УВАЖАЕМЫЕ СТУДЕНТЫ! Законспектируйте в своей рабочей тетради по дисциплине приведенную лекцию (объемом 4-5 страницы), ответьте *письменно* на контрольные вопросы.

Результаты работы, фотоотчет, предоставить преподавателю на e-mail: r.bigangel@gmail.com до 30.01.2023.

При возникновении вопросов по приведенному материалу обращаться по следующему номеру телефона: (072)111-37-59, (Viber, WhatsApp), vk.com: https://vk.com/daykini

ВНИМАНИЕ!!! ***При отправке работы, не забывайте указывать ФИО студента, наименование дисциплины, дата проведения занятия (по расписанию).***

Ознакомьтесь с материалом лекции кратко законспектируйте основные понятия и определения, представленные в лекции.

Лекция № 4
Тема «Лексические основы С++»

Язык С++ – это язык программирования общего назначения, цель которого – сделать работу серьёзных программистов более приятным занятием. За исключением несущественных деталей, язык С++ является надмножеством языка С. Помимо возможностей, предоставляемых языком С, язык С++ обеспечивает гибкие и эффективные средства определения новых типов.

Язык программирования служит двум взаимосвязанным целям: он предоставляет программисту инструмент для описания подлежащих выполнению действий и набор концепций, которыми оперирует программист, обдумывая, что можно сделать. Первая цель в идеале требует языка, близкого к компьютеру, чтобы все важные элементы компьютера управлялись просто и эффективно способом, достаточно очевидным для программиста. Язык С создавался на основе именно от этой идеи. Вторая цель в идеале требует языка, близкого к решаемой задаче, чтобы концепции решения могли быть выражены

понятно и непосредственно. Эта идея привела к пополнению языка С свойствами, превратившими его в язык С++.

Ключевое понятие в языке С++ – **класс**. Классы обеспечивают сокрытие информации, гарантированную инициализацию данных, неявное преобразование определяемых пользователем типов, динамическое определение типа, контроль пользователя над управлением памятью и механизм перегрузки операторов. Язык С++ предоставляет гораздо лучшие, чем язык С, средства для проверки типов и поддержки модульного программирования. Кроме того, язык содержит усовершенствования, непосредственно не связанные с классами, такие как: символические константы, встраивание функций в место вызова, параметры функций по умолчанию, перегруженные имена функций, операторы управления свободной памятью и ссылки. Язык С++ сохраняет способность языка С эффективно работать с аппаратной частью на уровне битов, байтов, слов, адресов и т.д. Это позволяет реализовывать пользовательские типы с достаточной степенью эффективности.

1. Алфавит

Множество символов языка С включает:

- прописные буквы латинского алфавита;
- строчные буквы латинского алфавита;
- арабские цифры;
- разделители: , . ; : ? ! ' " | / \ ~ _ ^ ( ) { } [ ] < > # % & - = + *

Остальные символы могут быть использованы только в символьных строках, символьных константах и комментариях. Язык С++ различает большие и маленькие буквы, таким образом, *name* и *Name* – **разные идентификаторы**.

2. Литералы

Литералы в языке С++ могут быть целые, вещественные, символьные и строковые.

– Целые (можно испрользовать апостроф как разделитель групп разрядов):

– десятичные: 10, 132, -32179, 2'147'483'647;

– двоичные (предваряются символами «0b»): 0b11, 0b1010b, 0b1111'0011;

– восьмеричные (предваряются символом «0»): 010, 0204, -076663;

– шестнадцатеричные (предваряются символами «0x»): 0xA, 0x84, 0x7db3.

– Вещественные: 15.75, 1.575e1, .75, -.125

– Символьные: 'a', 'e', '.', '?', '2'.

– Строковые: "строка".

3. Комментарии

Комментарий – это последовательность символов, которая игнорируется компилятором языка C++. Комментарий имеет следующий вид: /*<символы>*/. Комментарии могут занимать несколько строк, но не могут быть вложенными. Кроме того, часть строки, следующая за символами //, также рассматривается как комментарий.

Разумное использование комментариев (и согласованное употребление отступов) может сделать чтение и понимание программы более приятным занятием. При неправильном использовании комментариев читабельность программы может, напротив, серьёзно пострадать. Компилятор не понимает смысл комментариев, поэтому не существует способа проверить, что комментарий:

1. содержателен;

2. имеет какое-то отношение к программе;

3. не устарел.

Удачно подобранный и написанный набор комментариев является существенной частью хорошей программы. Написание «правильных» комментариев может оказаться не менее сложной задачей, чем написание самой программы.

На уровне библиотек/программ/функций комментарии отвечают на вопрос «ЧТО?»: «Что делают эти библиотеки/программы/функции?». Например: // Эта функция использует метод Ньютона для вычисления корня функции. Такие комментарии позволяют понять, что делает программа, без необходимости смотреть на исходный код.

Внутри библиотек/программ/функций комментарии отвечают на вопрос «КАК?»: «Как код выполняет задание?». Например: // Чтобы получить случайный элемент, мы делаем следующее: …

На уровне однострочного кода комментарии отвечают на вопрос «ПОЧЕМУ?»: «Почему код выполняет задание именно так, а не иначе?». Плохой комментарий на уровне операторов объясняет, что делает код. Если вы когда-нибудь писали код, который был настолько сложным, что нужен был комментарий, который бы объяснял, что он делает, то вам нужно было бы не писать комментарий, а переписывать этот код.

4. Типы данных языка C++

Имя	Размер	Представляемые значения	Диапазон
bool	1 байт	логические	false, true
(signed) char	1 байт	символы целые числа	от –128 до 127
wchar_t	2 байта	символы Unicode	от 0 до 65535
(signed) short int	2 байта	целые числа	от -32768 до 32767
(signed) int	зависит от реализации (в последних компиляторах обычно 4 байта)	целые числа	
(signed) long int	4 байта	целые числа	от -2147483648 до 2147483647
(signed) long long int (signed) __int64 (MS)	8 байт	целые числа	от – 9,223,372,036,854,775,808 до 9,223,372,036,854,775,807
unsigned char	1 байт	символы целые числа	от 0 до 255
unsigned short int	2 байта	целые числа	0 до 65535
unsigned int	зависит от реализации (в последних компиляторах обычно 4 байта)	целые числа	

unsigned long int	4 байта	целые числа	от 0 до 4294967295
(unsigned) long long int (unsigned) __int64 (MS)	8 байт	целые числа	от 0 до 18,446,744,073,709,551,615
float	4 байта	вещественные числа	от 1.175494351e–38 до 3.402823466e+38
double	8 байт	вещественные числа	от 2.2250738585072014e–308 до 1.7976931348623158e+308
long double	зависит от реализации	вещественные числа	

В языке C++ также существуют перечислимый тип – enum, который является подмножеством целого типа, и пустой тип – *void*, который имеет специальное назначение. Он используется для объявления функций, которые не возвращают никакого значения, а также для объявления указателей на значение типа *void*. Такие указатели могут быть преобразованы к указателям на любой другой тип.

В языке C++ можно объявлять структуры и так называемые объединения.

В языке C++ **нет специальных типов для массивов и строк**, которые представляются массивом символов.

В языке C не существовало логического типа. Логические значения представлялись данными целого типа, при этом значение 0 соответствовало логическому значению *ложь*, а **все остальные целые значения** соответствовали логическому значению *истина*. В языке C++ сохранена данная логика. По определению, *true* имеет значение 1 при преобразовании к целому типу, а *false* – значение 0. И наоборот, целые можно неявно преобразовать в логические значения: при этом ненулевые целые преобразуются в *true*, а ноль – в *false*. В любом месте, где требуется логическое значение, может стоять целочисленное выражение. В арифметических и логических выражениях логические значения преобразуются в целые, операции выполняются над преобразованными величинами.

Указатель можно неявно преобразовать в логическое значение, при этом ненулевой указатель принимает значение *true*, нулевой – *false*.

Такой подход позволяет вместо логической и целочисленной переменных объявлять только целочисленную, при этом значение переменной, равное 0, говорит об отсутствии некоторого признака у объекта, а остальные значения говорят о его наличии, и при этом несут какую-либо дополнительную информацию.

При выполнении бинарных операций производятся **преобразования по умолчанию** для приведения операндов к одному и тому же типу, который потом используется как тип результата:

1. если один из операндов имеет тип *long double*, другой тоже преобразуется в *long double*;

– иначе, если один операнд имеет тип *double*, то второй операнд преобразуется к типу *double*;

– иначе, если один операнд имеет тип *float*, то второй операнд преобразуется к типу *float*;

– иначе над обоими операндами производится интегральное продвижение, а именно: значения типов *char*, *signed char*, *unsigned char*, *short int* и *unsigned short int* преобразуются в *int*, если *int* может представить все значения исходных типов, в противном случае они преобразуются в *unsigned int*; *bool* преобразуется в *int*.

2. затем если один операнд имеет тип *unsigned long*, то второй операнд преобразуется к типу *unsigned long*;

– иначе, если один из операндов относится к типу long int, а другой к типу unsigned int, то если long int может представить все значений типа unsigned int, unsigned int преобразуется в long int, иначе оба операнда преобразуются в unsigned long int;

– иначе, если один операнд имеет тип long int, то второй операнд преобразуется к типу long int;

– иначе, если один операнд имеет тип unsigned int, то второй операнд преобразуется к типу unsigned int;

– иначе оба операнда имеют тип *int*.

В языке C++ нет операций преобразования между символом и кодом символа, т.к. в оперативной памяти символ и так храниться в виде его кода. Поэтому можно к переменной, хранящей символ, прибавить 1 и получить следующий символ.

5. Операции языка C++

Данная таблица описывает операции языка C++. Операции разделены на группы, расположенные в порядке убывания приоритета операций.

Знак операции	Наименование	Ассоциативность
::	Разрешение области видимости	Слева направо
( ) [ ] . -> ++ -- static_cast dynamic_cast reinterpret_cast const_cast	Первичные Постфиксный инкремент и декремент Преобразование с проверкой во время компиляции Преобразование с проверкой во время выполенения Преобразование без проверки Константное преобразование	Слева направо
- ~ ! * & ++ -- sizeof (<*тип*>)<*выражение*> new delete	Унарные Префиксный инкремент и декремент Вычисление размера Приведение типа Выделение памяти Освобождение памяти	Справа налево
.* ->*	Выбор члена класса	Слева направо
* / %	Мультипликативные	Слева направо
+ -	Аддитивные	Слева направо
<< >>	Сдвиг	Слева направо
< > <= >=	Отношение	Слева направо
== !=	Отношение	Слева направо
&	Поразрядное И	Слева направо
^	Поразрядное исключающее ИЛИ	Слева направо
|	Поразрядное ИЛИ	Слева направо
&&	Логическое И	Слева направо
||	Логическое ИЛИ	Слева направо
? :	Условная операция	Справа налево
= *= /= %= += - = <<= >>= &= ^= |=	Простое и составное присваивания	Справа налево
throw	Генерация исключения	Слева направо

,	Операция последовательного вычисления	Слева направо

– **::** – операция разрешения области видимости. При повторном объявлении имени во вложенном блоке или классе предыдущие объявления оказываются *скрытыми*. Однако скрытое имя члена класса можно использовать, квалифицировать его именем класса при помощи операции разрешения области видимости. Скрытое глобальное имя можно использовать, если квалифицировать его унарной операцией разрешения области видимости.

– **( )** – выражение в скобках (используется для изменения порядка вычисления) или вызов функции.

– **[ ]** – индексное выражение, используется для работы с массивами.

– **.** и **->** – выбор элемента, используются при работе с классами, структурами и объединениями.

– Операции **static_cast** (преобразование с проверкой во время компиляции), **dynamic_cast** (преобразование с проверкой во время выполнения), **reinterpret_cast** (преобразование без проверки), **const_cast** (константное преобразование) осуществляют различные *преобразования типов*. Они имеют следующий синтаксис: *операция<новый тип>(выражение)*. Угловые скобки являются элементом синтаксиса. Операция **static_cast** осуществляет преобразование родственных типов, например, указателя на один тип к указателю на другой тип из той же иерархии классов, целый тип в перечисление или тип с плавающей точкой в интегральный. Операция **reinterpret_cast** управляет преобразованиями между несвязанными типами, например, целых в указатели или указателей в другие (несвязанные) указатели. Такое различие позволяет компилятору осуществлять минимальную проверку типов при использовании **static_cast**, а программисту – легче обнаружить опасные преобразования, представляемые **reinterpret_cast**. Преобразование **dynamic_cast** выполняется и проверяется на этапе

выполнения. Преобразование **const_cast** аннулирует действие модификатора *const*.

- **-** – унарный минус.
- **~** – обратный код (см. лекцию 8).
- **!** – логическое отрицание.
- ***** – косвенная адресация (см. лекцию 5).
- **&** – адресация (см. лекцию 5).
- Операции ++ и -- инкрементируют (увеличивают на 1) и декрементируют (уменьшают на 1) свой операнд. Операнд должен иметь целый, вещественный тип или быть указателем. Операции инкремента и декремента могут записываться как перед своим операндом (префиксная форма записи), так и после него (постфиксная форма записи). При префиксной форме записи операнд сначала инкрементируется или декрементируется, а затем его новое значение участвует в дальнейшем вычислении выражения, содержащего данную операцию. При постфиксной форме записи операнд инкрементируется или декрементируется лишь после того, как его старое значение участвует в вычислении выражения. Таким образом, результатом операций инкремента и декремента является либо новое, либо старое значение операнда. Например, если переменная $i = 0$, то выражение $a[++i] = 1$ меняет элемент $a[1]$, а выражение $a[i++] = 1$ меняет элемент $a[0]$. В обоих случая переменная i получает новое значение, равное 1.
- **sizeof** – вычисление размера в байтах переменной или типа.
- **Операция приведения типа** записывается следующим образом: (<*новый тип*>)<*выражение*>. Например, (long int)n приводит переменную *n* к типу long int. При преобразовании типов надо помнить, что при преобразовании между знаковыми/беззнаковыми значениями и при преобразовании от типа с большей размерностью к типу с меньшей размерностью могут возникнуть ошибки. Более безопасным способом преобразования типов является использование операций *static_cast*, *dynamic_cast*, *reinterpret_cast* и *const_cast*.

– **%** – остаток от деления.

– В языке C++ имеется одна тернарная операция – **условная операция**. Она имеет следующий синтаксис: <*операнд 1*> ? <*операнд 2*> : <*операнд 3*>. Если <*операнд 1*> имеет ненулевое значение, то вычисляется <*операнд 2*> и результатом условной операции является его значение. Если же <*операнд 1*> равен нулю, то вычисляется <*операнд 3*> и результатом является его значение. В любом случае вычисляется только один из операндов, <*операнд 2*> или <*операнд 3*>, но не оба.

– **Простое присваивание**. Операция простого присваивания обозначается знаком «=». Значение правого операнда присваивается левому операнду. Операция вырабатывает результат, который может быть далее использован в выражении. Результатом операции является присвоенное значение. Например, выражение $a = b = c = 0$ присваивает всем переменным значение 0, а в результате вычисления выражения $a = (b = 3) + (c = 5)$ переменная *c* будет иметь значение 5, переменная *b* будет иметь значение 3, и переменная *a* будет иметь значение 8.

– **Составное присваивание**. Операция составного присваивания состоит из простой операции присваивания, скомбинированной с какой-либо другой бинарной операцией. При составном присваивании вначале выполняется действие, специфицированное бинарной операцией, а затем результат присваивается левому операнду. Оператор $n += 5$ эквивалентен оператору $n = n + 5$, но при этом первый оператор легче для понимания и выполняется быстрее.

– **Операция последовательного вычисления «,»** обычно используется для вычисления нескольких выражений в ситуациях, где по синтаксису допускается только одно выражение. Однако, запятая, разделяющая параметры функции, *не является* операцией последовательного вычисления.

Порядок вычислений подвыражений внутри выражений не определён. В частности, не стоит предполагать, что выражения вычисляются слева направо.

int x = f(2) + g(3); // Неизвестно, какая функция вызовется первой – *f()* или *g()*

При отсутствии ограничений на порядок вычислений можно сгенерировать более качественный код. Однако отсутствие ограничений на порядок вычислений может привести к неопределённым результатам.

Логические операции «И» и «ИЛИ», условная операция и операция последовательного вычисления гарантируют определенный **порядок вычисления своих операндов**. Условная операция вычисляет сначала свой первый операнд, а затем, в зависимости от его значения, либо второй, либо третий операнд. Логические операции также обеспечивают вычисление своих операндов слева направо, причём логические операции вычисляют минимальное число операндов, необходимое для определения результата выражения. Таким образом, второй операнд выражения может вообще не вычисляться. Операция последовательного вычисления обеспечивает вычисление своих операндов по очереди, слева направо. Обратите внимание, что запятая в качестве указателя последовательности логически отличается от запятой, используемой в качестве разделителя параметров при вызове функций.

f1(v[i], i++); // Два параметра f2((v[i], i++)); // Один параметр

Вызов функции *f1* осуществляется с двумя параметрами *v[i]* и *i++*, и порядок вычисления параметров не определён. Расчет на определённый порядок вычисления параметров является исключительно плохим стилем и приводит к непредсказуемому поведению программы. Вызов функции *f2* имеет один параметр – последовательность выражений, разделённых запятой. Порядок вычисления гарантирован, и вызов эквивалентен *f2(i++)*.