Лекция 1

Парадигма программирования

Парадигма программирования – это способ классификации языков программирования на основе их особенностей.

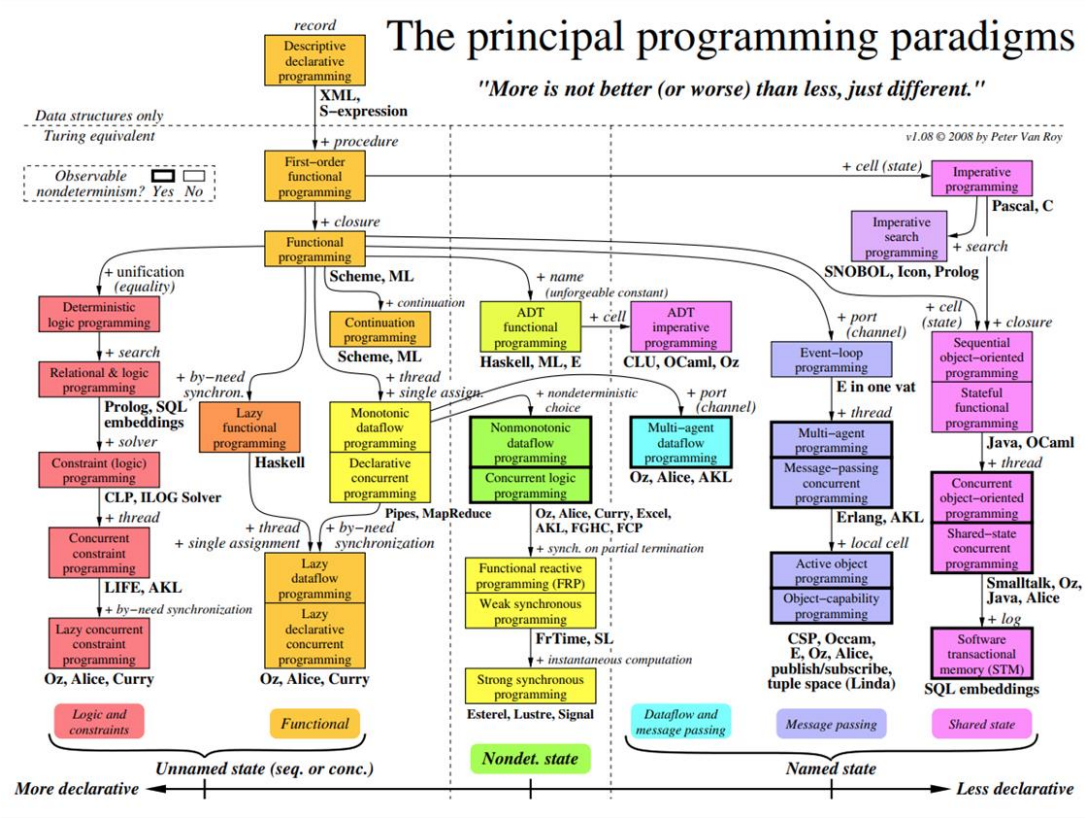


Рис. 1.1. Парадигмы программирования

Императивная парадигма - задает последовательность команд, как надо что-то сделать (программа - набор директив, обращенных к ПК) (Fortran, Pascal, Basic, C/C++)

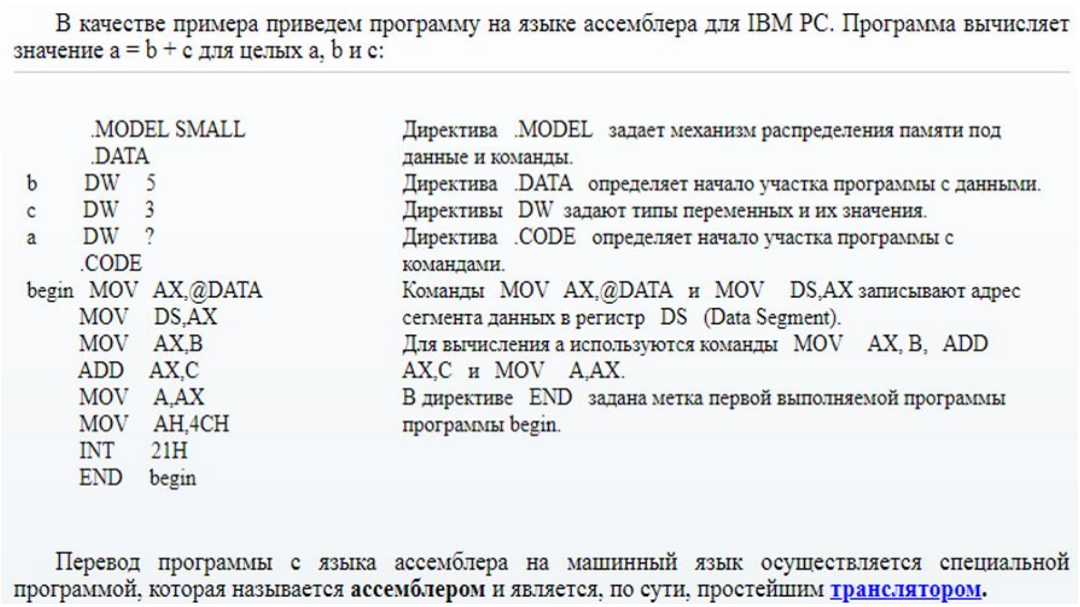
Декларативная - только то, что надо сделать (программа - не набор команд, а описание действий, которые необходимо осуществить (SML, Haskell, Prolog).

“Безпарадигменнное” программирование

«**Стихийное**» - отсутствие технологий, программирование – искусство.

Программы в машинных кодах. Сложность ограничивалась способностью программиста одновременно мысленно отслеживать последовательность выполняемых операций и местонахождение данных при программировании.

Ассемблеры - вместо двоичных и 16-ричных кодов используются символические имена данных и мнемоники кодов операций.



В качестве примера приведем программу на языке ассемблера для IBM PC. Программа вычисляет значение a = b + c для целых a, b и c:

```
        .MODEL SMALL
        .DATA
b       DW   5
c       DW   3
a       DW   ?
        .CODE
begin   MOV  AX,@DATA
        MOV  DS,AX
        MOV  AX,B
        ADD  AX,C
        MOV  A,AX
        MOV  AH,4CH
        INT  21H
        END  begin
```

Директива .MODEL задает механизм распределения памяти под данные и команды.
Директива .DATA определяет начало участка программы с данными.
Директивы DW задают типы переменных и их значения.
Директива .CODE определяет начало участка программы с командами.
Команды MOV AX,@DATA и MOV DS,AX записывают адрес сегмента данных в регистр DS (Data Segment).
Для вычисления a используются команды MOV AX, B, ADD AX,C и MOV A,AX.
В директиве END задана метка первой выполняемой программы программы begin.

Перевод программы с языка ассемблера на машинный язык осуществляется специальной программой, которая называется **ассемблером** и является, по сути, простейшим **транслятором**.

Рис. 1.2. Программа на языке ассебмлера

Структурное программирование – парадигма программирования, в основе которой лежит представление программы в виде иерархической структуры блоков. В 1960-1970-х г.г. математически обосновано возможность структурной организации программ (теоремы Бёма-Якопини) и Э.Дейкстра «О вреде оператора goto».

Любую программу можно построить из трёх базовых управляющих конструкций: последовательность, ветвление и цикл; также используются подпрограммы. Это программы без использования оператора goto.

Разработка программы ведётся пошагово, методом «сверху вниз».

В основе - декомпозиция сложных систем с целью последующей реализации в виде отдельных небольших п/п.

Объектно-ориентированное программирование - парадигма разработки, подразумевающая организацию программного кода, ориентируясь на данные и объекты, а не на функции и логические структуры.

- Программа – совокупность взаимодействующих объектов.
- Каждый О - экземпляр класса.
- Классы образуют иерархию с наследованием свойств.
- Взаимодействие О - путем передачи сообщений.

Функциональное программирование – парадигма программирования, процесс вычисления трактуется как вычисление значений функций (в математическом понимании) в отличие от функций как п/п в процедурном программировании.

Программа может интерпретироваться как функция с одним или несколькими аргументами. Прозрачное моделирование текста программ математическими средствами (SML).

Логическое программирование – парадигма программирования, основанная на математической логике.

Программа - совокупность логических утверждений (высказываний) и правил вывода. Естественно формализуется логика поведения. Применимы для ЭС, для описаний правил принятия решений (Prolog).

Инструменты разработки

Интегрированная среда разработки (англ. Integrated development environment, IDE) – это программное приложение, включающее в себя набор инструментов для разработки ПО

Как правило, IDE включает в себя: редактор кода, компилятор, отладчик кода, конструктор графического пользовательского интерфейса (GUI).

Интерактивная среда Adobe - это онлайн-инструмент, позволяющий писать, хранить и запускать код на C++ без установки чего-либо. Это упрощенная среда разработки, доступная удаленно через Интернет.

Популярные IDE

Microsoft © Visual Studio Экспресс ®. Одноплатформенная среда разработки, разработанная специально для создания программ на C/C++, как в ОС MS Windows, так и для нее.

https://www.visualstudio.com

Eclipse. Многоплатформенная среда разработки, разработанная специально для Java. Программирование на C++ возможно без дополнительной настройки (специальная версия C/C++ доступна для загрузки).

https://eclipse.org

NetBeans. Многоплатформенная среда разработки, разработанная специально для Java. Программирование на C++ возможно без дополнительной настройки (специальная версия C/C++ доступна для загрузки).

https://netbeans.org

Code::Blocks. Многоплатформенная среда разработки для программирования на C/C++. Установщик Windows по умолчанию не включает компилятор C++

http://www.codeblocks.org

XCode. Одноплатформенная среда разработки, разработанная специально для создания приложений для ОС, разработанных Apple Inc. Программирование на C++ полностью доступно.

https://developer.apple.com/xcode

Система контроля версий

Система контроля версий – специализированное программное обеспечение для работы с изменяющейся информацией. Система контроля версий позволяет хранить несколько версий одного и того же документа, и возвращаться к более ранним версиям, определять, кем и когда было выполнено то или иное изменение, если это необходимо, а также многое другое.

Примером распределенной системы контроля версий является Git, для использования которого существуют различные сервисы, например, GitHub/

Язык программирования C++

Ветвь языков: **B → C → C++ → C#**

B: 1963 год, К.Томпсон Основная цель разработки - реализация ОС UNIX.

C: 1972 г. (на основе языка B), К.Томпсон и Д.Ритчи. Язык C - язык B расширился: явное использование типов, структуры и др.

C++: 1984 году, Б.Страуструп, ООП расширение C, вводится понятие класса как объекта данных.

C#: (2000 г.) Microsoft выпустила C++ нового поколения - C#, его постулат: "всякая сущность есть объект".

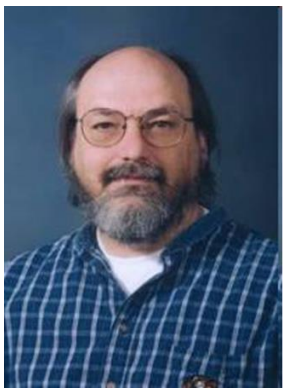


Рис. 1.3. Кен Томпсон, Деннис Ритчи, Бьёрн Страуструп

Известны благодаря двум важнейшим программным разработкам XX века: ОС UNIX и ЯП C.

"Язык программирования C++". Это первая книга, описывающая ЯП C++, написанная создателем языка Бьярне Страуструпом.

"A Tour of C++" "Экскурсия по C++" того же автора. Это скорее справочник, чем учебник, не используйте его для изучения C++ - используйте его для улучшения памяти, когда вы начнете серьезно использовать C++.

Алекс Аллейн "Переход на C++". Это отличная книга для начинающих, представляющая пошаговое руководство по становлению программистом на C++.

Скотт Мейерс "Эффективный C++" - кто стремится улучшить свои программы и узнать больше о лучших практиках в C++ - книга

Структура программы

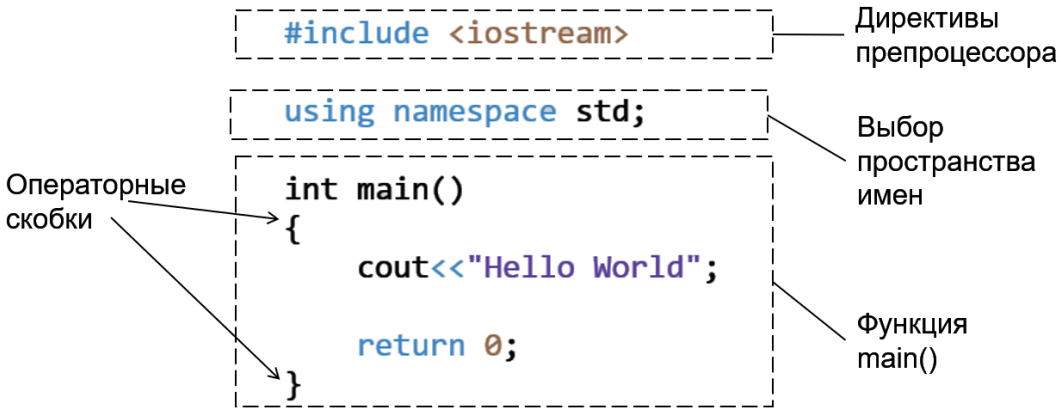


Рис. 1.4. Структура программы

Символ # (hash, хэш) означает, что содержимое этой строки является директивой препроцессора.

Это отдельная часть компилятора, задача которой - предварительно прочитать текст программы и внести в него некоторые изменения. Префикс "pre" предполагает, что эти операции выполняются до того, как произойдет полная обработка (компиляция).

Изменения, которые внесет препроцессор, полностью контролируются его директивами. В программе - директива include. Когда препроцессор встречает (выполняет) эту директиву, он заменяет директиву содержимым файла, имя которого указано в директиве (файл с именем iostream).

Изменения, внесенные препроцессором, не изменяют содержимое исходного файла. Любые изменения вносятся в изменяемую копию программы, которая исчезает сразу же после завершения компилятором своей работы.

Зачем препроцессору включать содержимое файла с названием iostream? Написание программы похоже на построение конструкции из готовых блоков. В программе собираемся использовать блок - cout, но компилятор пока не знает, что cout является допустимым именем для этого блока.

Файлы заголовков (header files) содержат набор предварительной информации, необходимой компилятору (информации о блоках, которые могут быть использованы программой для написания текста на экране или для чтения символов с клавиатуры).

Поэтому, когда программа собирается что-то написать (вывести на экран), она будет использовать блок под названием cout, который способен это выполнить. Нужно предупредить компилятор об этом. Разработчики компилятора поместили набор этой предварительной информации в файл iostream. Все, что нужно сделать, это использовать этот файл.

Препроцессор знает, где находится файл iostream.

Все элементы стандартной инфраструктуры C++ объявляются в namespace пространстве имен - std. **Пространство имен** - это абстрактный контейнер или среда, созданная для хранения логической группировки уникальных сущностей (блоков).

Сущность, определенная в **namespace** пространстве имен, связана только с этим пространством имен. Если вы хотите использовать многие стандартные сущности C++, вы должны вставить инструкцию using namespace в начало каждого файла, вне какой-либо функции.

В инструкции должно быть указано имя желаемого пространства имен namespace (в нашем случае **std**). Это сделает стандартные удобства доступными на протяжении всей программы.

Можно опустить использование using namespace пространства имен в коде, но тогда нужно сообщить компилятору, откуда взялся блок cout

Это нужно делать это каждый раз, когда используете любую из сущностей, производных от пространства имен std.

Без **using namespace std** нужно писать:

`std::cout` вместо: `cout`

Компилятору все равно, что вы выберете. Предпочтительно использовать пространство имен using namespace, это делает код более понятным и последовательным.

Имя главной функции программы: main. С нее начинается выполнение программы

Каждая функция в C++ начинается со следующего набора информации:

- каков результат выполнения функции?
- как называется функция?
- сколько аргументов принимает функция и каковы их имена?

В программе:

- результатом функции является целочисленное значение (int)
- название функции - main
- функция не требует никаких аргументов (между скобками ничего нет).

Этот набор информации называют prototype прототипом. Прототип ничего не говорит о том, для чего предназначена функция (это написано внутри функции). Внутренняя часть функции называется телом функции. Тело функции начинается с первой открывающей скобки { и заканчивается соответствующей закрывающей скобкой }. Тело функции может быть пустым, что означает, что функция ровно ничего не делает.

Можно создать lazy (ленивую) функцию:

```
void lazy() { }
```

Слово void перед именем функции (main), указывает компилятору, что функция не дает результата..

В более старых версиях C++ для объявления факта, что определенная функция (включая main) не принимает никаких аргументов:

```
int main(void) { }
```

<< указывает направление, в котором отправляется текст

Ввод и вывод

Подключение заголовочного файла iostream:

#include <iostream>

Ввод с клавиатуры: `cin >> x;`

Вывод на экран: `cout << x;`

Строки в программе на C++ заключены в кавычки.

Данная форма кода является наиболее естественной и наиболее легко читаемой людьми, но можно написать ее по-другому:

```
cout

<<

"It's me, your first program."

;
```

В языке C++ нет необходимости писать только один оператор в строке. Вы можете разместить два (или более) утверждения в одной строке или разделить одно утверждение на несколько строк, но читаемость (для людей) является очень важным фактором. Но компиляторы, в отличие от людей, никогда не будут жаловаться на ваш стиль.

Последняя строчка: return 0;

Это оператор языка C++. Его имя - return - возвращение. Используемый в функции, он вызывает завершение выполнения функции.

Если выполняется возврат внутри функции, эта функция немедленно прерывает ее выполнение.

Ноль 0 после слова return является результатом функции main.

«0» означает, что все в порядке.

Если написать: return 1; это означало бы, что что-то пошло не так, это не позволило успешно выполнить программу.

Обычно, если функция возвращает целочисленное значение (main), она обязана содержать оператор return , он обязан возвращать целочисленное значение. Иначе - ошибка компиляции.

Но основная функция main является исключительной. Объявлена как int, но не должна возвращать его. Более того, если внутри основной функции нет оператора return, компилятор предполагает, что неявно использовалось значение return 0.

Но рекомендуется использовать return в main.

```
Директивы препроцессора
 (в простейшем случае
#include <stdio.h>/*ввод/вывод*/
#include <math.h> /*стандартные
математические функции*/)
void main()
{ описания
   операторы
}
```

```
#include <stdio.h>
void main()
{int a,b,c;/*описание трех целых
переменных*/
 printf("введите a и b\n");
/*приглашение к вводу a и b*/
scanf("%d%d", &a, &b);/*ввод
a,b*/
c=a+b;/*вычисление c - суммы */
printf("c=%d\n", c);/*вывод c*/
}
```

Рис. 1.5. Пример программы на языке Си

Вывод с использованием printf

int *printf*(<форматная строка>, <список аргументов>)

```
float item = 10.12304;
printf("***%f***\n", item);
printf("***%10f***\n", item);
```

```
printf("***%012f***\n", item); // ведущие нули
printf("***%10.3f***", item);
```

Вывод:

```
***10.123040***
***  10.123040***
***00010.123040***
***   10.123***
```

Вывод таблицы квадратов и кубов:

```
int i;
/* вывод таблицы квадратов и кубов */
for(i=1; i<20; i++)
    // вывод с фиксированной шириной
    printf("%8d %8d %8d\n", i, i*i, i*i*i);
```

Использование <iomanip>

Таблица 3.1. Использование библиотеки iomanip

Файловый манипулятор	**Смысл**
fixed	**Указывает, что числа далее нужно выводить на экран с фиксированной точностью**
setprecision	**Задает количество знаков после запятой**
setw	**Указывает ширину поля, которое резервируется для вывода переменной**

Пример:

```cpp
#include <iostream>
#include <iomanip>

using namespace std;
const int N = 4;
int main()
{
    float mas[N] = {3.2569, -4.1245, 8.1144, -9.3241};
    cout << "***" << setw(4) << N << "***" << endl;
    cout << fixed << setprecision(2);
    for (int i = 0; i < N; i++) {
        cout << "***" << setw(8) << mas[i] << "***";
    }
    return 0;
}
```

```
***   4***
***    3.26******   -4.12******    8.11******   -9.32***
```

Рис. 3.1. Пример использования библиотеки iomanip

Числа

Числа, обрабатываемые компьютерами, бывают двух типов:

- целые числа int;
- числа с плавающей запятой float, содержат / могут содержать дробную часть.

Они различаются тем, как хранятся в памяти ПК, и диапазоном допустимых значений.

Тип – это характеристика числа, определяющая его вид, диапазон и применение. Как язык C++ распознает целые числа?

Использование одинарных кавычек в качестве разделителей, улучшающих читаемость целых чисел, было введено в C++17

Число одиннадцать миллионов сто одиннадцать тысяч сто одиннадцать.

В C++ это: 11111111 или можно записать: 11'111'11

Отрицательные числа в C++: -1234

Положительным числам не обязательно знак плюс: +123 123

Три дополнительных соглашения:

1 - позволяет использовать числа в (octal representation) восьмеричном представлении. Если целому числу предшествует цифра 0, оно будет рассматриваться как восьмеричное значение (что число должно содержать цифры, взятые только из диапазона от 0 до 7).

0123 восьмеричное число с (десятичным) значением = 83.

2 - шестнадцатеричные числа, предшествует префикс 0x или 0X.

0x123 - шестнадцатиричное число = десятичному значению 291.

3 - двоичные числа, которым предшествует префикс 0b или 0B.

0B1111 - двоичное число = десятичному значению 15.

Комментарии

Два способа вставки комментариев:

`// comment` - однострочные комментарии

и

`/* comment */` - блок комментариев (или комментарии в стиле C).

Вопрос: разрешено ли вам размещать один комментарий блока (например, /* int j; */)) внутри другого комментария блока?

```
/* int i; /* int j; */ int k; */
```

Ответ - нет. Такая нотация может быть не принята конкретным компилятором и, по сути, делает наш код подверженным капризам компилятора.

Числа с плавающей точкой

Десятичная точка необходима для распознавания чисел с плавающей запятой в C++: 4 и 4.0

Компилятор C++ видит их совершенно по-другому: 4 - это int 4.0 - это float.

```
300000000  3 • 10^8  3E8  6.62607E-34
```

Код:

```
int i = 10 / 4;  i =2
```

```
float x = 10.0 / 4.0;                x = 2,5
```

Пример. Вычислить значение q.

```
int i = 100;
```

```
int j = 25;
```

```
int k = 13;
```

```
int q = (5 * ((j % k) + i) / (2 * k)) / 2;
```

Ответ: q=10.

Базовые типы данных

Таблица 1.1. Базовые типы данных

Тип	Кол-во байт	Диапазон представимых значений	Описание
bool	1	true/false	логическая переменная (истина/ложь)
char	1	0 ... 255	символ или число
int	4	-2 147 483 648 ... 2 147 483 647	целое число со знаком
unsigned int	4	0 ... 4 294 967 295	целое число без знака
float	4	-2 147 483 648.0 ... 2 147 483 647.0	Число с плавающей точкой одинарной точности
double	8	-9 223 372 036 854 775 808.0 … 9 223 372 036 854 775 807	Число с плавающей точкой двойной точности

Операторы Pre и Post

```
int i = 1;

int j = i++;
```

В результате j получит значение = 1, а i – значение = 2.

```
int i = 1;

int j = ++i;
```

В результате i, и j будут = 2.

```
int i = 4;

int j = 2 * i++;

i = 2 * --j;
```

Результат = 14.

Операторы префиксной версии - справа налево, постфиксные операторы - слева направо.

Оператор быстрого доступа

```
x =  x+3;          x += 3;

i = i * 2;  i *= 2;
```

Общее описание: если op является оператором с двумя аргументами и используется в контексте:

```
variable = variable op expression;
```

то это выражение можно упростить:

```
variable op = expression;
```

Таблица 1.2. Оператор быстрого доступа

`i = i + 2 * j;`	`i += 2 * j;`
`Var = Var / 2;`	`Var /= 2;`
`Rem = Rem % 10;`	`Rem %= 10;`
`j = j - (i + Var + Rem);`	`j -= (i + Var + Rem);`

Ряд слов в языке Си++ не может использоваться в качестве идентификаторов – это ключевые слова.

Примеры:

asm auto bad_cast bad_typeid bool break

Пример:

```
int max (int x, int y) {
    if (x > y)
    return x;
    else
    return y;
}
```

max, x и y – имена или идентификаторы.

int, if, return и else – ключевые слова, они не могут быть именами переменных или функций.

Переменные

Идентификатор – это имя, которое присваивается переменной, константе, функции и др.

Переменная - это именованная область памяти, в которой хранятся данные определенного типа. Имя служит для обращения к области памяти, в которой хранится значение. Во время выполнения программы значение переменной можно изменять. Перед использованием переменная должна быть описана.

У каждой переменной есть: имя; тип; значение;

Правила:

- имя переменной должно состоять из прописных или строчных латинских букв, цифр и символа _ (подчеркивание);
- имя переменной должно начинаться с буквы;
- символ подчеркивания - это буква;
- прописные и строчные буквы обрабатываются как разные;

Стандарт языка C++ не накладывает ограничений на длину имен переменных, но у конкретного компилятора оно может быть.

Таблица 1.3. Имена переменных

Корректные имена переменных (но не всегда удобные)	**Некорректные имена переменных**
i	12y
x1	Birth Day
Count	2X a-b
Day_Of_The_Birth	10t Adiós_Señora
ItIsOnlyVariableWithAVeryLongName	Exchange Rate

Дополнительная информация о стиле именования и соглашениях C++ в Основных руководствах по C++: C++ Core Guidelines

Тип - это атрибут (attribute) , который однозначно определяет, какие значения могут храниться внутри переменной, например, типы целых чисел (int) и с плавающей запятой (float).

Можно ввести только значение, совместимое с типом переменной. Переменной типа int может быть присвоено только целочисленное значение.

Переменной можно присвоить значение с помощью операции присваивания.

Присвоить – это значит установить текущее значение переменной.

Или: операция присваивания запоминает новое значение в ячейке памяти, которая обозначена переменной.

```
int x;     // объявить целую переменную x

int y;     // объявить целую переменную y

x = 0;      // присвоить x значение 0

y = x + 1;  // присвоить y значение x + 1, т.е. 1

x = 1;     // присвоить x значение 1

y = x + 1;   // присвоить y значение x + 1,  теперь уже
2

x=x+2;

z+=3;
```

допустима цепочка: a=b=c=d=0

Константы

Константы – это неизменяемая величина

Явная запись значения в программе – это константа.

Чаще используются символические константы.

Например: `const int BITS_IN_WORD = 32;`

Имя `BITS_IN_WORD` можно использовать вместо целого числа 32.

Преимущества такого подхода.

1. Имя `BITS_IN_WORD` (битов в машинном слове) дает хорошую подсказку, для чего используется данное число.

2. Если надо изменить эту константу, потребуется изменить только одно место в программе – определение константы.

Выражения

Выражения – это переменные, функции и константы, называемые операндами, объединенные знаками операций.

Операции:

- унарные – с одним операндом (- унарный минус);
- бинарные – с двумя операндами, например сложение или деление.

В Си++ есть операция с тремя операндами – условное выражение.

Примеры выражений:

```
X * 12 + Y  // значение X *на 12 и к результату + значение Y

val < 3          // сравнить значение val с 3

-9               // константное выражение -9
```

Выражение, после которого стоит точка с запятой – это оператор-выражение.

Его смысл в том, что компьютер должен вычислить выражение.

```
x + y – 12;  // сложить значения x и y и затем вычесть

a = b + 1;  // прибавить единицу к значению b и запомнить результат в переменной a
```

Типы данных

В любом ЯП каждая константа, переменная, результат вычисления выражения или функции должны иметь тип данных.

Тип данных – это множество допустимых значений, которые может принимать тот или иной объект, а также множество допустимых операций, которые применимы к нему.

Тип также зависит от внутреннего представления информации.

Данные различных типов хранятся и обрабатываются по-разному.

Тип данных определяет:

- внутреннее представление данных в памяти компьютера;
- объем памяти, выделяемый под данные;

- множество (диапазон) значений, которые могут принимать величины этого типа;
- операции и функции, которые можно применять к данным этого типа.

Необходимо определять тип каждой величины, используемой в программе для представления объектов. Обязательное описание типа позволяет компилятору производить проверку допустимости различных конструкций программы. От выбора типа величины зависит последовательность машинных команд, построенная компилятором.

ЯП С++ поддерживает типы данных:

- Базовые типы - предопределены стандартом языка, указываются зарезервированными ключевыми словами и характеризуются одним значением. Их не надо определять и их нельзя разложить на более простые составляющие без потери сущности данных. Базовые типы объектов создают основу для построения более сложных типов.
- Производные типы. Производные типы задаются пользователем, и переменные этих типов создаются как с использованием базовых типов, так и типов классов.
- Типы класса. Экземпляры этих типов называются объектами.

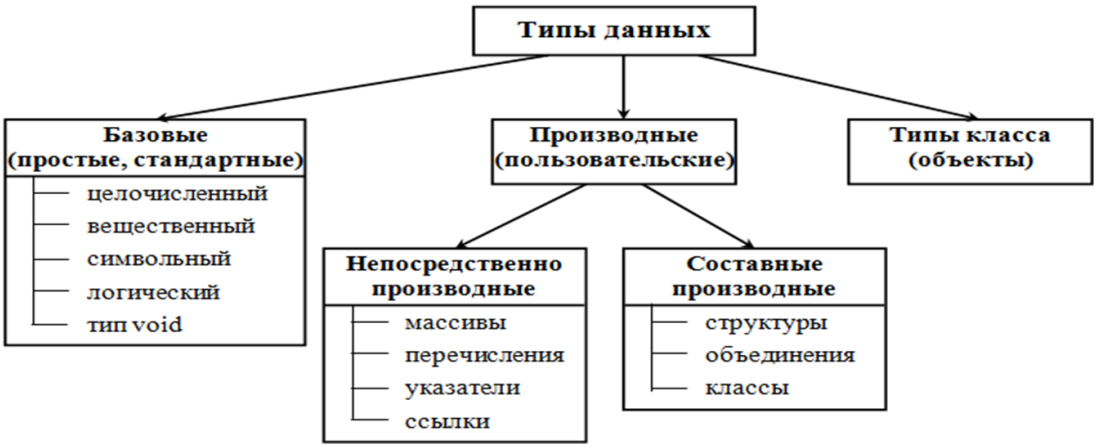


Рис. 1.6. Типы данных

Спецификаторы типа данных, уточняющие внутреннее представление и диапазон базовых типов:

- short (короткий)длина
- long (длинный)
- signed (знаковый) знак (модификатор)
- unsigned (беззнаковый)

Целочисленный (целый) тип данных (int) - для хранения целых чисел.

Вещественный (с плавающей точкой) тип данных (float и double) - для хранения вещественных чисел.

Применяются: типы данных float (с одинарной) и double (с двойной) точностью.

Символьный тип данных (char)

В стандарте C++ нет типа данных, который можно было бы считать действительно символьным. Для представления символьной информации есть два типа данных: char и wchar_t.

Переменная типа char хранит только один символ.

В памяти компьютера символы хранятся в виде целых чисел.

Соответствие между символами и их кодами определяется таблицей кодировки, которая зависит от компьютера и операционной системы.

В таблицах кодировки есть прописные и строчные буквы латинского алфавита, цифры 0, ..., 9, и некоторые специальные символы.

Распространенная таблица кодировки - таблица символов ASCII

(American Standard Code for Information Interchange – Американский стандартный код для обмена информацией).

В памяти компьютера символы хранятся в виде целых чисел, поэтому тип char является подмножеством типа int.

Под величину символьного типа char отводится 1 байт.

Тип char может использоваться со спецификаторами signed и unsigned.

Диапазон значений signed char: от –128 до 127.

unsigned char: диапазон от 0 до 255.

Символы с кодами от 0 до 31 – служебные, имеют самостоятельное значение только в операторах ввода-вывода.

Тип wchar_t - для работы с набором символов, для кодировки

которых недостаточно 1 байта, например в кодировке Unicode.

Размер типа wchar_t равен 2 байта.

Если в программе используют строковые константы типа

wchar_t, то их записывают с префиксом L,

например, L "Слово".

Пример:

```
char c='c';   char a,b;

char  r[]={'A','B','C','D','E','F','\0'};  char  s[]  =
"ABCDEF";
```

В ASCII "расстояние" между прописными и строчными буквами равно 32, и что 32 - это код пробела:

```
char character = 'A';

character += 32;

character -= ' ';
```

Примеры:

```
char character = 'A' + 32;  character = 'A'+' ';

character = 65 + ' ';    character = 97 - ' ';

character = 'a' - 32;  character = 'a' - ' ';
```

Ответы:

97, 97, 97, 65, 65, 65

Литерал

Литерал - это символ, который однозначно определяет его значение.

Примеры:

- character: это не литерал; это, вероятно, имя переменной; когда вы смотрите на него, вы не можете догадаться, какое значение в данный момент присвоено этой переменной;

- 'A': это литерал; вы можете сразу угадать его значение и знаете, что это литерал типа char;
- 100: это литерал (тип int);
- 100.0: это литерал (типа с плавающей запятой);
- i + 100: это выражение (комбинация литерала и переменной)

Логический (булевый) тип данных (bool)

В языке C++ используется двоичная логика (истина, ложь). Лжи соответствует нулевое значение, истине – единица. Величины данного типа могут также принимать значения true и false. Внутренняя форма представления значения false соответствует 0, любое другое значение интерпретируется как true. Под данные логического типа отводится 1 байт.

Пример:

```
int value1 = 0, value2 = 0;

bool answer = value1 >= value2;
```

Если value1 больше или равно value2, то answer = значению true (1).

Если value1 меньше value2, то answer = значению false (0).

Перечисляемый тип (enum)

Данный тип определяется как набор идентификаторов, являющихся именованными целыми константами, которым приписаны уникальные и удобные для использования обозначения.

Перечисления представляют собой упорядоченные наборы целых значений.

Переменная, которая может принимать значение из некоторого списка определенных констант, называется переменной перечисляемого типа или перечислением. Данная переменная может принимать значение только из именованных констант списка.

Именованные константы списка имеют тип int. Память, соответствующая переменной перечисления, – это память, необходимая для размещения значения типа int.

Пример:

```
enum year {winter, spring, summer, autumn};

enum week {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};
```

Тип void

Множество значений этого типа пусто.

Тип void имеет три назначения:

- указание о невозвращении функцией значения;
- указание о неполучении параметров функцией;
- создание нетипизированных указателей.

Тип void в основном используется для определения функций, которые не возвращают значения, для указания пустого списка аргументов функции, как базовый тип для указателей и в операции приведения типов.