ОСНОВЫ ПРОГРАММИРОВАНИЯ на C++

лектор доцент кафедры ПОКС
Макиева Замира Джумакматовна

Введение

Устройство ЭВМ основано на принципах двоичной арифметики, где для представления чисел используются всего две цифры 0 и 1. Процесс программирования в кодах малоэффективен.

Оптимизация программирования в двоичных кодах заключалась в разработке специальной системы кодирования двоичных машинных команд словесными сокращениями. Такое программирование удобно для программиста, но текст такой программы становится абсолютно непонятным компьютеру и требует специальной программы-переводчика (или компилятора), которая бы заменяла текст программы исходной двоичной командой. С момента реализации этой идеи кодирование становится программированием.

Языки, которые требуют предварительного перевода, называются языками высокого уровня. Эти языки более близки к естественному языку. Использование языков высокого уровня значительно повышает эффективность программирования по сравнению с обычным кодированием.

Решение задачи на ЭВМ состоит из следующих этапов:

1) постановка задачи;
2) выбор численного метода решения;
3) разработка алгоритма;
4) программирование алгоритма;
5) тестирование и отладка программы;
6) решение задачи на ЭВМ.

Постановка задачи

- определяет цель решения задачи, раскрывает её содержание. Задача формируется на уровне профессиональных понятий, должна быть корректной и понятной исполнителю. Ошибка в постановке задачи обнаруживается на последующих этапах и приводит к тому, что работа по подготовке задачи к решению должна будет начаться с самого начала. Для большинства задач на этом этапа разрабатывается математическая модель задачи.

Математическая модель – специальная форма описания задачи использующая язык математики. Математическая модель задаётся в виде уравнений или формул, необходимых для решения задачи. Кроме математической модели, на этапе постановки определяется перечень исходных данных, перечень результатов, начальные условия, точность вычисления.

Выбор численного метода:

Одна и та же задача может быть решена с помощью различных численных методов. Выбор метода должен определяться такими факторами, как точность результатов решения, время решения. Для простых задач, данный этап может отсутствовать.

Например: Если в задаче требуется вычислить интеграл, может быть выбран для численного интегрирования метод прямоугольников, метод Симпсона или метод трапеции.

Разработка алгоритма

- **Алгоритм** – конечная последовательность предписаний (правил), однозначно определяющая процесс преобразования исходных и промежуточных данных в результат решения задачи. При разработке алгоритма математическая модель и выбранный численный метод являются основой для определения последовательности действий.
- Алгоритм должен обладать следующими свойствами:

а) массовость – алгоритм должен решать не одну задачу, а целый класс задач;

б) детерминантность – однозначность выполняемых действий, т. е. промежуточные и окончательные результаты разных пользователей должны быть одинаковыми при одинаковых исходных данных;

в) результативность - алгоритм должен обеспечивать получение результата после конечного числа шагов.

Алгоритмы классифицируются на:

- Алгоритмы линейной структуры (последовательные алгоритмы), в которых все действия выполняются последовательно друг за другом.
- Алгоритмы разветвляющейся структуры, в которых в зависимости от выполнения логического условия, процесс пойдет по одной из 2-х ветвей.
- Алгоритмы циклической структуры, содержащие многократно выполняемые участки вычислительного процесса, называемые циклом. Их использование позволяет существенно сократить схему алгоритма.

Алгоритмы со структурой вложенных циклов, содержащие цикл, внутри которого размещены один или несколько других циклов.

Алгоритм смешанной структуры, содержащий линейные, разветвляющиеся и циклические структуры.

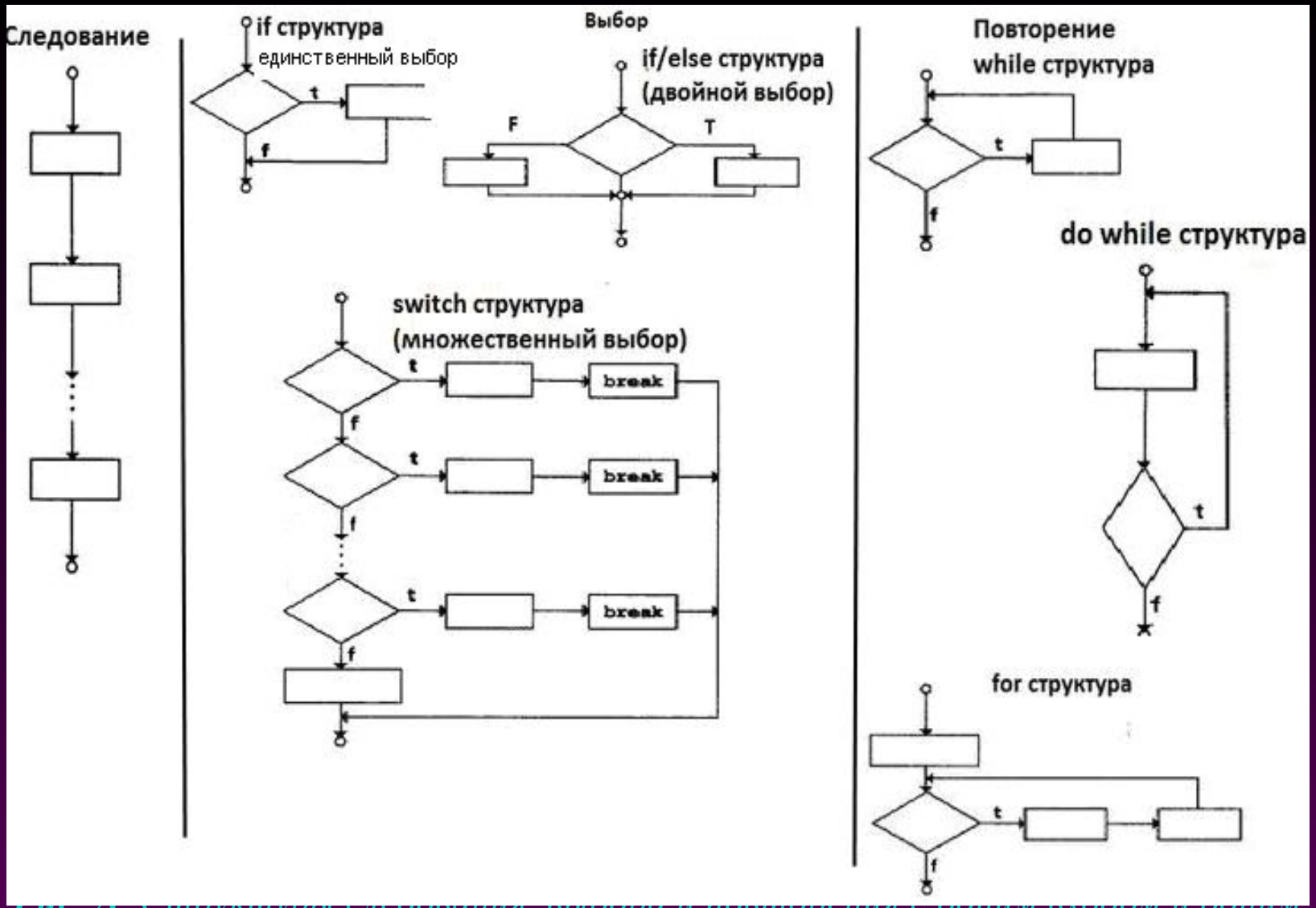

Рис. 1. Структуры следования, выбора и повторения с одним входом и одним выходом

Малые окружности использованы, чтобы отметить точки единственного входа и единственного выхода каждой структуры. Произвольное соединение отдельных символов блок-схем может привести к неструктурированным программам.

Для упрощения используются только структуры, имеющие только одну точку входа и одну точку выхода. Это позволяет формировать структурированные программы последовательным соединением управляющих структур, т.е. управляющие структуры просто размещается в программе одна за другой. Такой способ соединения называется пакетированием управляющих структур.

- При создании программ следует применять принципы **структурного программирования.**
- Структурное программирование позволяет создавать программы более простые для понимания, для проверки, отладки и модификации, чем неструктурированные. При изучении данной дисциплины мы будем учиться создавать структурированные программы.

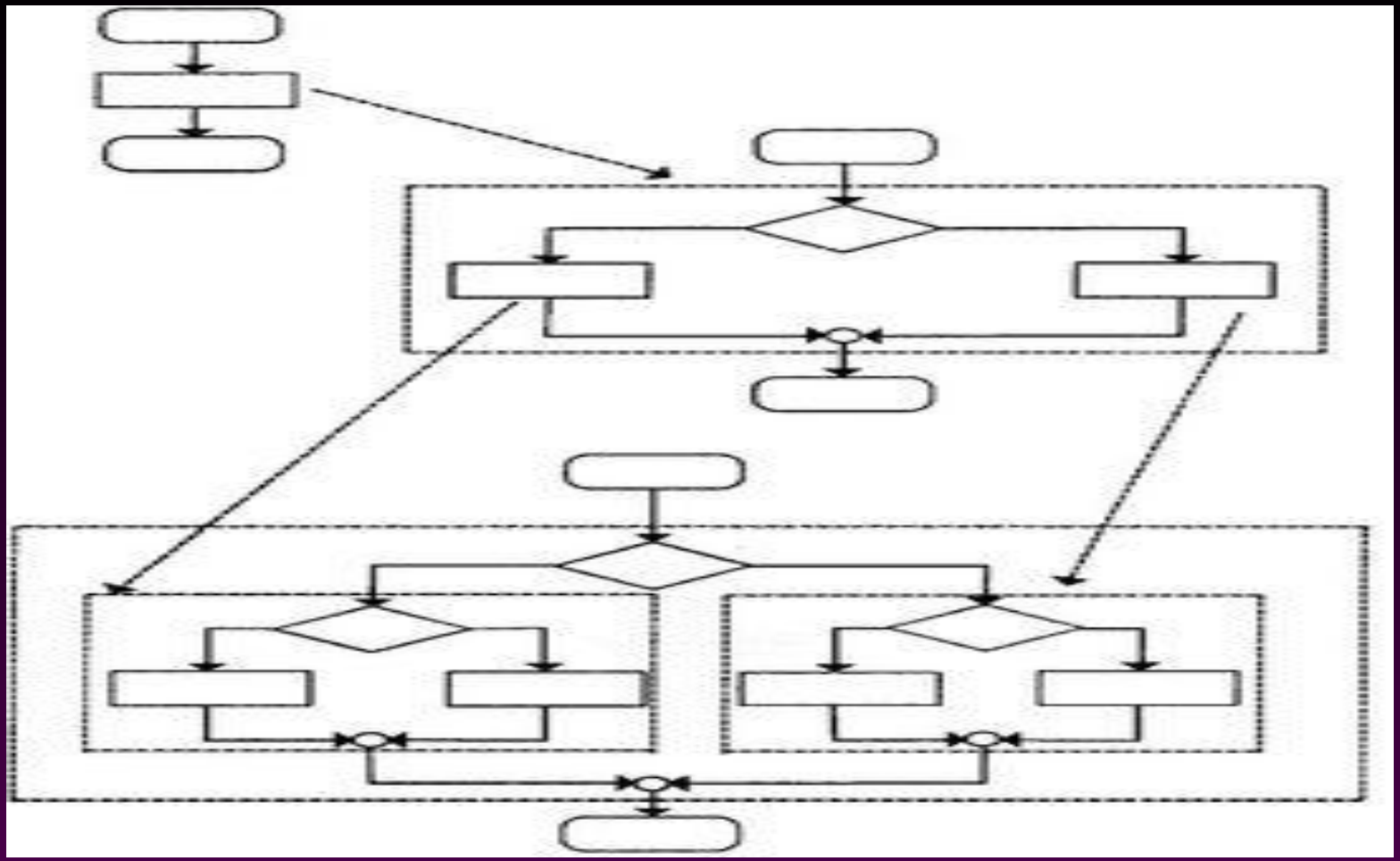

Рис. Б. *Правила формирования структурированных программ.* Символ прямоугольника на блок-схеме может использоваться для указания любых действий.

История создания C++

Язык C создан в начале 1970-х Кеном Томпсоном и Дэннисом Ритчи из Bell Labs для разработки операционной системы UNIX. Сначала они создали часть компилятора C, затем использовали ее для компиляции остальной части компилятора C и, наконец, применили полученный в результате компилятор для компиляции UNIX. Операционная система UNIX первоначально распространялась в исходных кодах на C среди университетов и лабораторий, а получатель мог откомпилировать исходный код на C в машинный код с помощью подходящего компилятора C. Распространение исходного кода сделало операционную систему UNIX уникальной; программист мог изменить операционную систему, а исходный код мог быть перенесен с одной аппаратной платформы на другую.

- C был третьим языком, который разработали Томсон и Ритчи в процессе создания UNIX; первыми двумя были, разумеется, A и B.
- В отличие от других языков высокого уровня, C мог работать с адресами памяти напрямую с помощью указателей и ссылок. Поскольку C сохранил способность прямого доступа к аппаратному обеспечению, его часто относят к языкам среднего уровня или в шутку называют "мобильным языком ассемблера".
- Что касается грамматики и синтаксиса, то C является структурным языком программирования.
- C++ является расширением языка C, поэтому почти все программы на C являются также C++ программами.
- C++ в отличие от C поддерживает объектно-ориентированное программирование.

Общая форма программы на C++

```
// комментарии
#include <библиотечные файлы>
using namespace std;
int main()
{   объявления констант и переменных;
    операторы;
    system("pause");
    return 0;
}
```

Ключевые слова в C++

- Ключевые слова выделяются голубым в Visual C++.
- Каждое ключевое слово имеет определенную роль и их нельзя использовать в именах переменных и констант.
- Ключевые слова:

```
bool, break, case, char, const, continue,
do, default, double, else, extern, false,
float, for, if, int, long, namespace,
return, short, static, struct, switch,
typedef, true, unsigned, void, while
```

Пример 1

```
// моя первая программа на C++
#include <iostream>
using namespace std;
int main()
{ setlocale (LC_ALL, "");
  int a, b, c;
  cout << " Введите число a:\n";
  cin >> a;
  cout << " Введите число b:\n";
  cin >> b;
  c = a * b;
  cout << " Если умножим " << a << " на " << b;
  cout << " получим " << c << endl;
  system("pause");
  return 0; }
```

a: 2 b: 5 c: 10

Идентификаторы в C++

Идентификаторы выделяются черным в **Visual C++.**

- Идентификатор – имя переменной, константы, функции и др.
- Идентификатор обязательно начинается с латинской буквы и далее следует любая последовательность букв, цифр и символов нижнего подчеркивания.
- Примеры правильных идентификаторов: First_name, age, y2000, y2k
- Примеры неправильных идентификаторов : 2000y
- Идентификаторы не должны содержать спецсимволы. Например: X=Y, J-20, ~Ricky,*Michael не могут быть идентификаторами.
- C++ чувствителен к регистрам: Hello, hello, HELLO это различные идентификаторы.

Комментарии в C++

- Комментарии выделяются зеленым в Visual C++.
- Комментарии нужны для пояснений в программе и игнорируются компилятором.
- В C++ существует два вида:

```
// однострочный комментарий

/*
многострочный комментарий
заключаем между знаками
 */
```

Директивы препроцессору

- Директива `#include` указывает компилятору включить некоторый существующий код в вашу программу. Этот код подключается в программу при компоновке.

- Две формы использования директив `#include`:

```
#include <iostream> //для библиотечных файлов
#include "my_lib.h"  //для файлов, созданных
          //программистом
```