объект
char *pn (NULL);	нулевой указатель на объект

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Слайд 92

Как говорилось ранее, все используемые в программе переменные должны быть описаны и определены. Согласно описанию, определяется тип переменной, объем памяти, отводимой под хранение переменной.

На слайде показано, как описывать переменную указатель.

Указатель – это, переменная, значением которой является адрес памяти, или ячейка памяти. Указатели объявляются точно так же, как и обычные переменные, только со звёздочкой между типом данных и идентификатором, Только инициализируется указатель не значением одного из множества типов данных языка **C++,** а адресом, адресом некоторой переменной, которая была объявлена ранее.

Нам не нужно беспокоиться о том, какие конкретно адреса памяти выделены для определенных переменных. Мы просто ссылаемся на переменную через присвоенный ей идентификатор, а компилятор конвертирует это имя в соответствующий адрес памяти. Однако доступ к участкам памяти возможен и через указатели.

Значениями указателей служат адреса участков памяти, выделенных для объектов конкретных типов. В определении и описании указателя всегда присутствует обозначение соответственного ему типа. С помощью указателя можно получить доступ ко всему сохраняемому объекту в целом.

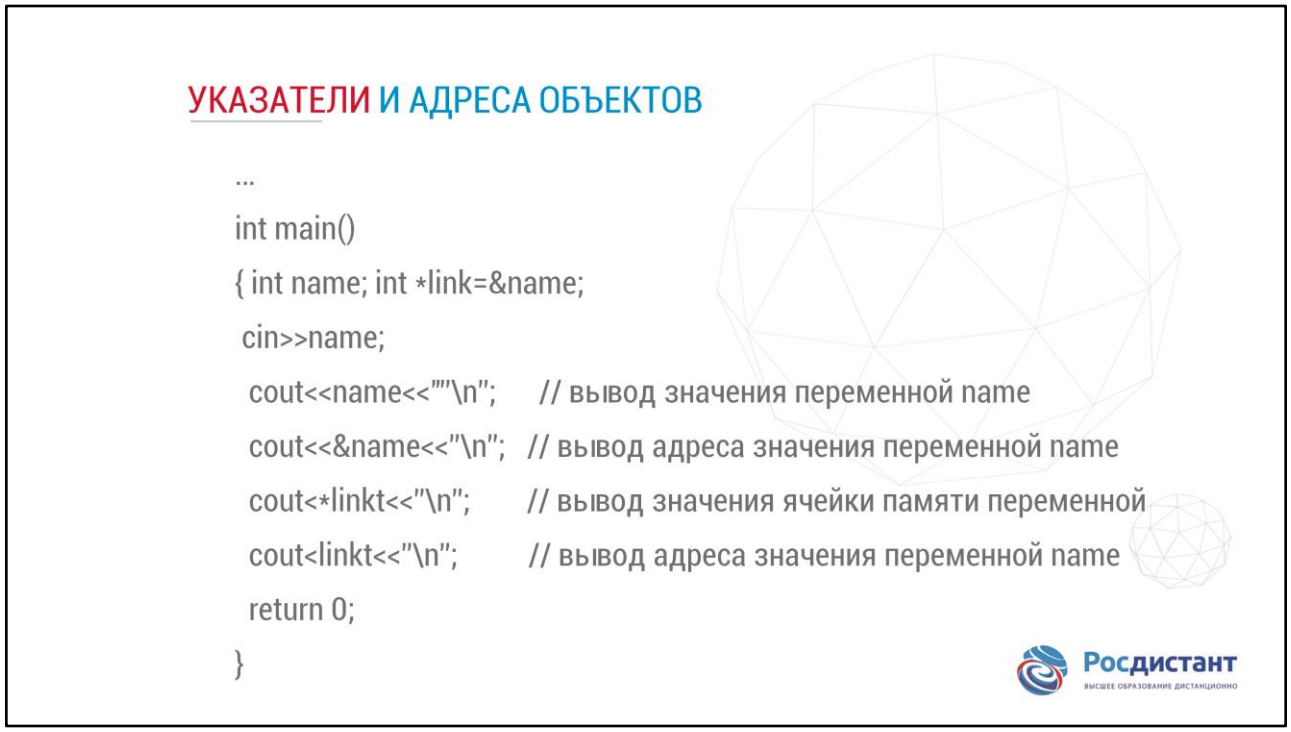
На слайде приведены примеры описания переменных. Две основные операции над переменными-указателями – это разыменование, символ ***,** и получение адреса, символ **&.**

В качестве типа, который используется при объявлении указателя, можно выбрать тип **void**. Но в этом случае при инициализации указателя придется приводить его к типу переменной, на которую он указывает.

Не следует путать оператор взятия адреса со ссылкой на некоторое значение, которое так же визуально отображается символом &.

Указатели делятся на - указатели на объекты и на указатели на функции.

С указателями на функции мы познакомимся позднее.



Слайд 93

На слайде приведен фрагмент обращения к одним и тем же ячейкам памяти через указатель и через имя переменной, через идентификатор.

Синтаксически язык **C++** принимает объявление указателя, когда звёздочка находится рядом с типом данных, с идентификатором или даже посередине. Обратите внимание, эта звёздочка не является оператором разыменования. Это всего лишь часть синтаксиса объявления указателя.

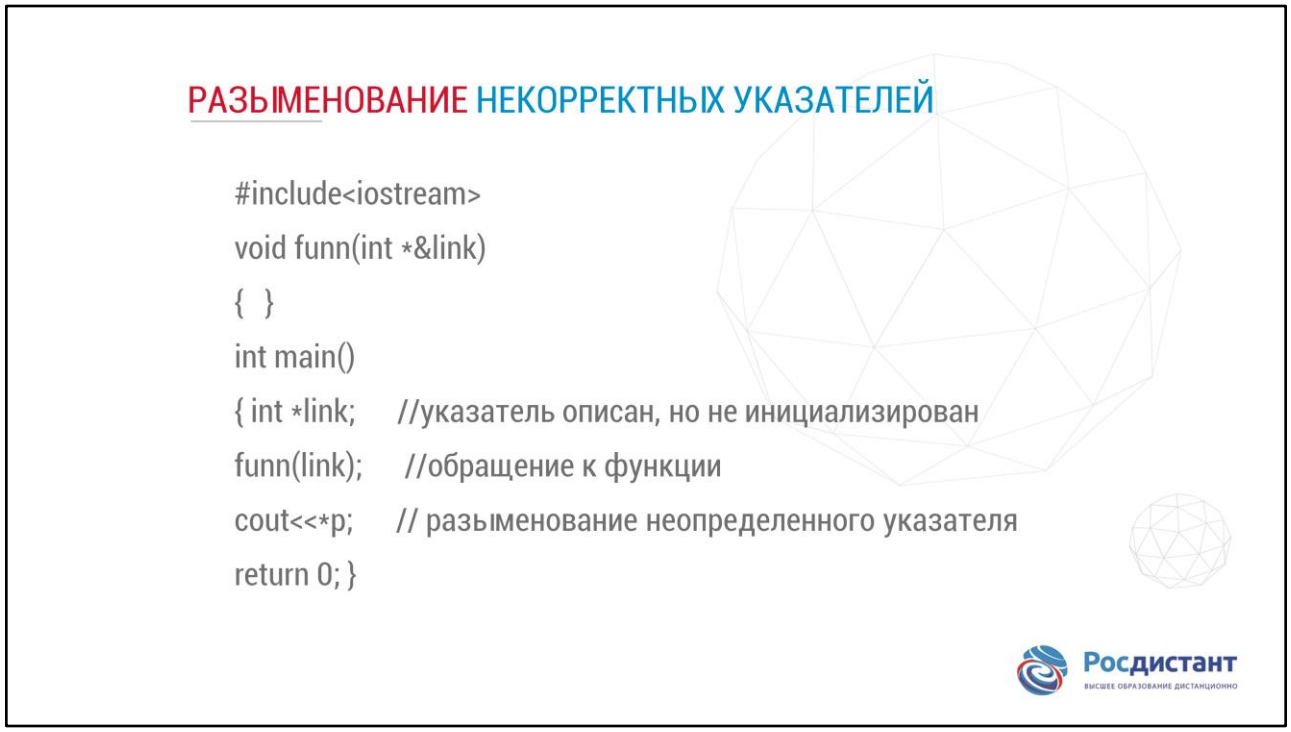
Указатели должны иметь тип данных. Без типа указатель не знал бы, как интерпретировать содержимое, на которое он указывает при разыменовании. Поэтому и должны совпадать тип указателя с типом переменной. Если они не совпадают, то указатель при разыменовании может неправильно интерпретировать биты. Например, вместо вещественного типа **double** использовать целочисленный тип **int** . Под хранение переменных целого и вещественного типа отводится одинаковое количество байт, по четыре байта.

Однако, при объявлении нескольких указателей, звёздочка должна находиться возле каждого идентификатора. Это легко забыть, если привыкли указывать звёздочку возле типа данных, а не возле имени переменной.

При объявлении указателя, рекомендуется указывать звёздочку возле имени переменной. Как и обычные переменные, указатели не инициализируются при объявлении. Содержимым неинициализированного указателя является

обычный мусор

Оператор разыменования выглядит так же, как и операция умножения, отличить их можно по тому, что операция разыменования — унарная операция, а операция умножения — бинарная. Как известно, в унарных операциях участвует один, а в бинарных операциях участвуют два операнда.



Слайд 94

«Так как указатель есть адрес, а адреса распределяет сам компьютер, то указатель не может быть инициализирован непосредственно. Знак **&** перед именем переменной возвращает ее адрес, или, как говорят, дает ссылку. Для того чтобы получить значение переменной, производится операция разыменования. Знак ***** перед указателем возвращает значение переменной, на которую указатель ссылается.

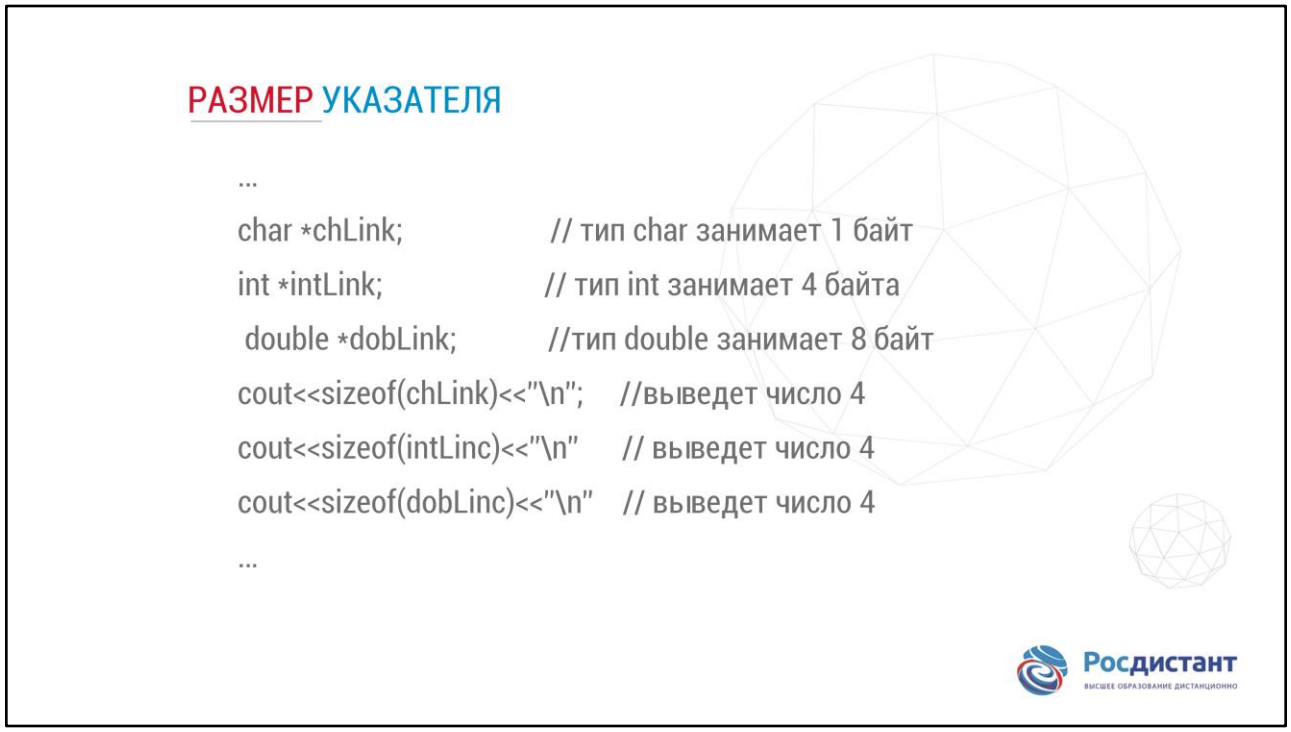
Указатели в языке **C++** по своей природе являются небезопасными, а их неправильное использование — один из лучших способов получить сбой программы.

При разыменовании указателя, программа пытается перейти в ячейку памяти, которая хранится в указателе и извлечь содержимое этой ячейки. По соображениям безопасности современные операционные системы запускают программы для предотвращения их неправильного взаимодействия с другими программами. А также, для защиты стабильности самой операционной системы.»

Рассмотрим на слайде фрагмент не корректного использования указателя. Если программа попытается получить доступ к ячейке памяти, не выделенной для нее операционной системой, то сразу завершится выполнение этой программы. Переменная-указатель **p** не описана и не определена, не известен адрес

памяти, по которому происходит обращение.

Фрагмент программы на слайде иллюстрирует вышесказанное. При запуске вы получите сбой.



Слайд 95

Размер указателя зависит от архитектуры, на которой скомпилирован исполняемый файл: тридцати двух-битный исполняемый файл использует тридцати двух-битные адреса памяти. Следовательно, указатель на 32-битном устройстве занимает 32 бита, что составляет 4 байта. С 64-битным исполняемым файлом указатель будет занимать 64 бита, что составляет 8 байт. И это вне зависимости от типа информации, на которую указывает переменная-указатель.

На слайде приведены примеры указателей на объекты разного типа: символьный тип, целочисленный тип и вещественный тип. Размер памяти, отводимой под хранение переменных перечисленных типов соответственно различны.

Используя функцию **sizeof** для определения размера указателей на переменные разного типа, можно увидеть, что размер указателя всегда один и тот же. Это связано с тем, что указатель — это всего лишь адрес памяти, а количество бит, необходимое для доступа к адресу памяти на определенном устройстве, всегда постоянное.

Предлагаем с помощью функции **sizeof** определить количество байт, отводимых под указатели на вашем устройстве. Примените проверку на указателях разного типа и убедиться в вышеизложенных положениях.

«Несмотря на то, что внутри функции значения переменных могут поменяться, по окончании работы, в основной программе они остаются неизменными. С данной проблемой можно справиться, используя передачу аргументов по ссылке с помощью указателей. При передаче указателя на самом деле передается лишь адрес объекта, а, следовательно, функция получает возможность манипулировать значением, находящимся по этому адресу. Этот факт так же свидетельствует о возможностях использования указателей. Подробнее о функциях на языке C++ мы поговорим в последующих темах.»

ОПЕРАЦИИ НАД УКАЗАТЕЛЯМИ

Операция	Наименование
*	операции разыменования, доступа по адресу
=	присваивание
&	получение адреса
+, -	сложение и вычитание (аддитивные);
++	инкремент или автоувеличение
--	декремент или автоуменьшение
<, >, ==, !=	операции отношений (операции сравнения)

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Слайд 96

«Динамические переменные создаются с помощью специальных функций и операций. Они существуют либо до конца работы программы, либо до тех пор, пока не будет освобождена выделенная под них память с помощью специальных функций или операций. То есть время жизни динамических переменных – от точки создания до конца программы или до явного освобождения памяти.»

Над указателями допустимы операции, приведенные в таблице.

При выполнении инициализации переменной, ей автоматически присваивается адрес памяти, и, любое значение, которое мы присваиваем переменной, сохраняется по этому адресу памяти.

Операции разыменования позволяет узнать, какой адрес памяти присвоен определённой переменной. Поскольку указатели содержат только адреса, то при присваивании указателю значения - это значение должно быть адресом.

Как только у нас есть указатель, указывающий на что-либо, мы можем его применить к нему операцию разыменования, чтобы получить значение, на которое он указывает. После операции разыменования указатель позволяет обратиться к содержимому ячейки памяти, в которой находится конкретная информация.

ПРИМЕРЫ ОПЕРАЦИЙ НАД УКАЗАТЕЛЯМИ

Примеры	Пояснения

int a =2, b =3;
 	
int *pa =&a, *pb =&b;
	Определены указатели на объекты целого типа

float c =0.1 , d =1.7;
 	
float *pc =&c, *pd =&d;
	Определены указатели на объекты вещественного типа

cout <<"\n int = "<<pa-pb;
 	
cout <<"\t float = "<<pd-pc;
	Результаты на экране:
	int = 1 float = -1

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Слайд 97

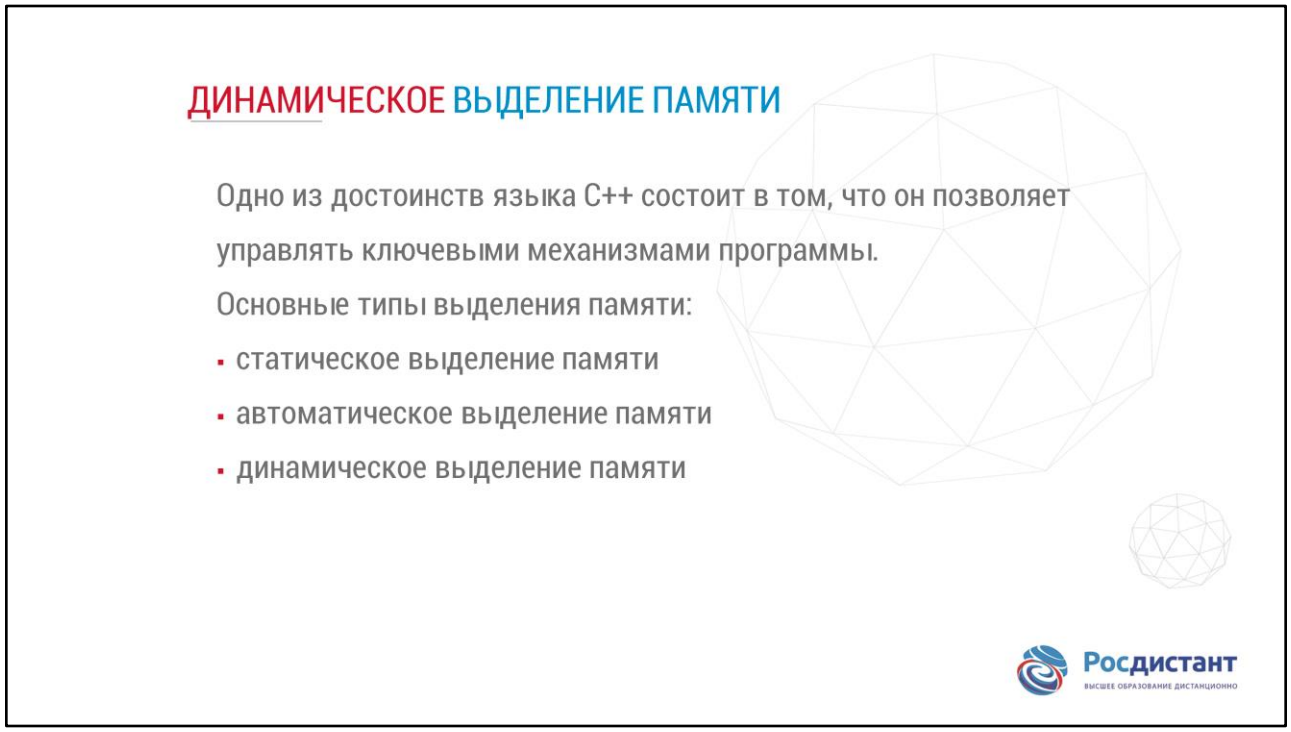
На слайде приведены примеры описания и определения указателей на переменные целого типа и переменные вещественного типа. Доступ к информации возможен как через имена переменных, так и через указатели.

Указатель должен иметь такой же тип, какой имеет объект, на который он указывает. Вычитая два объекта одного типа, можно определить расстояние между двумя участками памяти. Расстояние определяется в единицах, кратных длине в байтах объекта того типа, к которому отнесен указатель.

Разность однотипных указателей, содержащих адреса двух смежных объектов любого типа по абсолютной величине всегда равна единице.

На слайде продемонстрированы примеры с указателями на переменные разного типа, переменные целого типа и переменные вещественного типа. Разница указателей, в обоих случаях, равна единице.

При использовании переменной-указателя – подразумевается работа с адресами, если перед именем переменной-указателя ставится звездочка - происходит обращение к информации, записанной по данному адресу.



Слайд 98

Язык **C++** поддерживает три основных типа выделения, распределения памяти, с двумя из которых, мы уже знакомы.

Статическое выделение памяти выполняется для статических и глобальных переменных. Память выделяется один раз, при запуске программы, и сохраняется на протяжении работы всей программы.

«Автоматическое выделение памяти выполняется для параметров функции и локальных переменных. Память выделяется при входе в блок, в котором находятся эти переменные, и удаляется при выходе из него.

Динамическое выделение памяти является выделением памяти из операционной системы по требованию. При использовании динамической памяти появляются проблемы, связанные с тем, что любое выделение или освобождение памяти – это системный вызов, замедляющий работу программы.

Как для статического, так и автоматического распределения памяти размер переменной должен быть известен во время компиляции. Память для обычных переменных выделяется из специального резервуара памяти - стека. Объём памяти стека в программе, как правило, невелик.

Эти проблемы легко устраняются с помощью динамического выделения памяти. Эта память выделяется из большего хранилища, управляемого операционной

системой - кучи. На современных компьютерах размер кучи может составлять гигабайты памяти.»

«Правильное понимание и использование указателей имеет большое значение при создании большинства программ по четырем причинам:

- указатели предоставляют способ, позволяющий функциям модифицировать передаваемые аргументы;
- указатели используются для поддержки системы динамического выделения памяти;
- использование указателей может повысить эффективность работы некоторых подпрограмм;
- указатели, как правило, используются для поддержки некоторых структур данных типа связанные списки и двоичные деревья.»

Слайд 99

«Рассмотрим способы выделения памяти, динамического выделения памяти, связи указателей и динамического распределения памяти.

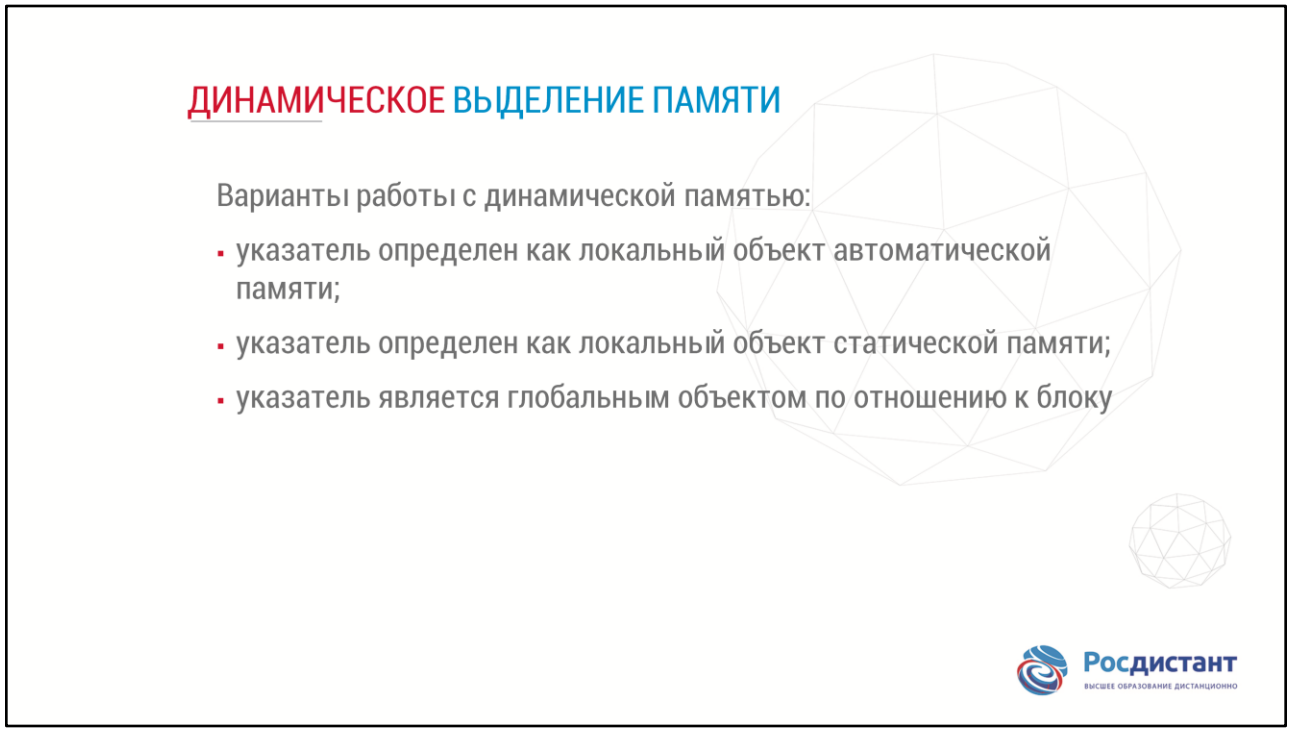
Существует два основных способа хранения информации в оперативной памяти. Первый заключается в использовании глобальных и локальных переменных. В случае глобальных переменных выделяемые под них поля памяти остаются неизменными на все время выполнения программы. Под локальные переменные программа отводит память из стекового пространства. Однако локальные переменные требуют предварительного определения объема памяти, выделяемой для каждой ситуации. Хотя C++ эффективно реализует такие переменные, они требуют от программиста заранее знать, какое количество памяти необходимо для каждой ситуации.

Второй способ, которым C++ может хранить информацию, заключается в использовании системы динамического распределения. При этом способе память распределяется для информации из свободной области памяти по мере необходимости. Область свободной памяти находится между кодом программы с ее постоянной областью памяти и стеком. Динамическое размещение удобно, когда неизвестно, сколько элементов данных будет обрабатываться.

По мере использования программой стековая область увеличивается вниз, то

есть программа сама определяет объем стековой памяти.
Например, программа с большим числом рекурсивных функций займет больше стековой памяти, чем программа, не имеющая рекурсивных функций, так как локальные переменные и возвращаемые адреса хранятся в стеках. Память под саму программу и глобальные переменные выделяется на все время выполнения программы и является постоянной для конкретной среды.»

Память, выделяемая в процессе выполнения программы, называется динамической. После выделения динамической памяти она сохраняется до ее явного освобождения, что может быть выполнено только с помощью специальной операции или библиотечной функции.



Слайд 100

Если динамическая память не освобождена до окончания программы, то она освобождается автоматически при завершении программы. Тем не менее, явное освобождение ставшей ненужной памяти является признаком хорошего стиля программирования.

В процессе выполнения программы участок динамической памяти доступен везде, где доступен указатель, адресующий этот участок. Таким образом, возможны следующие три варианта работы с динамической памятью, как представлено на слайде.

«Все переменные, объявленные в программе, размещаются в одной непрерывной области памяти, которую называют сегментом данных. Такие переменные не меняют своего размера в ходе выполнения программы и называются статическими. Размера сегмента данных может быть недостаточно для размещения больших объемов информации. Выходом из этой ситуации является использование динамической памяти. Динамическая память – это память, выделяемая программе для ее работы за вычетом сегмента данных, стека, в котором размещаются локальные переменные подпрограмм и собственно тела программы.

Для работы с динамической памятью используют указатели. С их помощью осуществляется доступ к участкам динамической памяти, которые

называются динамическими переменными. Для хранения динамических переменных выделяется специальная область памяти, называемая "кучей".

Динамические переменные создаются с помощью специальных функций и операций. Они существуют либо до конца работы программы, либо до тех пор, пока не будет освобождена выделенная под них память с помощью специальных функций или операций. То есть время жизни динамических переменных – от точки создания до конца программы или до явного освобождения памяти.»

ОПЕРАЦИИ С УКАЗАТЕЛЯМИ

Операция	Пояснения
int *p;	указатель находится в неопределенном состоянии, после его описания
p= new int;	отводится место в оперативной памяти под хранение переменной целого типа, и этот адрес вносится в указатель p.
p=Null;	указатель на пустой адрес (никуда)
delete p;	освобождается память

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Слайд 101

Для динамического выделения памяти для одной переменной используется оператор **new:**

В примере выделяется память для целочисленной переменной. Оператор **new** возвращает указатель, содержащий адрес выделенной памяти. Для доступа к выделенной памяти создаётся указатель. Затем мы можем разыменовать указатель для получения значения.

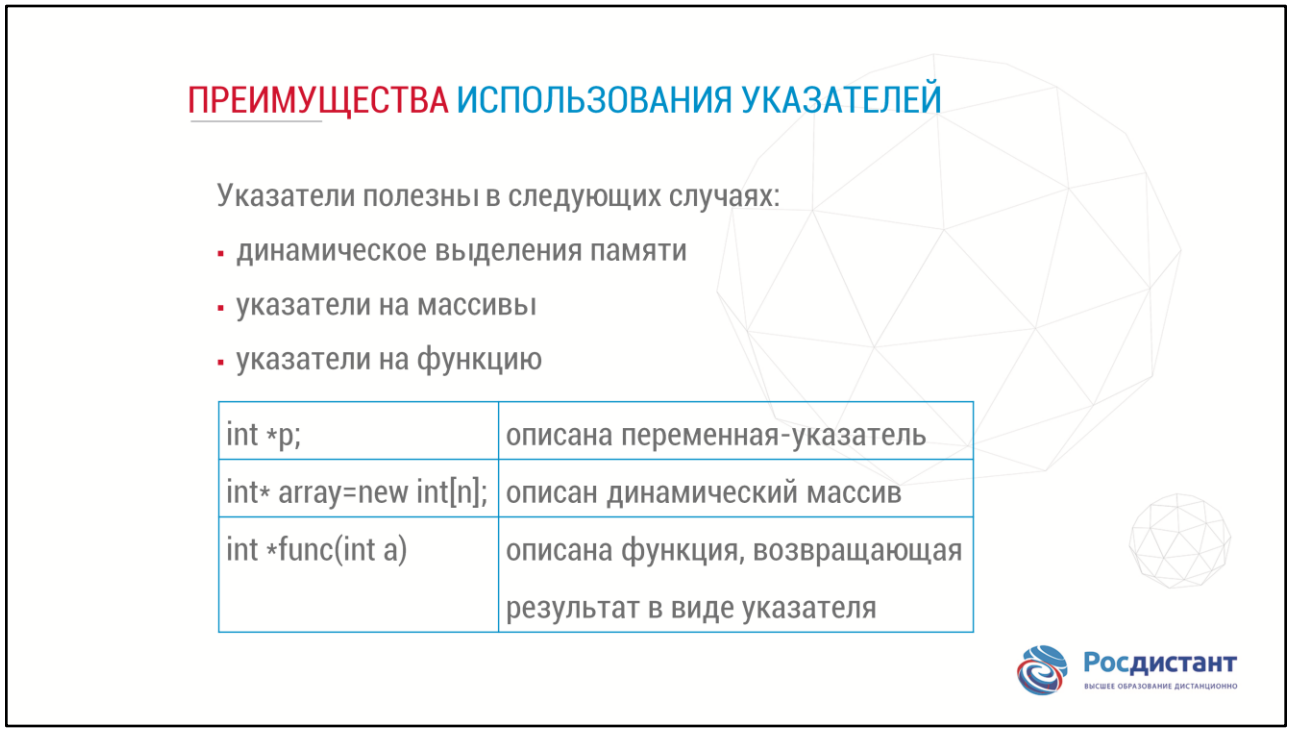
«В выделенный участок заносится значение, определяемое инициализатором, который не является обязательным элементом. В случае успешного выполнения оператор **new** возвращает адрес начала выделенного участка памяти. Если участок нужных размеров не может быть выделен, то оператор **new** возвращает нулевое значение адреса.

Когда уже выполнено все что было необходимо, с этой памятью, то её можно вернуть обратно в операционную систему. В отличие от статического или автоматического выделения памяти, программа самостоятельно отвечает за запрос и обратный возврат динамически выделенной памяти. Для динамических переменных это выполняется с помощью оператора **delete.**

Этот оператор на самом деле ничего не удаляет. Он просто возвращает память, которая была выделена ранее, обратно в операционную систему. Затем операционная система может переназначить эту память для любой другой

переменной. Оператор **new** чаще всего используется для размещения в памяти данных определенных пользователем типов, например, структур:

При выделении динамической памяти для массива его размеры должны быть полностью определены. Такая операция позволяет выделить в динамической памяти участок для размещения массива соответствующего типа, но не позволяет его инициализировать. В результате выполнения оператор **new** возвратит указатель, значением которого служит адрес первого элемента массива.



Слайд 102

«Операции **new** и **delete служат** для выделения и освобождения блоков памяти. Область памяти, в которой размещаются эти блоки, называется свободной памятью. Операция **new** позволяет выделить и сделать доступным свободный участок в основной памяти, размеры которого соответствуют типу данных, определяемому именем типа.

В случае успешного выполнения операция **new** возвращает адрес начала выделенного участка памяти.

Если участок нужных размеров не может быть выделен, нет памяти, то операция **new** возвращает нулевое значение адреса.»

Рассмотрим, в каких случаях полезно использование указателей на различные объекты.

Указатели являются единственным способом динамического выделения памяти на языке **C++. Э**ту тему мы рассмотрим в следующих лекциях. Это, безусловно, самый распространённый вариант использования указателей. Указатели эффективно использовать при обработке массивов. Указатели могут использоваться для итерации по массиву. Заранее не определять размер памяти, выделенной под хранение элементов массива, выделять и освобождать память под массив непосредственно во время выполнения программы.

Подведем итоги. Перечислим все преимущества использования переменных-

указателей:

Указатели могут использоваться для передачи большого количества данных в функцию без копирования этих данных, могут использоваться для передачи одной функции в качестве параметра другой функции.

Понятие указателя будет проходить тонкой линией во всех последующих темах, которые мы будем изучать: и в массивах, и в функциях, и в символьных массивам, и в файлах, и наконец, полностью раскроются при изучении динамических структур данных. Указатели могут использоваться для представления одной структуры в другую структуру, формируя, таким образом, целые цепочки, динамические структуры данных.

О динамических массивах, вернее о выделении динамической памяти под хранение элементов массива, мы поговорим уже на следующей лекции.

Введение в языки программирования C и C++ | Уроки C++ - Ravesli / https://ravesli.com/urok-2-vvedenie-v-yazyki-programmirovaniya-c-i-s/

Т, А. Павловская С/С++ Программирование на языке высокого уровня/http://cph.phys.spbu.ru/documents/First/books/7.pdf

Введение в программирование | Уроки C++ - Ravesli/https://ravesli.com/urok-1-vvedenie-v-programmirovanie/

Технология программирования - Информатика, автоматизированные информационные технологии и системы/ https://studref.com/441961/informatika/tehnologiya_programmirovaniya

Бьерн Страуструп. Язык программирования С++ 11/ https://vk.com/doc-145125017_450056359

Язык СИ++ Учебное собие / http://5fan.ru/wievjob.php?id=4301

А.А. Шелупанов, В.Н. Кирнос ИНФОРМАТИКА. БАЗОВЫЙ КУРС Учебник в четырех частях. /https://edu.tusur.ru/publications/521/download