ИНФОРМАТИКА

Лекции – C++, 51 час, среда разработки: Microsoft Visual Studio 2008

Семинарские занятия – C++, 34 часа;

Лабораторные (1 – 9 C++) : 34 часа;

АК3-11, ФН11 – Среда 10^{15}-17^{15};

РК1: Циклические процессы(6 неделя);

РК2: Матрицы, подпрограммы (11 неделя);

РК3: Динамическая память (15 неделя).

ДЗ1(3ч.): 5 неделя, **ДЗ2(2ч.)**: 10 неделя, **ДЗ3(2ч.)**: 14 неделя.

3 модуля проставляются по результатам ДЗ, РК и лабораторных (максимольно - 70 баллов)

Зачет (кафедральный) по лабораторным и ДЗ.

Экзамен (результат идет в диплом): 30 баллов

1 –Введение в информатику, C++ - начало;

2 – C++ (модульное программирование) ;

3 – Задача

Посещение всех занятий ОБЯЗАТЕЛЬНО!!

Литература

1. **Подбельский В.В. Язык C++: Учеб. пособие. – М.: Финансы и статистика, 2006.**
2. Иванова Г.С., Ничушкина Т.Н. Консольные приложения C++ в среде Microsoft Visual Studio 2008 (Visual C++): Методические указания по выполнению лабораторных работ. – М.: МГТУ им. Н.Э. Баумана, 2008. – 13 с. – В электронном виде.
3. Иванова Г.С., Ничушкина Т.Н., Самарев Р.С. . C++. Часть 1. Средства процедурно-го программирования Microsoft Visual C++ 2008: Учебное пособие. – М.: МГТУ им. Н.Э. Баумана, 2010. – 126 с. – В электронном виде.
4. Шилдт Г. Полный справочник по C++, 4 изд. – М.: Изд. дом "Вильямс", 2009. – 800 с.

Цели и задачи курса

Цель курса информатика – ознакомление с методами и средствами обработки информации и решения задач на ЭВМ. Формирование навыков программирования прикладных задач с использованием языков высокого уровня.

Задачами этого курса является изучение:

- способов представления информации
- основных сведений об ЭВМ
- инструментальных средств программирования
- основ алгоритмизации
- универсального языка программирования С++
- способы тестирования и отладки программ

1 Введение. Основные понятия

1.1 Способы представления информации в ЭВМ

Базовым понятием для всех направлений информатики является понятие **информации.**

Информация в широком смысле - это самые разнообразные сведения, сообщения, известия, знания и умения (любые виды отражения реально существующего вокруг нас реального мира).

Информация в узком смысле - это любые сведения, которые являются объектом хранения, передачи и обработки.

Информация передается в виде информационных **сообщений**.

Любое информационное сообщение может иметь произвольную физическую природу (механическую, тепловую, световую, электрическую, акустическую (символ на листе бумаги, световой сигнал, радиоволна и т.д.)

Человек принимает информацию с помощью органов чувств (слух, зрение, осязание, обоняние, вкус и т.д.) и обрабатывает ее в мозгу.

Способы представления информации в ЭВМ (2)

Информация может быть аналоговой и дискретной.

Аналоговая информация – это информация непрерывная в некотором допустимом диапазоне (температура, давление и т. д.)

Дискретная информация – это информация, которая может принимать только определенные фиксированные значения (датчик вкл. или выкл.).

Разновидностью дискретной информации является **цифровая** информация.

Вся информация в компьютере представляется в двоичном виде.

Наименьшая единица памяти называется бит, который может принимать значения 0 и 1. Бит – основной строительный блок памяти, арифметико – логического устройства и процессора.

Наименьшая адресуемая единица памяти и более удобный ее элемент – байт.

Байт состоит из 8 бит.

Так как каждый бит может принять значение 0 и 1, то 8 бит могут представить 256 (2^8) комбинаций из 0 и 1.

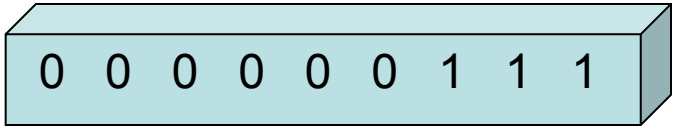
Способы представления информации в ЭВМ(3)

Для удобства обработки, чтения и записи информации байты могут объединяться в слова (2 байта), двойные слова (4 байта) и т.д.

Информация, с которой работает пользователь, бывает числовой, символьной, аудио, видео и т.д.

Для представления числовой информации используются целые и вещественные числа.

Целое число не имеет дробной части (2, -45, 789). Представив целое число в двоичном виде, его нетрудно разместить в памяти. Например, число 7 – это 111.

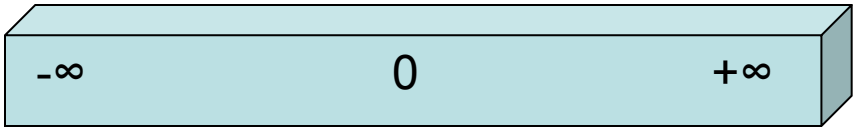


2^2 2^1 2^0

4+2+1 = 7

Способы представления информации в ЭВМ(4)

Целое число



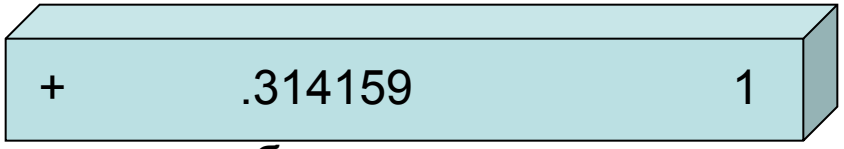
Целые числа могут быть положительными (без знака) и отрицательными (со знаком).

Для хранения знака используется один двоичный разряд (старший). Целые числа являются дискретной информацией и в машине представляются точно.

Вещественные числа – это разновидность аналоговой информации. Включают в себя числа, расположенные между целыми.

В машине такие числа представляются в двоичном виде с определенной точностью. Это связано со схемой размещения и обработки в памяти вещественного числа.

Способы представления информации в ЭВМ(5)



$+ \quad .314159 \quad \times 10^{1} \quad = \quad 3.14159$

Вещественное число представляется в форме числа с плавающей точкой. Формирование представления такого числа состоит в его разбиении на дробную часть и порядок, которые затем размещаются в памяти ($7.5 - 0.75 \times 10^{1}$).

Для размещения чисел в памяти используются двоичные числа и степени двойки вместо степеней десяти. Поэтому точно можно представить только дроби, являющиеся степенями 2.

Однако, такое разбиение дает возможность представить число несколькими способами, например 75×10^{-1}, 7.5×10^{0}, 0.75×10^{1}.

Способы представления информации в ЭВМ(6)

Символьная информация представляется двоичным кодом, который может быть не более 8 двоичных разрядов (1 байт) в соответствии с таблицей кодировки и может содержать коды 256 символов.

Так символ А представляется кодом 65,

символ 0 кодом 48,

символ 9 кодом 58,

символ е кодом 101.

1.2 Программы и алгоритмы

Основное назначение компьютера – обработка информации, для чего необходимо выполнить определенный набор операций - **программу**.

Программа – набор инструкций, описывающих последовательность действий, приводящих к результату.

Программу можно написать на машинном языке, однако это требует высокой квалификации программиста.

Для возможности написания программы пользователем непрограммистом используют специальные языки называемые языками программирования (Бэйсик, Паскаль, С и т.д.).

Программа на языке программирования преобразуется в машинные команды, которые затем выполняются компьютером.

Программы и алгоритмы (2)

Однако, чтобы составить программу, необходимо хорошо представлять себе, что нужно сделать, чтобы решить какую либо задачу.

Алгоритм – это конечная последовательность четко определенных действий, задающая обработку исходных данных с целью получения нужного результата.

1.2.1 **Свойства алгоритмов**

1. Массовость (обеспечение функций алгоритма для большой совокупности данных)
2. Дискретность (возможность представить алгоритм в виде отдельных последовательных шагов)
3. Определенность (каждый шаг алгоритма должен быть четко определен и однозначно понятен)

Свойства алгоритмов(2)

4. Результативность (получение нужного результата)
5. Конечность (выполнение алгоритма за конечное число шагов)

1.4.2 **Способы представления алгоритма**

1. Описательная форма (на естественном языке)
2. Псевдокод (описательная форма с ограниченным числом элементов)
3. Графическая форма (схема алгоритма)
4. Табличная форма (таблицы решений)

Основные конструкции псевдокода

Псевдокод:

1. Следование

…
Действие 1
Действие 2
…

2. Ветвление

…
Если Условие
    то Действие 1
    иначе Действие 2
Все-если
…

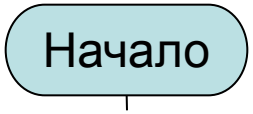
3. Цикл-пока

…
Цикл-пока Условие
    Действие
Все-цикл
…

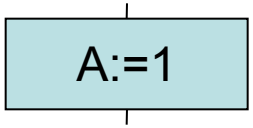
Схемы алгоритмов

Обозначения ГОСТ 19.701 – 90

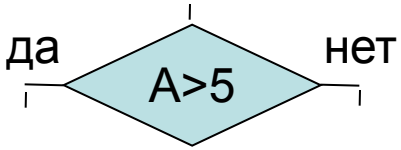
1. Терминатор (начало/конец)



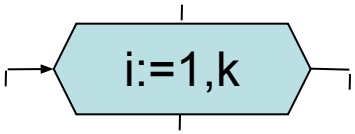
2. Процесс (вычисления)



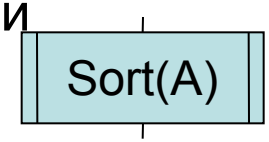
3. Анализ (проверка)



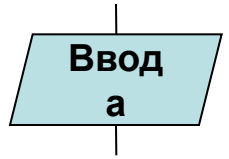
4. Модификатор (автоматическое изменение)



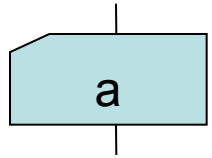
5. Предопределенный процесс (подпрограмма)



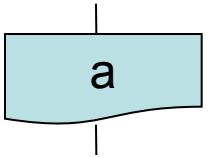
6. Ввод/вывод данных



7. Ввод с перфокарт



8. Вывод на принтер



9. Комментарий

10. Соединитель

Правила выполнения схем алгоритмов

Схемы алгоритмов должны быть выполнены аккуратно, желательно с применением карандаша и линейки или графических редакторов на компьютере.

Стрелки на линиях, идущих сверху вниз и слева направо, т. е. в направлении нашего письма не ставят, чтобы не затенять схему.

Если линия – ломанная, и направление ее хотя бы в одном сегменте не совпадает со стандартными, то стрелка ставится в конце линии, перед блоком, в который она входит.

Если схема не умещается на странице или линии многократно пересекаются, то линии разрывают. При этом один соединитель ставится в месте разрыва, второй – в месте продолжения линии. Оба соединителя помечаются одной и той же буквой или цифрой.

Для простоты чтения схемы ее начало должно быть сверху, а конец – снизу. При этом количество изгибов, пересечений и обратных направлений линий должно быть минимальным.

Таблицы решений

Таблица составляется следующим образом.

В столбик выписываются все условия, от которых зависят дальнейшие вычисления, а по горизонтали - все случаи для вычислений.

На пересечении каждого столбца и строки ставят букву Y, если для данного решения данное условие должно выполняться, букву N, если данное условие обязательно должно не выполняться, и прочерк, если исход сравнения не важен.

Например, для алгоритма вычисления корней квадратного уравнения можно составить следующую таблицу:

Таблицы решений(2)

	Нет корней	$x = -b / 2a$	$x=\pm(-b\sqrt{D}) / 2a$
$D < 0$	Y	N	N
$D = 0$	N	Y	N
$D > 0$	N	N	Y

Иногда, составленная таблица может иметь довольно сложный вид. Рассмотрим, например, таблицу:

	P1	P2	P3	P4
Условие 1	Y	-	N	Y
Условие 2	N	Y	N	N
Условие 3	Y	-	-	N

Таблицы решений(3)

Если строго придерживаться заданного порядка проверки условий, то получится довольно сложный алгоритм и его построение вызывает определенные трудности.

Но этот алгоритм можно значительно упростить, если в таблице поменять местами проверяемые условия, а также для удобства построения алгоритма поменять местами столбцы таблицы. Если преобразовать таблицу следующим образом:

	P1	P4	P3	P2
Условие 2	N	N	N	Y
Условие 1	Y	Y	N	-
Условие 3	Y	N	-	-

1.3 Основы алгоритмизации и процедурное программирование
Введение. Этапы создания ПО

1. **Постановка задачи** – неформальное описание задачи
2. **Анализ и уточнение требований** – формальная постановка задачи и выбор метода решения
3. **Проектирование** – разработка структуры ПО, выбор структур данных, разработка алгоритмов, определение особенностей взаимодействия с программной средой
4. **Реализация** – составление программ, тестирование и отладка
5. **Модификация** – выпуск новых версий

Пример разработки программы

1. **Постановка задачи**: Разработать программу, которая определяет наибольший общий делитель двух целых чисел.
2. **Анализ и уточнение требований:**

   1) *Функциональные требования*

   исходные данные: a, b – натуральные числа; 0 < a, b < **?** ;

   результат: x – натуральное число, такое, что

$x = \max \{y_i \,/\, i = \overline{1,n}\}$, где $((a \bmod y_i) = 0) \,\&\, (b \bmod y_i) = 0)$

   Метод решения:

   а) найти делители Y = { y_i } и определить x = max {Y};

   б) метод Евклида

Пример 1:

a	b
24	18
6	18
6	12
6 =	6

Пример 2:

a	b
3	4
3	1
2	1
1 =	1

Пример разработки программы (2)

2) *Эксплуатационные требования*:

а) процессор – не ниже Pentium;

б) операционная система – Windows XP (консольный режим);

в) предусмотреть запрос на ввод данных с клавиатуры;

г) результаты вывести на экран дисплея.

3) *Технологические требования*:

а) язык программирования: C++;

б) среда программирования: Microsoft Visual Studio .Net 2003;

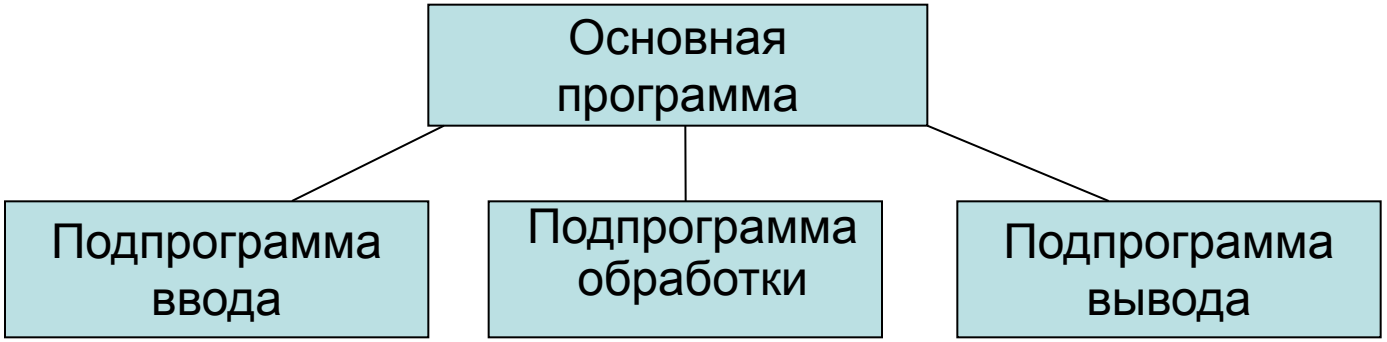
в) технология: структурный подход.

Пример разработки программы(3)

3. Проектирование

Виды проектной документации:

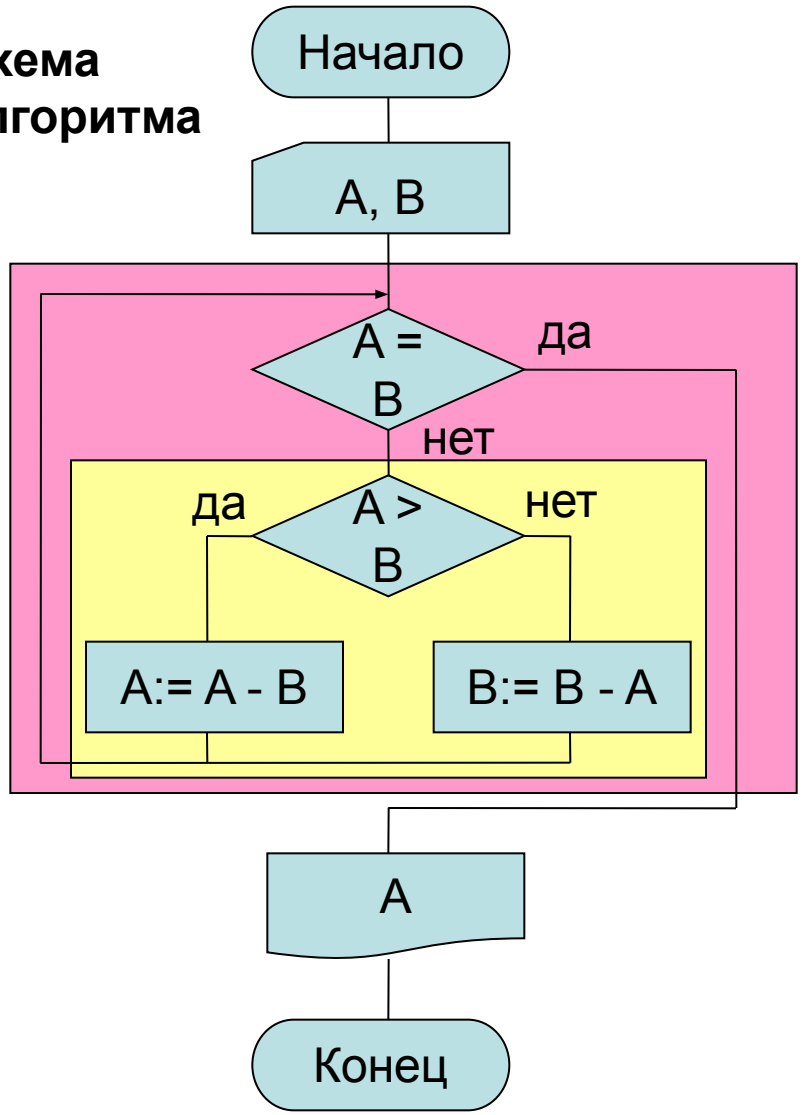
Структурная схема ПО – показывает взаимодействие по управлению основной программы и подпрограмм.



Алгоритм основной программы и подпрограмм в соответствии с выбранным способом представления

Пример разработки программы (4)

Схема алгоритма



Алгоритм на псевдокоде

Начало

Ввести A,B

Цикл-пока A ≠ B

Если A > B

то A := A – B

иначе B := B – A

Все-если

Все-цикл

Вывести A

Конец

4. Реализация программы, ее тестирование и отладка.

Схема процесса подготовки программы

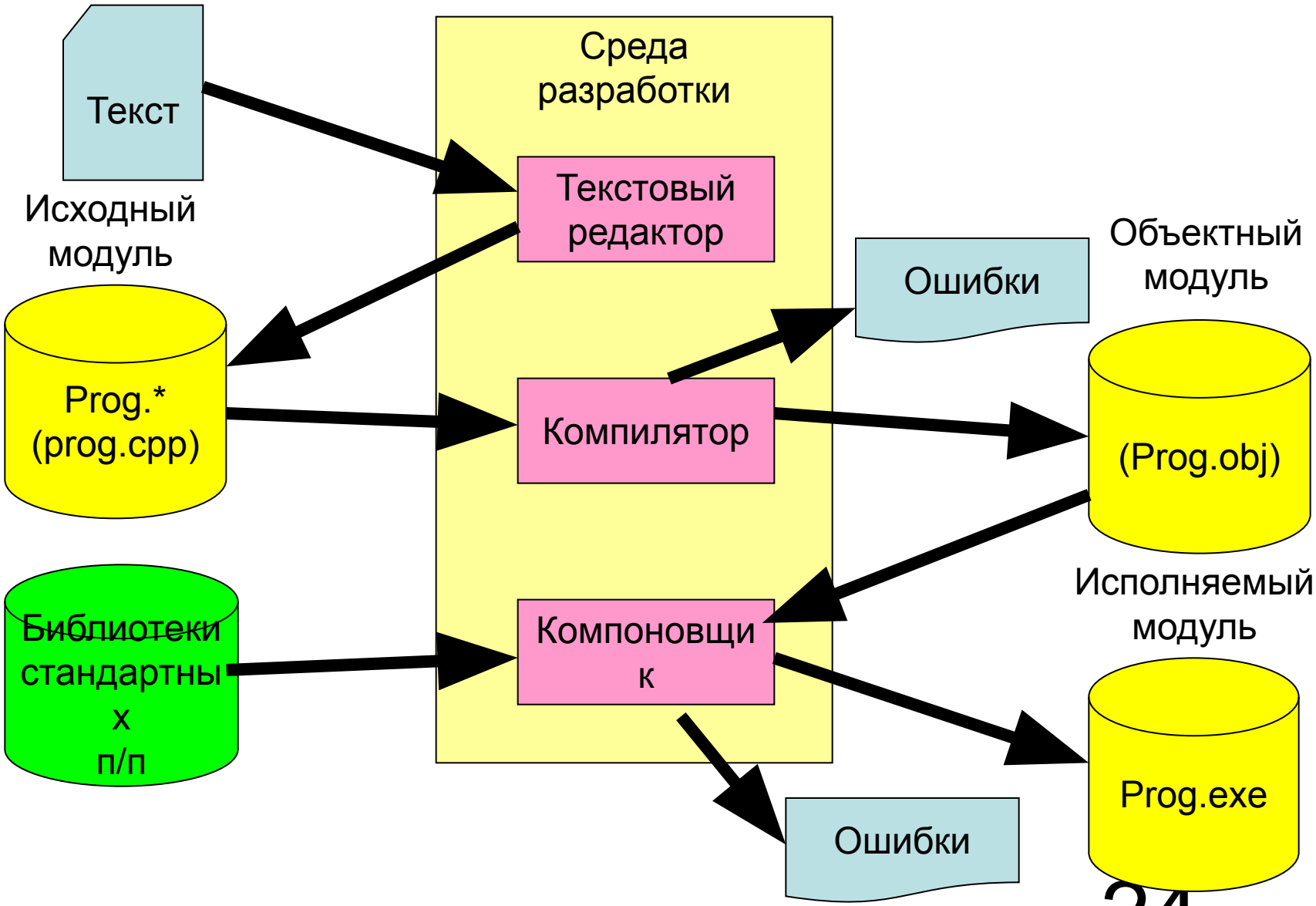
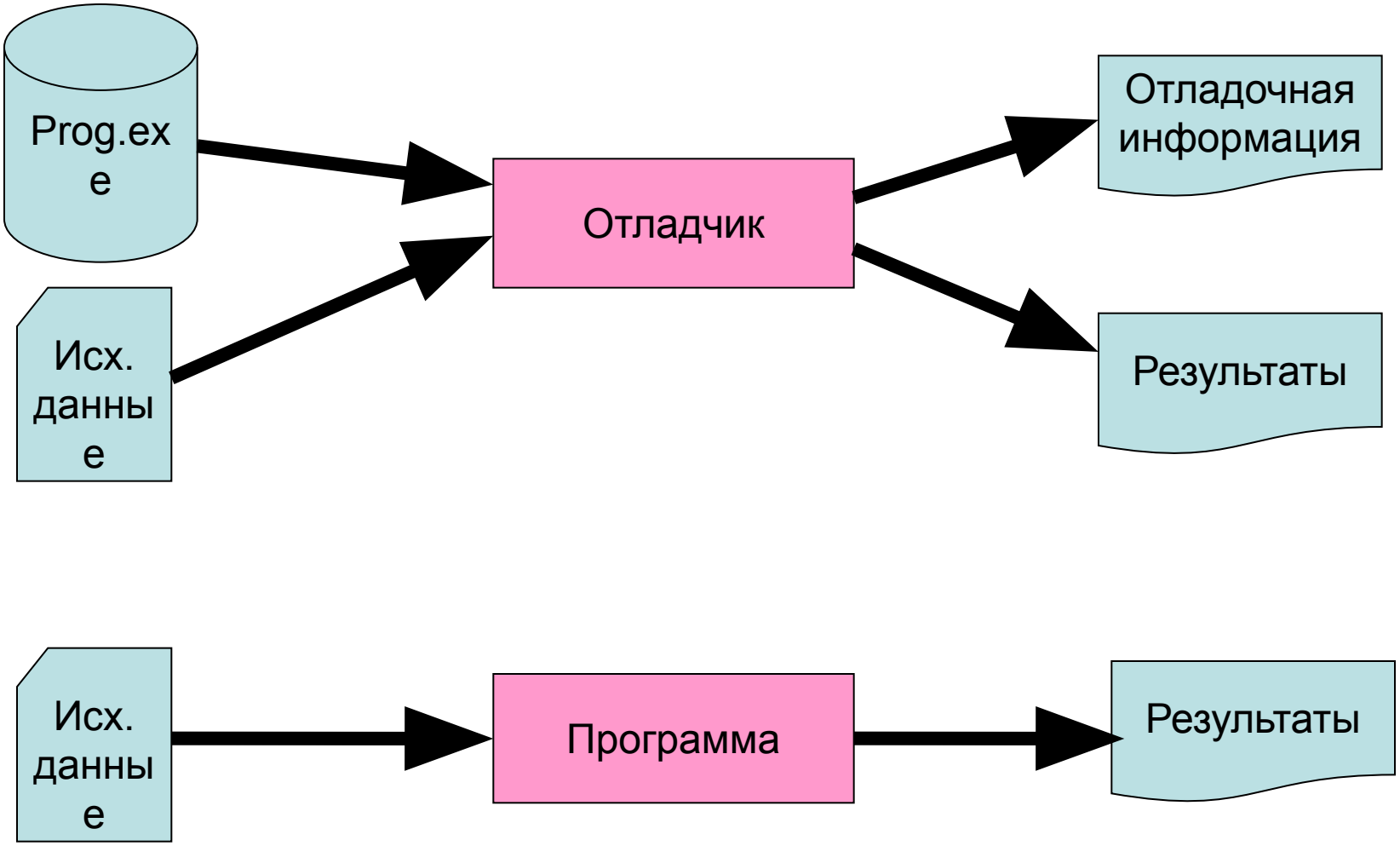
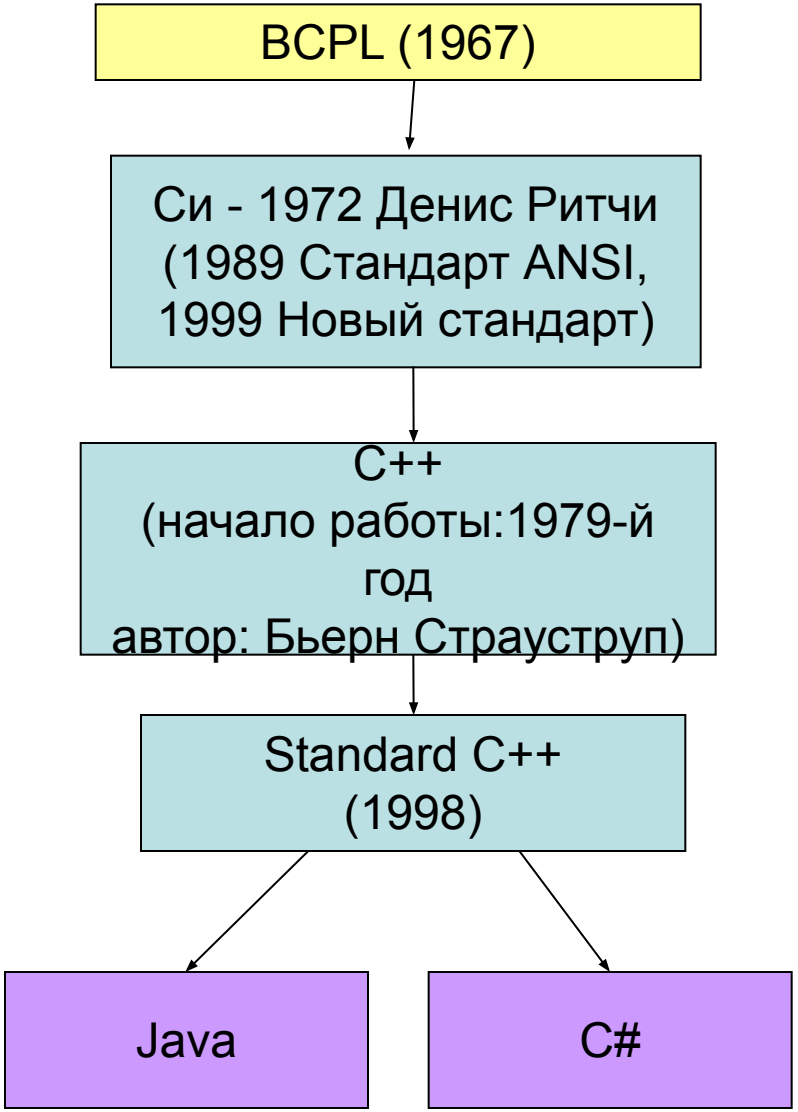


Схема процесса отладки и выполнения



1.4 Язык программирования C++



Первоначальное название «C with Classes».

Основное достоинство – наличие большого количества специальных средств и механизмов, упрощающих написание сложных системных программ.

Основной недостаток – незащищенный синтаксис, который часто не позволяет точно идентифицировать ошибку на этапе компиляции программы.

Глава 1 Простейшие конструкции языка
1.1 Алфавит и основные лексемы языка программирования

Алфавит языка C++ включает:

1) строчные и прописные буквы латинского алфавита;

2) арабские цифры: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9;

3) шестнадцатеричные цифры: 0..9, a..f или A..F;

4) специальные символы: + - * / = < & ; и т. д.;

Из символов алфавита формируются **лексемы**.

Лексема – это единица текста программы, расположенная между пробельными разделителями, которая имеет самостоятельный смысл для компилятора и не содержит в себе других лексем.

Лексемами языка C++ являются:

- идентификаторы;
- ключевые (зарезервированные) слова;
- константы;
- знаки операций;
- разделители (знаки пунктуации).

1.1.1 Идентификаторы

Идентификатор – последовательность из букв латинского алфавита, десятичных цифр и символов подчеркивания, начинающаяся не с цифры.

Прописные и строчные буквы различаются.

Примеры:

ABC abc Abc ABc AbC MY_Primer_1 Prim_123

На длину различаемой части идентификатора конкретные реализации накладывают ограничения.

Компиляторы фирмы Borland различают не более 32-х первых символов любого идентификатора.

Идентификаторы используются для обозначения имен переменных, констант, типов подпрограмм и т.д.

1.1.2 Ключевые слова

Ключевые (служебные) слова – это идентификаторы, зарезервированные в языке для специального применения. Их использование строго регламентировано.

Далее приведен список ключевых слов, предусмотренных стандартом ANSI.

auto	**do**	**for**	**return**	**switch**
break	**double**	**goto**	**short**	**typedef**
case	**else**	**if**	**signed**	**union**
char	**enum**	**int**	**sizeof**	**unsigned**
continue	**extern**	**long**	**static**	**void**
default	**float**	**register**	**struct**	**while**

В разных реализациях есть дополнительные ключевые слова, например, в Turbo C 2.0: asm, cdecl, far, pascal, const, volatile. Язык C++ добавляет еще несколько: catch, class, friend, inline,new, operator, private .

1.2 Структура программы

<Команды препроцессора>

[<Объявление типов, переменных и констант>]

[<Объявления (прототипы) функций>]

<Описание функции **main()**>

[<Описания других функций>]

Описание функции

<Тип результата или **void**> <Имя функции> ([<Список параметров>])

{[< Объявление локальных переменных и констант >]

  <Операторы>

}

С++ различает прописные и строчные буквы!

Пример программы на C++

Microsoft Visual C++ (Ex1_01)

```
#include "stdafx.h"
#include <stdio.h>



int a=18,
    b=24,
    c;

int nod(int a,int b)
{
   while (a!=b)
     if  (a>b) a=a-b;
     else b=b-a;
   return a;
}
int main()
{
   c=nod(a,b);
   printf("nod=%d\n", c);
   return 0;
}
```

Команды препроцессора

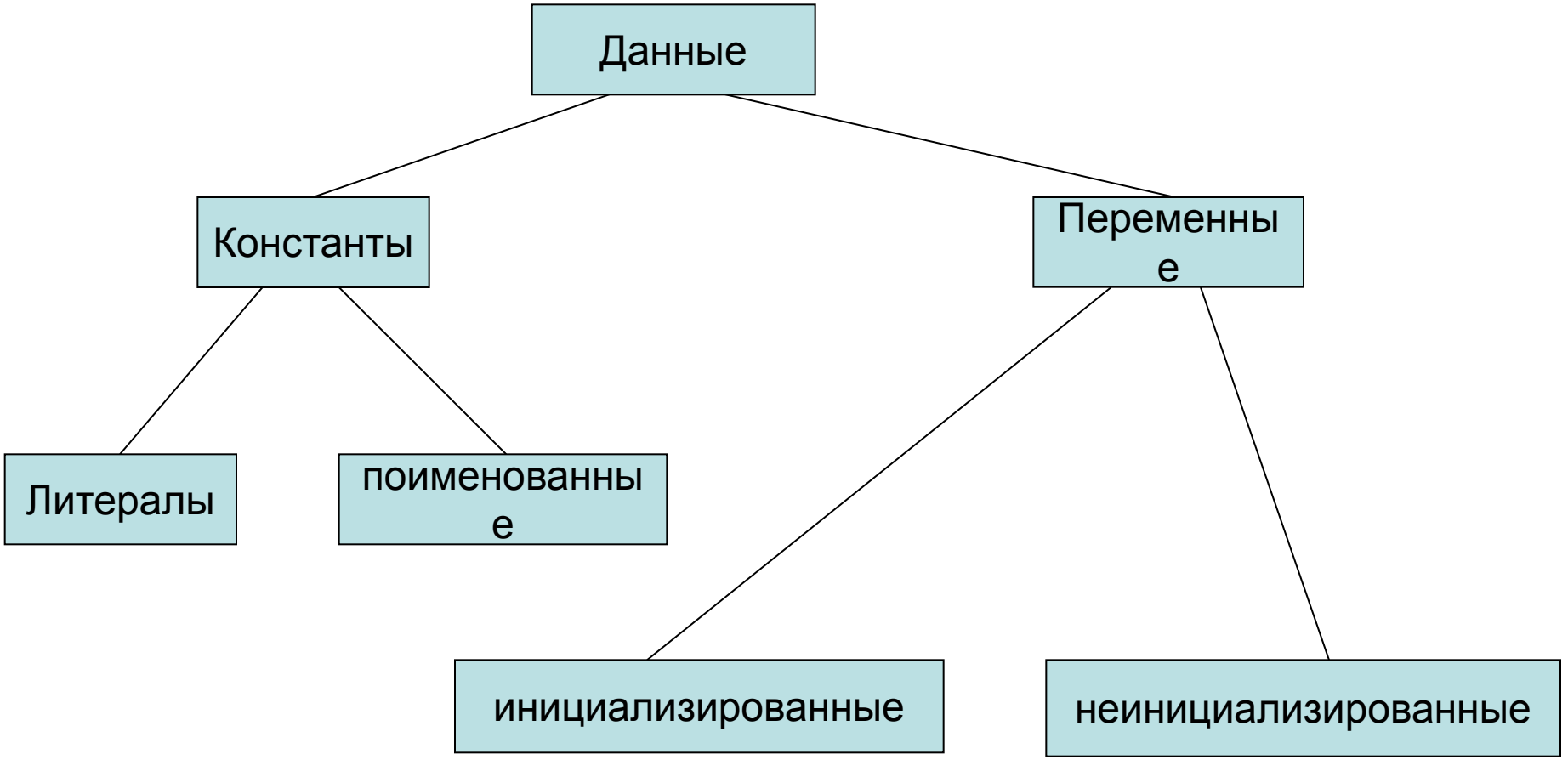
Объявление переменных

Описание функции

Основная функция

1.3 Константы и переменные

Основными объектами любой программы являются данные



1.3.1 Константы

Константы – данные, не изменяемые в процессе выполнения программы.

Поименованные константы – константы, обращение к которым выполняется по имени. Они описываются в разделе описаний.

Литералы –это лексема, представляющая изображение фиксированного числового, строкового или символьного значения, записанная в тексте программы.

Константы делятся на пять групп:

- целые,
- вещественные,
- перечислимые,
- символьные,
- строковые.

Компилятор, выделив константу, относит ее к той или другой группе по ее «внешнему виду» (по форме записи) в исходном тексте и по числовому значению.

Константы(2)

Целые константы могут быть десятичными, восьмиричными и шестнадцатиричными.

Десятичная константа определена как последовательность десятичных цифр, начинающаяся не с нуля, если это число не нуль. Может быть отрицательной и положительной.

Пример: 16, 56783, 0, -567, 7865.

Восьмиричная константа определена как последовательность последовательность десятичных цифр от 0 до 7, всегда начинающаяся с нуля. Может быть отрицательной и положительной.

Пример: 016, 020, 0777,

Шестнадцатиричная константа определена как последовательность шестнадцатиричных цифр, которая начинается сочетанием 0x. Может быть отрицательной и положительной.

Пример: 0x30, 0xF, 0xe,0x56AD.

В зависимости от значения целой константы компилятор представляет ее в памяти в соответствии с *типом*. Для явного указания способа представления программист может использовать суффиксы L,l или U,u (64L, 067u, 0x56L).

Константы(3)

Вещественные константы представлены в формате с плавающей точкой.

Константа с плавающей точкой может включать семь частей:

- целая часть (десятичная целая константа);
- десятичная точка ;
- дробная часть (десятичная целая константа) ;
- признак экспоненты (символ e или E);
- показатель десятичной степени (десятичная целая константа, возможно со знаком) ;
- суффикс F(или f) либо L(или l).

В записи вещественного числа могут опускаться целая или дробная часть (но не одновременно), десятичная точка или признак экспоненты с показателем степени, суффикс.

Пример: 66. .045 .0 3.1459F 1.34e-12 45E+6L 56.891

Без суффиксов F или L под вещественную константу отводится 8 байт.

Константы (4)

Символьные константы – это один или два символа, заключенные в апострофы.

Примеры:

‘Z’ ‘*’ ’$’ ‘\012’ ‘\0’ ‘\n’ – односимвольные константы.

‘db’ ‘\x07\x07’ ‘\n\t’ - двухсимвольные константы.

Символ ‘\’ используется для

- записи кодов , не имеющих графического изображения
- символов (‘),(\),(?),(“)
- Задания символьных констант, указывая их коды в 8-ричном или 16 -ричном виде.

Последовательность символов, начинающаяся с символа ‘\’ называется *эскейп-последовательностью*.

-

Константы (5)

Строка или строковая константа определяется как последовательность символов, заключенная в кавычки.

Пример:

“Это пример строки, называемой строковой константой”

Среди символов строки могут быть эскейп-последовательности, то есть сочетания, соответствующие неизображаемым символьным константам или символам, задаваемых их внутренними кодами. В этом случае они начинаются с символа ‘\’ .

“\nЭто строка,\nиначе -\”стринг\”,\nиначе - \”строковый литерал\”.”

Перечислимые константы по существу (по внутреннему представлению) являются обычными целыми константами, которым приписаны уникальные и удобные для использования обозначения. Будут рассмотрены далее.

1.3.2. Переменные

Переменные – поименованные данные, которые могут изменяться в процессе выполнения программы.

Переменные характеризуются именем и значением.

Именем служит идентификатор.

Переменная – это частный случай объекта как поименованной области памяти. Отличительной чертой переменной является возможность связывать с ее именем различные значения, совокупность которых определяется типом переменной.

При определении значения переменной в соответствующую ей область памяти помещается некоторый код.

Это может происходить:

- во время компиляции, тогда переменная называется инициализированной (int s=56);
- во время выполнения программы, тогда переменная называется неинициализированной (char C).

Переменная типизируется с помощью определений и описаний.

1.4 Типы данных

Тип – описатель данных, который определяет:

а) ***диапазон изменения значения***, задавая размер ее внутреннего представления;

б) ***множество операций***, которые могут выполняться над этой переменной

в) требуемое для переменной *количество памяти* при ее начальном распределении

г) *интерпретацию двоичного кода* значений при последующих обращениях к переменным.

Кроме того, тип используется для контроля типов с целью обнаружения возможных случаев недопустимого присваивания.

В C++ стандартно определено большое количество типов, которые программист может использовать без предварительного описания.

1.4.1 Фундаментальные типы данных
1. Интегральные типы

Имя типа	Подтипы	Размер , байт	Интервал значений
char или **_int[8]**	**[signed] char** **unsigned char**	1	-128..127 0..255
short или **_int[16]**	**[signed] short** **unsigned short**	2	-32768..32767 0..65535
[int] или **long** или **_int[32]**	**[signed] [int]** **unsigned [int]** **[signed] long** **unsigned long**	4	-2^{31}.. $2^{31}-1$ 0.. $2^{32}-1$
long long или **_int[64]**	**[signed] long long** **Unsigned long long**	8	-2^{63}.. $2^{63}-1$ 0.. $2^{64}-1$
bool		1	false (0), true(1)

Примечание – Для совместимости считается: 0 – false; не 0 – true.

Фундаментальные типы данных (2)

2. Вещественные типы

Тип	Размер, байт	Значащих цифр	Минимальное положительное число	Максимальное положительное число
float	4	6	1.175494351e-38	3.402823466e38
double (long double)	8	15	2.2250738585072014 e-308	1.797693134862318 e308

3. Неопределенный тип void

Нельзя объявлять значения типа void, он используется только при объявлении

- нетипизированных указателей;
- функций, не возвращающих значений (процедур).

1.5 Объявление переменных и поименованных констант

[<Изменчивость>] [<Тип>]<Список идентификаторов> [=<Значение>];

где <Изменчивость> – описатель возможности изменения значений:

const – поименованная константа,

volatile – переменная, меняющаяся в промежутках между явными обращениями к ней

без указания изменчивости – обычная переменная

<Тип> – описатель типа: **int**, **char**, **float**, **double** и т.д.;

<Список идентификаторов> – список имен переменных или констант;

<Значение> – начальное значение переменной или значение константы.

Примеры объявлений переменных и констант

Неинициализированные переменные:

int f,c,d; float r;

I,j;unsigned int max,min;

char c1,c2; unsigned char c5;

Инициализированные переменные

double k=89.34; char ch='G';

Поименованные константы

const long a=6; const float pp=6.6e-34;

На практике все объявления могут быть перемешаны в описаниях программы:

const char simt='T';float max=100,min=-100;

double f,s,eps=0.001;

Переменные и поименованные константы могут быть объявлены в любом месте программы:

вне всех функций, внутри функций, в любом месте функции.

Основное условие – объявление должно стоять до обращения к переменной или константе.

1.5.1. Перечисляемый тип

Используется для объявления набора поименованных целых констант.

Формат:

enum {<Ид>[=<Целое>] [,<Ид>[<>]…]}

<Список переменных>;

Пример:

```
enum {SUN, MON, TUES, FRI=5, SAT} day;
```

Имя переменной

SUN =0, MON = 1, TUES = 2, FRI=5, SAT=6

Константы присваиваются, начиная с нуля или с указанного значения.

1.6 Объявление типа пользователя

typedef <Описание типа> <Имя объявляемого типа>;

Примеры:

Имя нового типа

1) `typedef unsigned int word;`

2) `typedef enum {false, true} boolean;`

Имя нового типа

1.7 Выражения

Выражение – это последовательность операндов, разделителей и знаков операций, задающая вычисление

Выражение есть правило для получения значения.

В качестве операндов могут выступать константы, переменные, стандартные функции, определенные в языке.

Порядок операций определяется рангами (приоритетами) и правилами их группирования (ассоциативностью).

Для изменения порядка выполнения операций используются круглые скобки.

Операции делятся на

- унарные;
- бинарные.

Бинарные могут быть:

- аддитивные;
- мультипликативные;
- сдвиговые;
- операции присваивания
- поразрядные;
- операции отношения
- логические

1.8 Операции

Унарные операции

Выполняются над одним операндом

- унарный минус - меняет знак арифметического операнда;

+ унарный плюс - введен для симметрии с унарным минусом;

! логическое отрицание;

& операция получения адреса операнда

* обращение по адресу (операция разыменования)

Порядковые*:*

++<идентификатор>, <идентификатор>++ (следующее);

- -<идентификатор>, <идентификатор> - - (предыдущее).

Местоположение знаков операций определяет в какой момент осуществляется изменение операнда.

Если знак стоит слева от операнда – то сначала значение изменяется, а потом принимает участие в вычислении.

Если знак стоит справа от операнда – то сначала операнд принимает участие в вычислении, а затем меняется его значение. (i++; a*++i; --i+c; c*i--)

Операции(2)

БИНАРНЫЕ

Аддитивные: **+**, **-**,

Мультипликативные:

* - умножение, если операнды целые, то результат целый;

/ - если делимое и делитель - целые, то результат - целое ,

% - остаток от деления целых чисел.

Пример:

```
int a=5;int b = 3; float c=9.3
```

…

```
a+b
```
8

```
a / b
```
1

```
a % b
```
2

```
a*b
```
15

```
c / b
```
3.1

```
(a+b)/(a-b*a)
```

Операции (3)

2. **Операции отношения** – применяют к числам, символам– в результате получают логическое значение:

<, >, ==, !=, <=, >= результат операций отношения – это истина или ложь

В C++ истина – это не 0 (true)

ложь - это 0 (false)

Пример:

```
int a = 5;  int b = 3;
```

…

```
a > b
```

не 0

```
a == b
```

0

Операции(4)
Логические

&& - конъюнкция (и) арифметических операндов или операций отношений. Результат целочисленный 0 (ложь) или не 0 (истина).

|| - дизъюнкция (или) арифметических операндов или отношений. Результат целочисленный 0 (ложь) или не 0 (истина).

(к логическим операциям относится и унарная операция ! -отрицание).

Чаще всего операндами логических операций являются условные выражения.

Логические выражения:

выражение1&&выражение2 – истинно только тогда, когда оба выражения истинны;

выражение1||выражение2 – истинно, хотя бы одно из выражений истинно;

!выражение - истинно, если выражение ложно, и наоборот.

6>2&&3==3 - истина

!(6>2&&3==3) - ложь

x !=0 && 20/x<5 - второе выражение вычисляется, если x!=0.

Операции (5)

Логические поразрядные

& (и) - поразрядная конъюнкция (и) битовых представлений значений целочисленных выражений,

| (или) поразрядная дизъюнкция (или) битовых представлений значений целочисленных выражений,

^ (исключающее или) поразрядная исключающая или битовых представлений значений целочисленных выражений.

Примеры:

6&5 - 4 00000110 & 00000101 □ 00000100

6|5 - 7 00000110 | 00000101 --> 00000111

6^5 -3 00000110 ^ 00000101 □ 00000011

Операции (6)

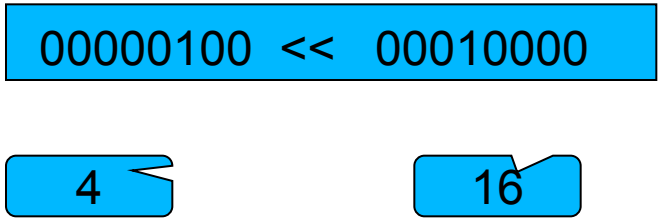
Операции сдвига

>> сдвиг вправо битового представления значения левого целочисленного операнда на количество разрядов, равное значению правого целочисленного операнда,

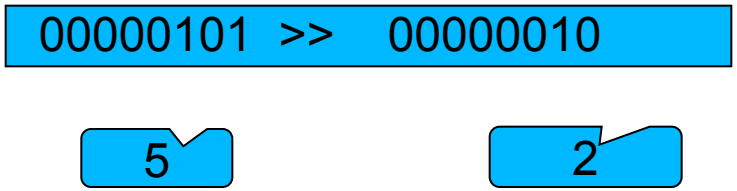
<< сдвиг влево битового представления значения левого целочисленного операнда на количество разрядов, равное значению правого целочисленного операнда.

Примеры:

4<<2 16



5>>1 2



Операции(7)

Операции присваивания

В C++ присваивание относится к операциям и используется для формирования бинарных выражений. Поэтому в C++ отсутствует отдельный оператор присваивания.

В качестве левого операнда в операциях присваивания может использоваться только переменная.

= += -= *= /= %= &= ^= |= <<= >>=

= - присваивает левому операнду значение выражения правой части;

Остальные операции присваивают левому операнду результат выполнения операции, указанной слева от операции равно, левого операнда и правого.

Примеры:

Int k;

k=35/4;

8

k*=5-2;

24

k+=21/3;

31

Операции(8)

Условная операция

Единственная операция, которая выполняется над тремя операндами

выражение_1 ? Выражение_2 : выражение_3

Первым вычисляется значение **выражения_1**.

Если оно истинно, т.е. не равно 0, то вычисляется **выражение_2**, которое становится результатом.

Если при вычислении выражения_1 получится 0, то вычисляется **выражение_3**, которое становится результатом.

Примеры:

```
x < 0 ? -x : x;
```

```
printf("%3d%c%",a,i==n?' ':'\n');
```

Операции (9)

Запятая, как разновидность операции

В C++ несколько выражений могут быть записаны через запятую.

Выражения, разделенные запятой выполняются последовательно слева направо.

<Выражение1>,<Выражение2>,...<Выражение n>

В качестве результата сохраняется тип и значение самого правого выражения.

Примеры:

```
int m=5,z;
z=(m=m*5,m*3);
int d,k;
k=(d=4,d*8);
```

m=25, z=75

d=4, Результат к=32

В C++ круглые и квадратные скобки также играют роль бинарных операций (обращение к функциям, обращение к элементам массива и т.д.)

Приоритет операций

1. **( ) [ ] -> :: .**
2. **!** (не) **+ - ++ -- &(**адрес**) *(**указатель**) sizeof new delete**
3. **.* ->***
4. *** / %**
5. **+ -** (бинарные)
6. **<< >>**
7. **< <= > >=**
8. **= = ! =**
9. **&**(поразрядное и)
10. **^**(исключающее или)
11. **|** (поразрядное или)
12. **&&**
13. **||**
14. **?:**
15. **= *= /= %= += -= &= ^= |= <<= >>=**
16. **,**

Примеры выражений

а) **int a=10, b=3; float ret; ret=a/b;** ret=3

б) **c=1; b=c++;** b=1, c=2

в) **c=1; sum=++c;** c=2, sum=2

г) **c=a<<4;** эквивалентно **c=a*16;**

д) **a+=b;** эквивалентно **a=a+b;**

е) **a=b=5;** эквивалентно **b=5; a=b;**

ж) **c=(a=5, b=a*a);** эквивалентно **a=5; b=a*a; c=b;**

з) **a=(b=s/k)+n;** эквивалентно **b=s/k; a=b+n;**

и) **c=(a>b)?a:b;** если a>b, то c=a, иначе c=b

Математические функции

В выражениях можно использовать следующие математические функции из библиотеки **<math.h>** :

fabs(< вещественное выражение>) // абс. значение
abs(<Целое выражение>) // абс. значение
sqrt(<Вещественное выражение>) // $\sqrt{x}$
exp(<Вещественное выражение>) // e^x
log(<Вещественное выражение>) // ln x
log10 (< Вещественное выражение >) // log10(x)
sin(<Вещественное выражение>)
cos(<Вещественное выражение>)
atan(<Вещественное выражение>) // arctg x
tan(< Вещественное выражение >) // tg x
acos (< Вещественное выражение >) // арккосинус
asin (< Вещественное выражение >) // арксинус
sinh(<Вещественное выражение>) // гиперболический синус
cosh(<Вещественное выражение>) //гиперболический косинус

Библиотека **<conio.h>**

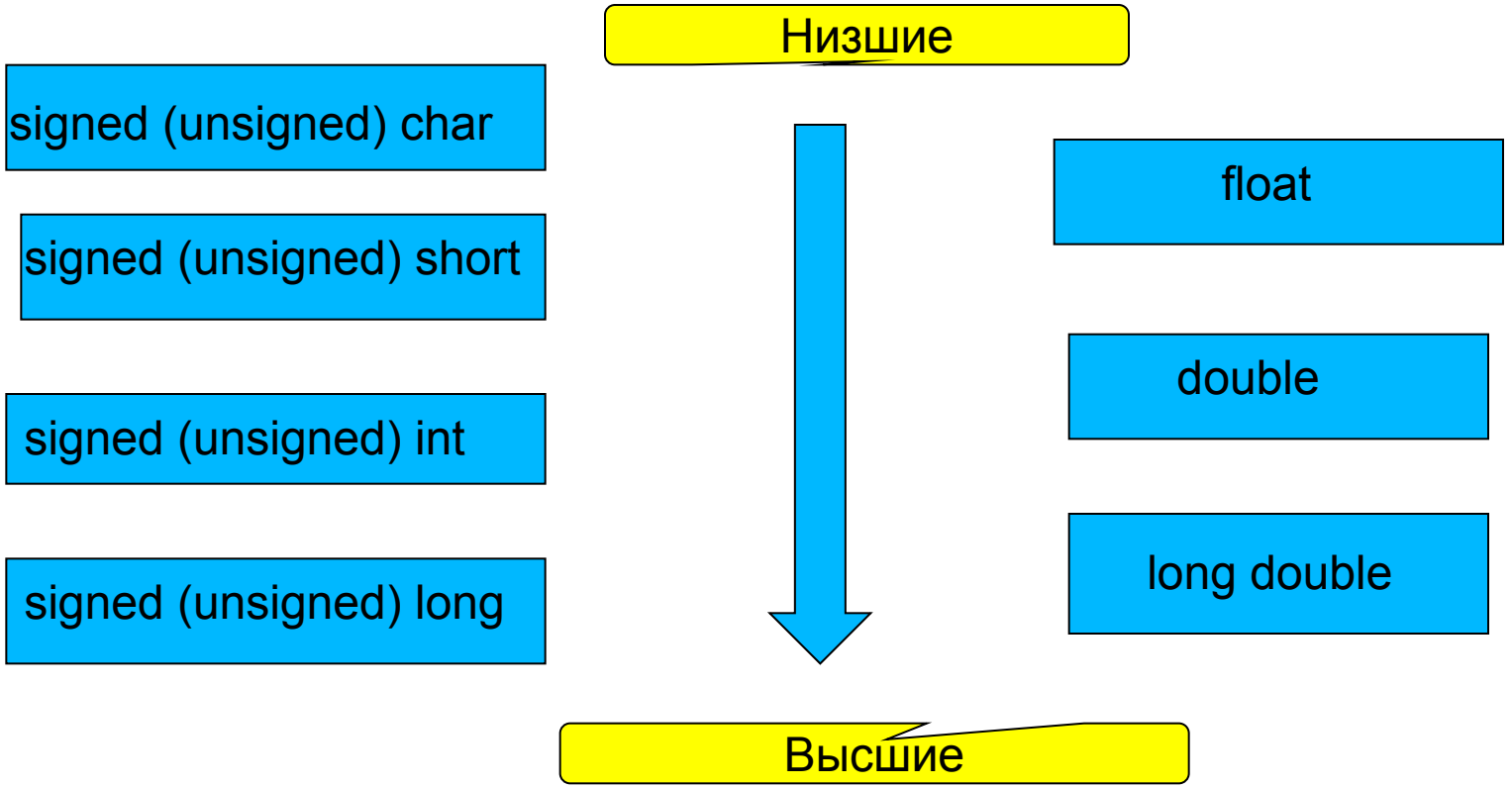
rand () – генерация случайного числа $0 \le x < 2^{15}-1$;
srand (<Ц. выр. >) – инициализация генератора случайных чисел;

Правила вычисления выражений

При вычислении выражений некоторые операции требуют , чтобы операнды были соответствующего типа. Если это требование не выполняется – осуществляется стандартное принудительное ***неявное преобразование типов***.

Стандартное преобразование включает преобразование «низших» типов к «высшим».

Такое преобразование гарантирует сохранение значимости.



Правила вычисления выражений (2)

Для выполнения операций над некоторыми типами данных требуется *явное переопределение типов.*

Различают:

Функциональное преобразование

<имя типа> (Список выражений)

Примеры:

```
int(3.14); float(2/3);  int('A');
```

Однако, функциональная запись не подходит для сложного типа.

В этом случае применяется **каноническая** форма преобразования:

(имя типа)<выражение>

Примеры:

```
(unsigned long)(x/3+2); (long)25;(char)123;
```

Если ввести новый тип – тогда можно использовать и функциональное преобразование

```
typedef unsigned long int uli;
uli(x/3-123);
```