Процедурное программирование

Для курса _1_ групп_____

По направлениям подготовки 09.03.01, 09.04.01, 15.03.04

Москва, 2017

Содержание

13. Простая программа на C++
14. Переменные и типы данных в C++
15. Ветвления в C++
16. Циклы в C++
17. Массивы в C++
18. Функции в C++
19. Указатели в C++
20. Динамические массивы в C++
21. Параметры командной строки.
22. Литература

АЛГОРИТМ — это точно определенная последовательность действий, которые необходимо выполнить над исходной информацией, чтобы получить решение задачи.

Свойства алгоритма: результативность, реалистичность, достоверность, массовость, детерминированность.

Способы записи алгоритма: словесный, формульный, табличный, операторный, графический

Алгоритм и его особенности

РЕЗУЛЬТАТИВНОСТЬ — алгоритм должен давать конкретное конструктивное решение, а не указывать на возможность решения вообще.

ДОСТОВЕРНОСТЬ — алгоритм должен соответствовать сущности задачи и формировать верные, не допускающие неоднозначного толкования решения.

РЕАЛИСТИЧНОСТЬ — возможность реализации алгоритма при заданных ограничениях: временных, программных, аппаратных.

МАССОВОСТЬ — алгоритм должен быть воспроизводимым, пригодным для решения всех задач определенного класса на всем множестве допустимых значений исходных данных.

ДЕТЕРМИНИРОВАННОСТЬ (определенность) — алгоритм должен содержать набор точных и понятных указаний, не допускающих неоднозначного толкования.

Способы задания алгоритма

СЛОВЕСНЫЙ способ записи содержит последовательные этапы алгоритма и описывается в произвольной форме на естественном языке;

ФОРМУЛЬНЫЙ способ основан на строго формализованном аналитическом задании необходимых для исполнения действий;

ТАБЛИЧНЫЙ способ подразумевает отображение алгоритма в виде таблиц, использующих аппарат реляционного исчисления и алгебру логики для задания подлежащих исполнению взаимных связей;

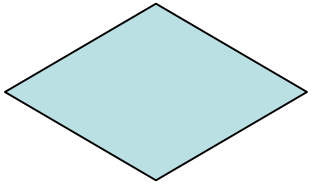
ОПЕРАТОРНЫЙ способ базируется на использовании для отображения алгоритма условного набора специальных операторов: арифметических, логических, печати и т. д.;

ГРАФИЧЕСКОЕ отображение алгоритмов в виде блок-схем — самый распространенный способ.СС

Элементы блок-схем алгоритмов

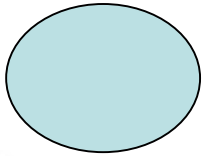
- Ввод/вывод

- Начало/конец



- Условие

- Процесс



- Соединитель

Блок-схемы алгоритмов

- ГОСТ 19.701-90
- http://cert.obninsk.ru/gost/282/282.html
- https://prog-cpp.ru/block-schema/
- https://sites.google.com/site/anisimovkhv/learning/pris/lecture/tema8/tema8_2

Программирование

Программирование – это написание текста на алгоритмическом языке.

Цель программирования – описание процесса обработки данных.

Система программирования – это язык и средства разработки программ.

В состав интегрированной среды разработки программ входят:

- текстовый редактор;
- транслятор;
- редактор связей;
- библиотеки подпрограмм;
- программа-отладчик;
- система помощи и подсказок.

Язык программирования (ЯП) — формальная знаковая система, предназначенная для описания алгоритмов в форме, которая удобна для исполнителя (например, компьютера).

ЯП определяет набор лексических, синтаксических и семантических правил, используемых при составлении компьютерной программы. Он позволяет программисту точно определить то, как будут храниться и передаваться данные, а также какие именно действия следует выполнять над этими данными при различных обстоятельствах.

Общие особенности языков программирования

- **ЯП предназначен для написания компьютерных программ, которые применяются для передачи компьютеру инструкций по выполнению вычислительного процесса и организации управления отдельными устройствами.**
- **ЯП отличается от естественных языков, тем что предназначен для передачи команд и данных от человека компьютеру, в то время, как естественные языки используются лишь для общения людей между собой.**
- **ЯП может использовать специальные конструкции для определения и манипулирования структурами данных и управления процессом вычислений.**

Виды алгоритмических языков: машинно-ориентированные, процедурно-ориентированные, проблемно-ориентированные.

Машинно-ориентированные языки программирования низкого уровня — программирование на них наиболее трудоемко, но позволяет создавать оптимальные программы, максимально учитывающие функционально-структурные особенности конкретного компьютера.

Процедурно-ориентированные и проблемно-ориентированные языки относятся к языкам высокого уровня, использующим макрокоманды. Макрокоманда при трансляции генерирует много машинных команд.

Языки программирования делятся на два класса — компилируемые и транслируемые.

- **Программа на компилируемом языке при помощи специальной программы компилятора преобразуется (компилируется) в набор инструкций для данного типа процессора (машинный код) и далее записывается в исполняемый файл, который может быть запущен на выполнение как отдельная программа (компилятор переводит программу с языка высокого уровня на низкоуровневый язык, понятный процессору).**

- **Программа на интерпретируемом языке, то интерпретатор непосредственно выполняет (интерпретирует) ее текст без предварительного перевода. При этом программа остается на исходном языке и не может быть запущена без интерпретатора.**

Трансляторы – это программы, которые переводят текст с языка программирования на машинный язык.

Трансляторы бывают двух видов: трансляторы-компиляторы и трансляторы-интерпретаторы

КОМПИЛЯТОРЫ при трансляции переводят на машинный язык сразу всю программу и затем хранят ее в памяти машины в двоичных кодах.

ИНТЕРПРЕТАТОРЫ каждый раз при исполнении программы заново преобразуют в машинные коды каждую макрокоманду и передают ее для непосредственного выполнения компьютеру.

Компилятор переводит программу на машинный язык сразу и целиком, создавая при этом отдельную программу, а интерпретатор переводит на машинный язык прямо во время исполнения программы

Разделение на компилируемые и интерпретируемые языки является условным:

- для любого интерпретируемого языка можно создать компилятор
- создаваемый во время исполнения программы код может так же динамически компилироваться во время исполнения.

Особенности скомпилированных программ

- выполняются быстрее и не требуют для выполнения дополнительных программ, так как уже переведены на машинный язык
- при каждом изменении текста программы требуется ее перекомпиляция
- скомпилированная программа может выполняться только на том же типе компьютеров и, как правило, под той же операционной системой, на которую был рассчитан компилятор

Особенности интерпретируемых программ

- интерпретируемые программы выполняются заметно медленнее, чем компилируемые
- программы можно запускать сразу же после изменения, что облегчает разработку
- программа на интерпретируемом языке может быть зачастую запущена на разных типах машин и операционных систем без дополнительных усилий
- не могут выполняться без дополнительной программы-интерпретатора.

Некоторые языки, например, Java и C#, находятся между компилируемыми и интерпретируемыми:

- программа компилируется не в машинный язык, а в машинно-независимый код низкого уровня, байт-код
- байт-код выполняется виртуальной машиной.

Для выполнения байт-кода обычно используется интерпретация, хотя отдельные его части для ускорения работы программы могут быть транслированы в машинный код непосредственно во время выполнения программы по технологии компиляции «на лету» (Just-in-time compilation, JIT).

Классификация языков программирования по близости к естественному языку

- Языки программирования высокого уровня
- Языки программирования низкого уровня

Высокоуровневый язык программирования — разработанный для быстроты и удобства использования программистом.

Слово «высокоуровневый» здесь означает, что язык предназначен для решения абстрактных высокоуровневых задач и оперирует не инструкциями к оборудованию, а логическими понятиями и абстракцией данных.

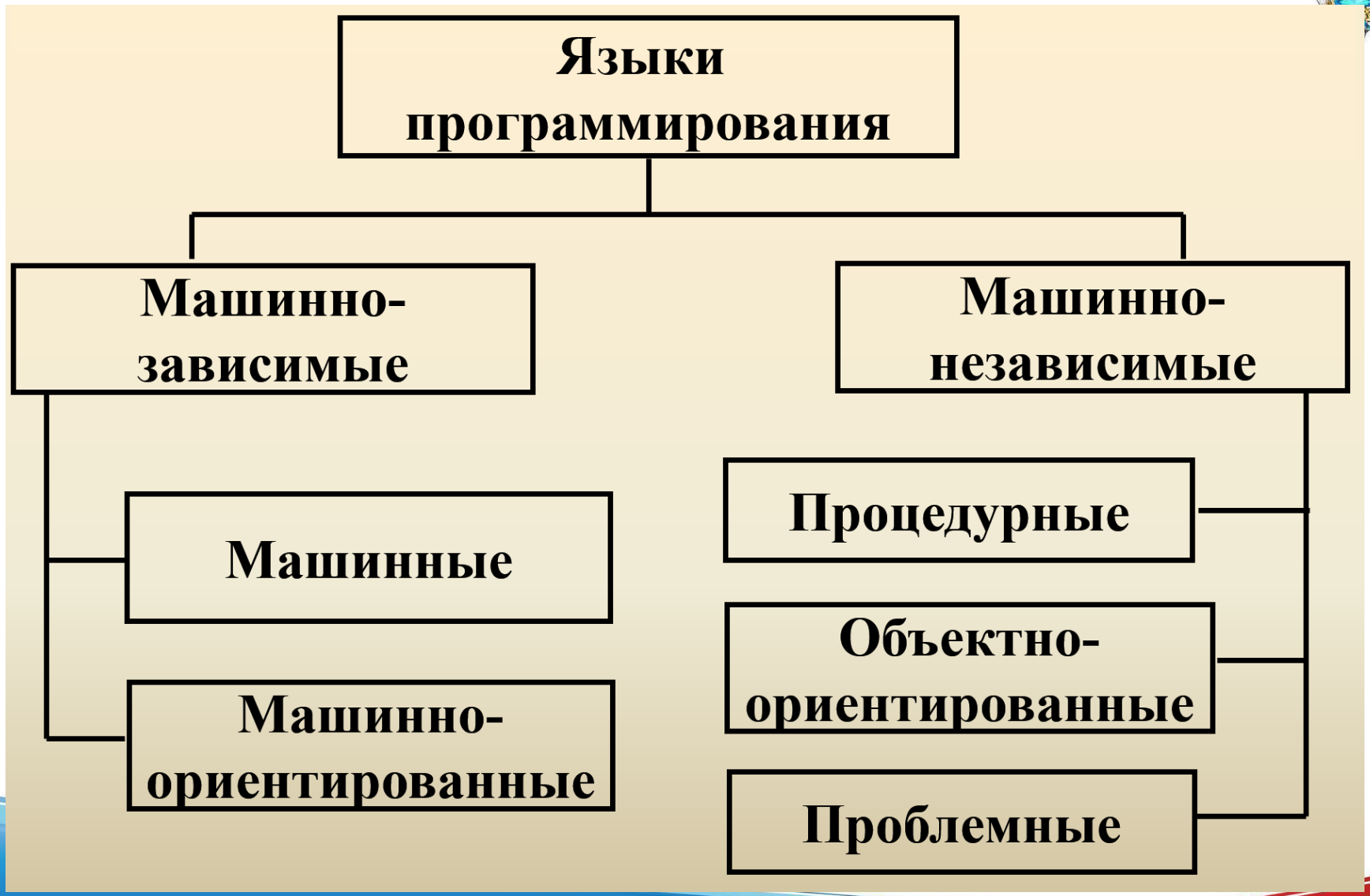
(C++, Visual Basic, Java, Python, Ruby, Perl, Delphi, PHP)

Программы проще для понимания программистом, но гораздо менее эффективны, чем создаваемые при помощи низкоуровневых языков.

Низкоуровневый язык программирования — язык программирования, близкий к программированию непосредственно в машинных кодах.

Как правило, использует особенности конкретного семейства процессоров. Общеизвестный пример низкоуровнего языка — язык ассемблера

Классификация языков программирования



Языки программирования

- Машинный язык – это система команд компьютера.
- Машинно-ориентированные языки – это мнемокоды, автокоды, языки ассемблера.
- Проблемные языки направлены на решение узкого круга задач.
- Процедурные языки – это машинно-независимые языки для описания алгоритмов решения задачи.
- Объектно-ориентированные языки основаны на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного типа (класса).

Структурное программирование

Это программирование с разбиением на части сложных систем для последующей реализации в виде отдельных подпрограмм.

Примеры: PL/1, Pascal, Си

Модульное программирование

Оно предполагает выделение групп подпрограмм, использующих одни и те же глобальные переменные, в отдельные модули (библиотеки подпрограмм).

Примеры: Pascal, Си (C++), Ada, Modula

Процедурное программирование —
основанное на концепции вызова процедуры (подпрограммы, методы или функции).

Процедуры просто содержат последовательность шагов для выполнения. В ходе выполнения программы любая процедура может быть вызвана из любой точки, включая саму данную процедуру.

Отличительные особенности:

- возможность повторного использования одного и того же кода из нескольких мест программы без его копирования;
- возможность написания программы без инструкций GOTO или JUMP (спагетти-код);
- возможность поддержки модульности и структурности.

(Ада, Бейсик, Си, C++, C# (из Microsoft), ColdFusion, КОБОЛ, Delphi, JavaScript, JScript, Forth, Фортран, Java, Модула-2, Оберон, Глагол, Паскаль, Перл, ПЛ/1, Рапира, Visual Basic, REXX, PHP)

Команда машинной программы (иначе, машинная команда) — это элементарная инструкция машине, выполняемая ею автоматически без каких-либо дополнительных указаний и пояснений. Машинная команда состоит из двух частей: операционной и адресной.

Операционная часть команды (КОП — код операции) — это группа разрядов в команде, предназначенная для представления кода операции машины.

Адресная часть команды (Адреса) — это группа разрядов в команде, в которых записываются коды адреса (адресов) ячеек памяти машины, предназначенных для оперативного хранения информации, или иных объектов, задействованных при выполнении команды.

Адреса команды делятся на: безадресные, одно-, двух- и трехадресные

ТРЕХАДРЕСНЫЕ - a1 — адрес ячейки (регистра), куда следует поместить число, полученное в результате выполнения операции.

ДВУХАДРЕСНЫЕ - a1 — это обычно адрес ячейки (регистра), где хранится первое из чисел, участвующих в операции.

ОДНОАДРЕСНЫЕ - a1 в зависимости от модификации может обозначать либо адрес ячейки (регистра), либо адрес ячейки (регистра) куда следует поместить число.

БЕЗАДРЕСНЫЕ - содержат только код операции, а информация должна быть заранее помещена в определенные регистры машины.

Виды машинных команд по выполняемым операциям:

1) операции пересылки информации внутри компьютера,
2) арифметические операции над информацией,
3) логические операции над информацией,
4) операции над строками (текстовой информацией),
5) операции обращения к внешним устройствам компьютера,
6) обслуживающие и вспомогательные операции,
7) операции передачи управления.

Операции передачи управления (или иначе — ветвления программы), которые служат для изменения естественного порядка выполнения команд.

Виды передачи управления: БЕЗУСЛОВНАЯ И УСЛОВНАЯ

Операции БЕЗУСЛОВНОЙ ПЕРЕДАЧИ управления всегда приводят к выполнению после данной команды не следующей по порядку, а той, адрес которой в явном или неявном виде указан в адресной части команды.

Операции УСЛОВНОЙ ПЕРЕДАЧИ управления тоже вызывают передачу управления по адресу, указанному в адресной части команды, но только в том случае, если выполняется некоторое заранее оговоренное для этой команды условие.

Адресация операндов в командах программы:

1) непосредственная,

2) прямая,

3) косвенная,

4) ассоциативная,

5) неявная.

Непосредственная адресация заключается в указании в команде самого значения операнда, а не его адреса.

Прямая адресация состоит в указании в команде непосредственно абсолютного или исполнительного адреса операнда.

Косвенная адресация имеет в виду указание в команде регистра(ов) или ячейки памяти, в которых находятся абсолютный, исполнительный адрес операнда или их составляющие.

Ассоциативная адресация — указание в команде не адреса, а идентифицирующего содержательного признака операнда, подлежащего выборке.

Неявная адресация — адреса операнда в команде не указано, но он подразумевается кодом операции.

ОТНОСИТЕЛЬНАЯ АДРЕСАЦИЯ

Абсолютный (Аинд) адрес формируется как сумма адресов исполнительного (Аисп) и сегментного (Асегм): Aabc= Асегм+ Аисп;

Асегм=16·А′сегм ;

Аисп= [Асмещ]+[Абаз]+[Аинд]. При адресации данных могут использоваться все составляющие адреса:

Aabc=Асегм+[Асмещ]+[Абаз]+[Аинд].

При адресации команд программы могут использоваться только две составляющие адреса:

Aabc=Асегм+[Асмещ]=16·А′сегм + Асмещ .

РЕЖИМЫ РАБОТЫ КОМПЬЮТЕРА

ОДНОПРОГРАММНЫЙ РЕЖИМ использования компьютера самый простой, применяется во всех поколениях компьютеров. Из современных машин этот режим чаще всего используется в персональных компьютерах.

МНОГОПРОГРАММНЫЙ РЕЖИМ обеспечивает лучшее расходование ресурсов компьютера, но несколько ущемляет интересы пользователя. Для реализации этого режима необходимо прежде всего разделение ресурсов машины в пространстве и во времени.

Программное обеспечение (англ. software) — это совокупность программ, обеспечивающих функционирование компьютеров и решение задач предметных областей.

Системное ПО – управляет работой компьютера и выполняет различные вспомогательные функции;

Прикладное ПО – предназначено для решения конкретных задач пользователя в предметных областях;

Инструментальное ПО – это средства создания программного обеспечения и информационных систем (системы программирования, инструментальные среды и системы моделирования).

Системное программное обеспечение предназначено для организации эффективной работы компьютера и пользователя, а также эффективного выполнения прикладных программ.

Состав системного ПО: Операционная система и Сервисные программы

Операционная система (ОС) — это комплекс программ, предназначенных для управления загрузкой, запуском и выполнением других пользовательских программ, а также для планирования и управления вычислительными ресурсами ЭВМ, т.е. управления ее работой с момента включения до момента выключения питания.

ОСНОВНЫЕ ФУНКЦИИ ОС

1. Поддержка диалога с пользователем
2. Ввод-вывод и управление данными
3. Планирование и организация процесса обработки программ
4. Распределение ресурсов (оперативной и кэш памяти, процессора, внешних устройств)
5. Запуск программ на выполнение
6. Выполнение вспомогательных операций обслуживания
7. Передача информации между различными внутренними устройствами
8. Поддержка работы периферийных устройств (внешние устройства)

ОСНОВНЫЕ КОМПОНЕНТЫ ОС:

1. Модули, обеспечивающие пользовательский интерфейс
2. Модуль, управляющий файловой системой
3. Модуль, расшифровывающий и выполняющий команды
4. Драйверы периферийных устройств

КЛАССИФИКАЦИЯ ОС ПО ПРИЗНАКАМ:

1. По числу параллельно решаемых на компьютере задач
2. По числу одновременно работающих пользователей
3. По типу интерфейса
4. По типу аппаратуры
5. По числу разрядов адресной шины компьютеров

Основная задача файловой системы — обеспечение взаимодействия программ и физических устройств ввода/вывода (различных накопителей). Она также определяет структуру хранения файлов и каталогов на диске, правила задания имен файлов, допустимые атрибуты файлов, права доступа и др.

Файл — это поименованная последовательность любых данных, стандартная структура которой обеспечивает ее размещение в памяти машины (компьютера).

АТРИБУТЫ ФАЙЛА:

1. **R (Read-Only) — «только для чтения»**
2. **H (Hidden) — «скрытый файл»**
3. **A (Archive) — «неархивированный файл»**

Надежность ПО – это вероятность безотказной работы ПО на основе заданных условий в течении определенного промежутка времени.

Основные задачи при проектировании ПО:

1. Выявление и исключение ошибок на этапе разработки ПО;

2. Проектирование отказоустойчивого ПО, которая выявляет ошибки программы за счет ошибок аппаратной части и восстанавливает нормальное состояние ПО.

Критерии оценки ПО:

1. **Функциональные в соответствии с поставленной задачей**
2. **Конструктивные полностью характеризующие ПО**

Основные виды контроля:

1. **Статический**
2. **Динамический**
3. **Визуальный**

Статический контроль – проверка программы на правдоподобия текста без применения инструментальных средств (целостность, живучесть, завершенность, работоспособность).

Визуальный контроль – это проверка программы без использования компьютера. Сквозной контроль программы – насколько корректно написана программа.

Динамический контроль – проверка правильности программы и ее выполнение на компьютере.

Язык С++

Язык программирования Си++ — компилируемый строго типизированный язык программирования общего назначения. Язык поддерживает объектно-ориентированное программирование.

Язык является регистрозависимым, то есть различает заглавные и строчные буквы.

Синтаксис языка C++

Комментарий до конца строки	//
Комментарии, которые не могут быть вложенными	/* ... */
Алфавит языка	a…z, A…Z, 0…9
Присваивание	имя = выражение
Объявление переменной	тип имя
Объявление переменной с присваиванием значения	тип имя = значение
Группировка выражений	(…)
Блок	{…}
Равенство (сравнение)	==
Неравенство	!=
Сравнения	< > <= >=
Определение функции	тип function имя (аргументы)
Вызов функции	function имя (аргументы)
Вызов функции без параметров	function имя ()

Синтаксис языка C++

Последовательность	;
Если-то	if условие истинно блок
Если-то-иначе	if условие истинно блок1 else блок2
Бесконечный цикл	while (true) тело цикла
Цикл с предусловием	while (условие истинно) тело цикла
Цикл с постусловием	do тело цикла while (условие ложно)
Цикл for-next для диапазона целых чисел с шагом +1	for (int i = first; i <= last ; i++)
Цикл for-next для диапазона целых чисел с шагом -1	for (int i = first; i <= last ; i--)

Структура программы для Microsoft Visual Studio

```
// struct_program.cpp: определяет точку входа для
консольного приложения.
#include "stdafx.h"
//здесь подключаем все необходимые препроцессорные
директивы
int main() { // начало главной функции с именем main
//здесь будет находится ваш программный код
}
```

Любая программа на C++ должна содержать функцию с именем **main**. С этой функции начинается выполнение программы. include — директива препроцессора, т.е. сообщение препроцессору. Строки, начинающиеся с символа # обрабатываются препроцессором до компиляции программы.

Операции в С++

*	умножение;
/	деление;
+	сложение;
—	вычитание;
%	остаток от целочисленного деления;
++	инкрементирование (увеличение на 1);
— —	декрементирование (уменьшение на 1);
—	изменение знака;
&&	логическое И;
||	логическое ИЛИ;
!	логическое НЕ.

Комбинированные операторы

Комбинированные (или составные) операторы, помимо выполнения своей арифметической роли, одновременно выполняют роль присваивания значения переменным.

объект *= выражение; // объект = объект * выражение

объект /= выражение; // объект = объект / выражение

объект += выражение; // объект = объект + выражение

объект -= выражение; // объект = объект — выражение

объект %= выражение; // объект = объект % выражение

Создание проекта

Откройте меню «Файл → Создать → Проект». Перейдите на вкладку «Общие» и выберите «Пустой проект». Придумайте проекту любое название, например, «program1» и нажмите «OK».

В окне обозревателя решений (обычно он находится в левом верхнем углу) щелкните правой кнопкой на папке «файлы исходного кода». В диалоговом окне выберите пункт меню «Добавить → Создать элемент». Введите название для нового файла — main.cpp и нажмите кнопку «Добавить».

Код первой программы

Наберите следующий код:

```
#include<iostream>
#include<cstdlib> // для system
Using namespace std;

intmain()
{
cout<<"Hello, world!"<<endl; system("pause"); // Только для тех, у кого MS Visual Studio
Return 0;
}
```

Простая программа на C++

Описание синтаксиса

1. Директива #include используется для подключения других файлов в код.
2. Строка #include <iostream>, будет заменена содержимым файла «iostream.h», который находится в стандартной библиотеке языка и отвечает за ввод и вывод данных на экран.
3. #include <cstdlib> подключает стандартную библиотеку языка C. Это подключение необходимо для работы функции system.
4. Содержимое третьей строки — using namespace std; указывает на то, что мы используем по умолчанию пространство имен с названием «std». Все то, что находится внутри фигурных скобок функции
5. int main() {} будет автоматически выполняться после запуска программы.
6. Строка cout << "Hello, world!" << endl; говорит программе выводить сообщение с текстом «Hello, world» на экран.
7. Оператор cout предназначен для вывода текста на экран командной строки. После него ставятся две угловые кавычки (<<). Далее идет текст, который должен выводиться. Он помещается в двойные кавычки.
8. Оператор endl переводит строку на уровень ниже.
9. Если в процессе выполнения произойдет какой-либо сбой, то будет сгенерирован код ошибки, отличный от нуля. Если же работа программы завершилась без сбоев, то код ошибки будет равен нулю.
10. Команда return 0 необходима для того, чтобы передать операционной системе сообщение об удачном завершении программы.
11. — В конце каждой команды ставится точка с запятой.

Компиляция и запуск

Теперь скомпилируйте и запустите программу: нужно нажать сочетание клавиш «Ctrl+F5».

Если программа собралась с первого раза, то хорошо. Если компилятор говорит о наличии ошибок, значит вы что-то сделали неправильно.

Прочитайте текст ошибки и попробуйте ее исправить своими силами.

Типы данных

В языке C++ все переменные имеют определенный тип данных. Например, переменная, имеющая целочисленный тип, не может содержать ничего кроме целых чисел, а переменная с плавающей точкой — только дробные числа. Тип данных присваивается переменной при ее объявлении или инициализации. Ниже приведены основные типы данных языка C++, которые нам понадобятся.

Основные типы данных в C++

1. **int — целочисленный тип данных.**
2. **float — тип данных с плавающей запятой.**
3. **double — тип данных с плавающей запятой двойной точности.**
4. **char — символьный тип данных.**
5. **bool — логический тип данных.**

Объявление переменной

Объявление переменной в C++ происходит таким образом: сначала указывается тип данных для этой переменной а затем название этой переменной.

Пример объявления переменных int a; // объявление переменной a целого типа. float b; // объявление переменной b типа данных с плавающей запятой. double c = 14.2; // инициализация переменной типа double. char d = 's'; // инициализация переменной типа char. bool k = true; // инициализация логической переменной k.

Заметьте, что в C++ оператор присваивания (=) — не является знаком равенства и не может использоваться для сравнения значений. Оператор равенства записывается как «двойное равно» ==.

Присваивание используется для сохранения определенного значение в переменной. Например, запись вида a = 10 задает переменной a значение числа 10.

Простой калькулятор на C++

Сейчас мы напишем простую программу-калькулятор, которая будет принимать от пользователя два целых числа, а затем определять их сумму:

```
#include<iostream>
Using namespace std;
Int main() {
setlocale(0, "");
int a, b; // объявление двух переменных a и b целого типа данных.
cout<<"Введите первое число: "; /*7*/
cin>> a; // пользователь присваивает переменной a какое-либо значение.
cout<<"Введите второе число: ";
cin>> b;  /*12*/
int c = a + b; // новой переменной c присваиваем значение суммы введенных пользователем данных.
cout<<"Сумма чисел = "<< c << endl; // вывод ответа.
Return 0;
}
```

Встречаются ситуации, когда программе нужно выбрать, какую операцию ей выполнить, в зависимости от определенного условия.

К примеру, мы вводим с клавиатуры целое число. Если это число больше десяти, то программа должна выполнить одно действие, иначе — другое. Реализуем этот алгоритм на C++ с помощью конструкции ветвления.

Оператор if

- Оператор if служит для того, чтобы выполнить какую-либо операцию в том случае, когда условие является верным. Условная конструкция в C++ всегда записывается в круглых скобках после оператора if.
- Внутри фигурных скобок указывается тело условия. Если условие выполнится, то начнется выполнение всех команд, которые находятся между фигурными скобками.
- Каждому оператору if соответствует только один оператор else. Совокупность этих операторов — else if означает, что если не выполнилось предыдущее условие, то проверить данное. Если ни одно из условий не верно, то выполняется тело оператора else.
- Если после оператора if, else или их связки else if должна выполняться только одна команда, то фигурные скобки можно не ставить.

Пример конструкции ветвления

```
#include<iostream>
Using namespace std;
Int main() {
setlocale(0, "");
double num;
cout<<"Введите произвольное число: "; cin>> num;
if (num <10) // Если введенное число меньше 10. cout<<"Это
число меньше 10."<< endl;
elseif (num == 10) cout<<"Это число равно 10."<< endl;
Else cout<<"Это число больше 10."<< endl;
Return 0;
}
```

```
#include<iostream>
Using namespace std;
Int main() {
setlocale(0, "");
double num;
 cout<<"Введите произвольное число: "; cin>> num;

if (num <10) { // Если введенное число меньше 10.
cout<<"Это число меньше 10."<< endl;
}
else{ // иначе
cout<<"Это число больше либо равно 10."<< endl;     }
Return 0;
 }
```

ЦИКЛ FOR

- Если мы знаем точное количество действий (итераций) цикла, то можем использовать цикл for. Синтаксис его выглядит примерно так: for (действие до начала цикла; условие продолжения цикла; действия в конце каждой итерации цикла) { инструкция цикла; инструкция цикла 2; инструкция цикла N; }
- Итерацией цикла называется один проход этого цикла
- Существует частный случай этой записи, который мы сегодня и разберем:
- for (счетчик = значение; счетчик < значение; шаг цикла) { тело цикла; }
- Счетчик цикла — это переменная, в которой хранится количество проходов данного цикла.

Описание синтаксиса

1. Сначала присваивается первоначальное значение счетчику, после чего ставится точка с запятой.
2. Затем задается конечное значение счетчика цикла. После того, как значение счетчика достигнет указанного предела, цикл завершится. Снова ставим точку с запятой.
3. Задаем шаг цикла. Шаг цикла — это значение, на которое будет увеличиваться или уменьшаться счетчик цикла при каждом проходе.

Пример кода

```
#include<iostream>
Using namespace std;
Int main() {
int i; // счетчик цикла
int sum = 0; // сумма чисел от 1 до 1000.
setlocale(0, "");
for (i = 1; i <= 1000; i++) // задаем начальное значение 1, конечное
1000 и задаем шаг цикла - 1.
{
  sum = sum + i;
 }
cout<<"Сумма чисел от 1 до 1000 = "<< sum << endl;
return 0;
 }
```

ЦИКЛ WHILE

Когда мы не знаем, сколько итераций должен произвести цикл, нам понадобится цикл while или do...while. Синтаксис цикла while в C++ выглядит следующим образом.

while (Условие) { Тело цикла; }

Данный цикл будет выполняться, пока условие, указанное в круглых скобках является истиной. Решим ту же задачу с помощью цикла while. Хотя здесь мы точно знаем, сколько итераций должен выполнить цикл, очень часто бывают ситуации, когда это значение неизвестно.

Ниже приведен исходный код программы, считающей сумму всех целых чисел от 1 до 1000.

Пример кода.

```
#include<iostream>
Using namespace std;
Int main() {
setlocale(0, "");
int i = 0; // инициализируем счетчик цикла.
int sum = 0; // инициализируем счетчик суммы.
while (i <1000)
    {
i++; sum += i;
}
cout<<"Сумма чисел от 1 до 1000 = "<< sum << endl;
Return 0;
}
```

ЦИКЛ DO WHILE

Цикл do while очень похож на цикл while.
Единственное их различие в том, что при выполнении цикла do while один проход цикла будет выполнен независимо от условия.
Решение задачи на поиск суммы чисел от 1 до 1000, с применением цикла do while.

Пример кода

```
#include<iostream>
Using namespace std;
Int main() {
setlocale(0, "");
int i = 0; // инициализируем счетчик цикла.
int sum = 0; // инициализируем счетчик суммы.
do {// выполняем цикл.
i++;
sum += i;
 }
while (i <1000); // пока выполняется условие.
cout<<"Сумма чисел от 1 до 1000 = "<< sum << endl;
Return 0;
 }
```

Массив — это область памяти, где могут последовательно храниться несколько значений.

Массив создается почти так же, как и обычная переменная. Для хранения десяти фамилий нам нужен массив, состоящий из 10 элементов. Количество элементов массива задается при его объявлении и заключается в квадратные скобки.

Чтобы описать элементы массива сразу при его создании, можно использовать фигурные скобки. В фигурных скобках значения элементов массива перечисляются через запятую. В конце закрывающей фигурной скобки ставится точка с запятой.

Пример массива

```
#include<iostream>
#include<string>
Int main() {
std::string students[10] = {
"Иванов", "Петров", "Сидоров",
"Ахмедов", "Ерошкин", "Выхин",
"Андеев", "Вин Дизель", "Картошкин", "Чубайс"    };
for (int i = 0; i <10; i++) {
std::cout<< students <<std::endl; // Пытаемся вывести весь
массив непосредственно
}
Return 0;
 }
```

Пример массива, заполняемого с клавиатуры

```
#include<iostream>
#include<string>
Using namespace std;
 intmain() {
int arr[10]; // Заполняем массив с клавиатуры
for (int i = 0; i <10; i++) {
cout<<"["<< i + 1<<"]"<<": ";
cin>> arr[i];
}
// И выводим заполненный массив.
cout<<"\nВашмассив: ";
for (int i = 0; i <10; ++i) { cout<< arr[i] <<" ";        }
cout<< endl;
Return 0;
}
```

Очень часто в программировании необходимо выполнять одни и те же действия. Например, мы хотим выводить пользователю сообщения об ошибке в разных местах программы, если он ввел неверное значение. без функций это выглядело бы так:

```
#include<iostream>
#include<string>
Using namespace std;
Int main() {
string valid_pass = "qwerty123";
string user_pass;
cout<<"Введите пароль: ";
getline(cin, user_pass);
if (user_pass == valid_pass) {
cout<<"Доступ разрешен."<<endl;     }
 else { cout<<"Неверный пароль!"<< endl;
}
return 0;
}
```

Аналогичный пример с функцией

```
#include<iostream>
#include<string>
Using namespace std;
voidcheck_pass(string password) {
string valid_pass = "qwerty123";
if (password == valid_pass) { cout<<"Доступ разрешен."<<endl;     }
else { cout<<"Неверный пароль!"<<endl;
}
}
Int main() {
string user_pass;
cout<<"Введите пароль: ";
getline (cin, user_pass);
check_pass (user_pass);
Return 0;
 }
```

- Любая функция имеет тип, также, как и любая переменная.
- Функция может возвращать значение, тип которого в большинстве случаев аналогично типу самой функции.
- Если функция не возвращает никакого значения, то она должна иметь тип void (такие функции иногда называют процедурами)
- При объявлении функции, после ее типа должно находиться имя функции и две круглые скобки — открывающая и закрывающая, внутри которых могут находиться один или несколько аргументов функции, которых также может не быть вообще.
- После списка аргументов функции ставится открывающая фигурная скобка, после которой находится само тело функции.
- В конце тела функции обязательно ставится закрывающая фигурная скобка.

Функции очень сильно облегчают работу программисту и намного повышают читаемость и понятность кода, в том числе и для самого разработчика

При выполнении любой программы, все необходимые для ее работы данные должны быть загружены в оперативную память компьютера.

Для обращения к переменным, находящимся в памяти, используются специальные адреса, которые записываются в шестнадцатеричном виде, например, 0x100 или 0x200.

Если переменных в памяти потребуется слишком большое количество, которое не сможет вместить в себя сама аппаратная часть, произойдет перегрузка системы или ее зависание.

Если мы объявляем переменные статично, так как мы делали в предыдущих уроках, они остаются в памяти до того момента, как программа завершит свою работу, а после чего уничтожаются.

Указатель — это переменная, хранящая в себе адрес ячейки оперативной памяти, например, 0x100.

Мы можем обращаться, например к массиву данных через указатель, который будет содержать адрес начала диапазона ячеек памяти, хранящих этот массив.

После того, как этот массив станет не нужен для выполнения остальной части программы, мы просто освободим память по адресу этого указателя, и она вновь станет доступно для других переменных.

- **Когда мы оперируем данными, то используем знак ***
- **Когда мы оперируем адресами, то используем знак &**

Пример использования указателей

```
#include<iostream>
usingnamespacestd;
intmain() {
int *a = newint; // Объявление указателя для переменной типа int
int *b = newint(5); // Инициализация указателя
*a = 10;
*b = *a + *b;
cout<<"b is "<< *b << endl;
delete b;
delete a;
Return 0;
}
```

Указатель “this ->”

This - это указатель. Он указывает на объект, для которого вызывается функция-член.

Статические функции-члены не имеют указатель this.

Когда мы разбирали понятие массива, топри объявлении мы задавали массиву определенный постоянный размер. Возможно, кто-то пробовал делать так:

```
int n = 10;
int arr[n];
```

Но, как уже было сказано — при объявлении статического массива, его размером должна являться числовая константа, а не переменная.

В большинстве случаев, целесообразно выделять определенное количество памяти для массива, значение которого изначально неизвестно.

Например, необходимо создать динамический массив из N элементов, где значение N задается пользователем. Мы уже учились выделять память для переменных, используя указатели. Выделение памяти для динамического массива имеет аналогичный принцип

Пример динамического массива

```
#include<iostream>
Using namespace std;
intmain() {
int num; // размер массива
cout<<"Enter integer value: ";
cin>> num; // получение от пользователя размера массива
int *p_darr = newint[num]; // Выделение памяти для массива
for (int i = 0; i < num; i++) {
// Заполнение массива и вывод значений его элементов
p_darr[i] = i;
cout<<"Value of "<< i <<" element is "<< p_darr[i] << endl;
}
delete [] p_darr; // очистка памяти
Return 0;
}
```

При запуске программы из командной строки, ей можно передавать дополнительные параметры в текстовом виде. Например, следующая команда
“ping -t 5 google.com”
Будет отправлять пакеты на адрес google.com с интервалом в 5 секунд. Здесь мы передали программе ping три параметра: «-t», «5» и «google.com», которые программа интерпретирует как задержку между запросами и адрес хоста для обмена пакетами.

В программе эти параметры из командной строки можно получить через аргументы функции mainпри использовании функции main в следующей форме:

Int main(int argc, char* argv[]){ /* ... */ }

Первый аргумент содержит количество параметров командной строки. Второй аргумент — это массив строк, содержащий параметры командной строки. Т.е. первый аргумент указывает количество элементов массива во втором аргументе.

Первый элемент массива строк (argv[0]) всегда содержит строку, использованную для запуска программы (либо пустую строку).

Следующие элементы (от 1 до argc - 1) содержат параметры командной строки, если они есть. Элемент массива строк argv[argc] всегда должен содержать 0.

Пример программы

```
#include<iostream>
Using namespace std;
Int main(int argc, char *argv[]) {
for (int i = 0; i < argc; i++) {
// Выводим список аргументов в цикле
cout<<"Argument "<< i <<" : "<< argv[i] << endl;
}
Return 0;
}
```

Литература

1. Бройдо В.Л. Вычислительные системы, сети и телекоммуникации: Учеб. пособие для вузов.-2-е изд. - СПб.:Питер, 2015. - 702 с.
2. Максимов Н.В. и др. Архитектура ЭВМ и вычислительных систем: Учебник/Н.В.Максимов, Т.Л.Партыка, И.И.Попов. - М.: ФОРУМ: ИНФРА-М, 2016. - 511 с.
3. Романов Е.Л. Си++. От дилетанта до профессионала. М., 2015. - 600 с.
4. Учебник по C++ для начинающих [Электронный ресурс] http://www.programmersclub.ru/main
5. Литвиненко Н. А. Технология программирования на C++ [Электронный ресурс] http://www.proklondike.com/books/cpp/technology_of_programming_on_cplus.html
6. Стефан Р. Дэвис. C++ Для чайников [Электронный ресурс] http://www.proklondike.com/books/cpp/cplus_dlja_chainikov.html
7. Магда Ю.С. Программирование и отладка C/C++ приложений для микроконтроллеров ARM. - М.: ДМК Пресс, 2015. - 168 с.

ГОСТы

1. ГОСТ 19.701-90
2. http://cert.obninsk.ru/gost/282/282.html
3. https://prog-cpp.ru/block-schema/
4 .https://sites.google.com/site/anisimovkhv/learning/pris/lecture/tema8/tema8_2

Бьярне Страуструп. Программирование. Принципы и практика с использованием C++. – М.: Вильямс, 2016. – 1328 с.
Стивен Язык программирования C++. Лекции и упражнения. – М.: Вильямс, 2017. – 1248 с.
Эндрю Кениг, Барбара Э. Му. Эффективное программирование на C++. Практическое программирование на примерах. - М.: Вильямс, 2016. – 368 с.
Алексей Васильев. Программирование на C++ в примерах и задачах. – М.: Эксмо, 2016. – 368 с.
Учебник по C++ для начинающих [Электронный ресурс] http://www.programmersclub.ru/main
Литвиненко Н. А. Технология программирования на C++ [Электронный ресурс]

КНИГИ

1. Бьярне Страуструп. Программирование. Принципы и практика с использованием C++. – М.: Вильямс, 2016. – 1328 с.
2. Стивен Язык программирования C++. Лекции и упражнения. – М.: Вильямс, 2017. – 1248 с.
3. Эндрю Кениг, Барбара Э. Му. Эффективное программирование на C++. Практическое программирование на примерах. - М.: Вильямс, 2016. – 368 с.
4. Алексей Васильев. Программирование на C++ в примерах и задачах. – М.: Эксмо, 2016. – 368 с.

Учебник по C++ для начинающих [Электронный ресурс] http://www.programmersclub.ru/main

Литвиненко Н. А. Технология программирования на C++ [Электронный ресурс] http://www.proklondike.com/books/cpp/technology_of_programming_on_cplus.html

Стефан Р. Дэвис. C++ для чайников [Электронный ресурс] http://www.proklondike.com/books/cpp/cplus_dlja_chainikov.html

ЭЛЕКТРОННЫЕ РЕСУРСЫ

1. Учебник по C++ для начинающих [Электронный ресурс] http://www.programmersclub.ru/main
2. Литвиненко Н. А. Технология программирования на C++ [Электронный ресурс] http://www.proklondike.com/books/cpp/technology_of_programming_on_cplus.html
3. Стефан Р. Дэвис. C++ для чайников [Электронный ресурс] http://www.proklondike.com/books/cpp/cplus_dlja_chainikov.html