История и эволюция C++

- Язык C. 1972 г. Денис Ритчи (Bell)
  цель – системное программирование (UNIX)
- Язык C++. 1983 г. Бьерн Страуструп (Bell)
  цель – реализация методологии ООП
- Язык Java. 1994 г. Патрик Наутон, Билл Джой, Джеймс Гослинг (Sun)
  цель – переносимость и надежность
- Язык C#. 2000 г. Андерс Хейлсберг и др. (Microsoft)
  цель – удобство и надежность
- Perl, PHP. цель – простота использования

Рассматриваемые языки [править | править код]

TIOBE ориентируется на полные по Тьюрингу языки, поэтому популярность, к примеру, XML, HTML или базовый SQL не исследуется. В то же время расширения SQL, такие как PL/SQL и T-SQL входят в индекс.[1]

Кроме Тьюринг-полноты, авторы индекса требуют от исследуемого языка наличие статьи в Википедии, в которой было бы чётко указано, что язык является языком программирования. По этому критерию в индекс не были включены Ruby on Rails, Ex
Android, Boost, Cocoa, ASP, AJAX.[1]

Язык года [править | править код]

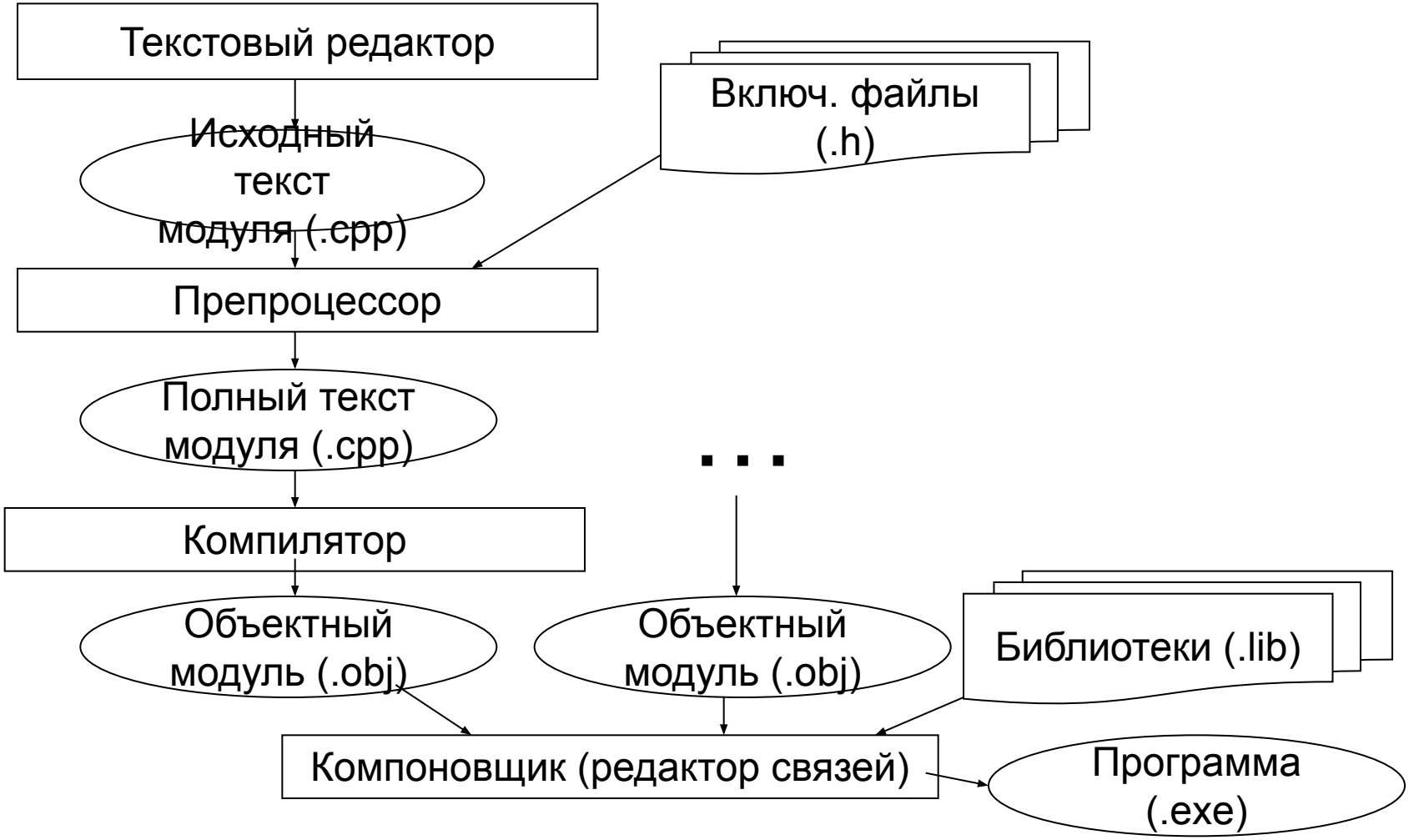
Каждый год, начиная с 2003, авторами TIOBE выбирается язык года (*Programming Language of the Year*):

- 2017 C
- 2016 Go
- 2015 Java
- 2014 Javascript
- 2013 Transact-SQL
- 2012 Objective-C
- 2011 Objective-C
- 2010 Python
- 2009 Go
- 2008 C
- 2007 Python
- 2006 Ruby
- 2005 Java
- 2004 PHP
- 2003 C++

Критика [править | править код]

Tim Bunce, автор Perl DBI[en], критиковал индекс и методы, используемые при ранжировании[5].

Этапы создания программы



Алфавит языка C++

- Прописные и строчные латинские буквы (различаются в именах), знак подчеркивания
- Цифры (0…9)
- Специальные знаки “ { } , | [ ] ( ) + - * / % \ ; ‘ : ? < = > ! & ~ ^ . #
- Разделители (пробел, табуляция, перевод строки)

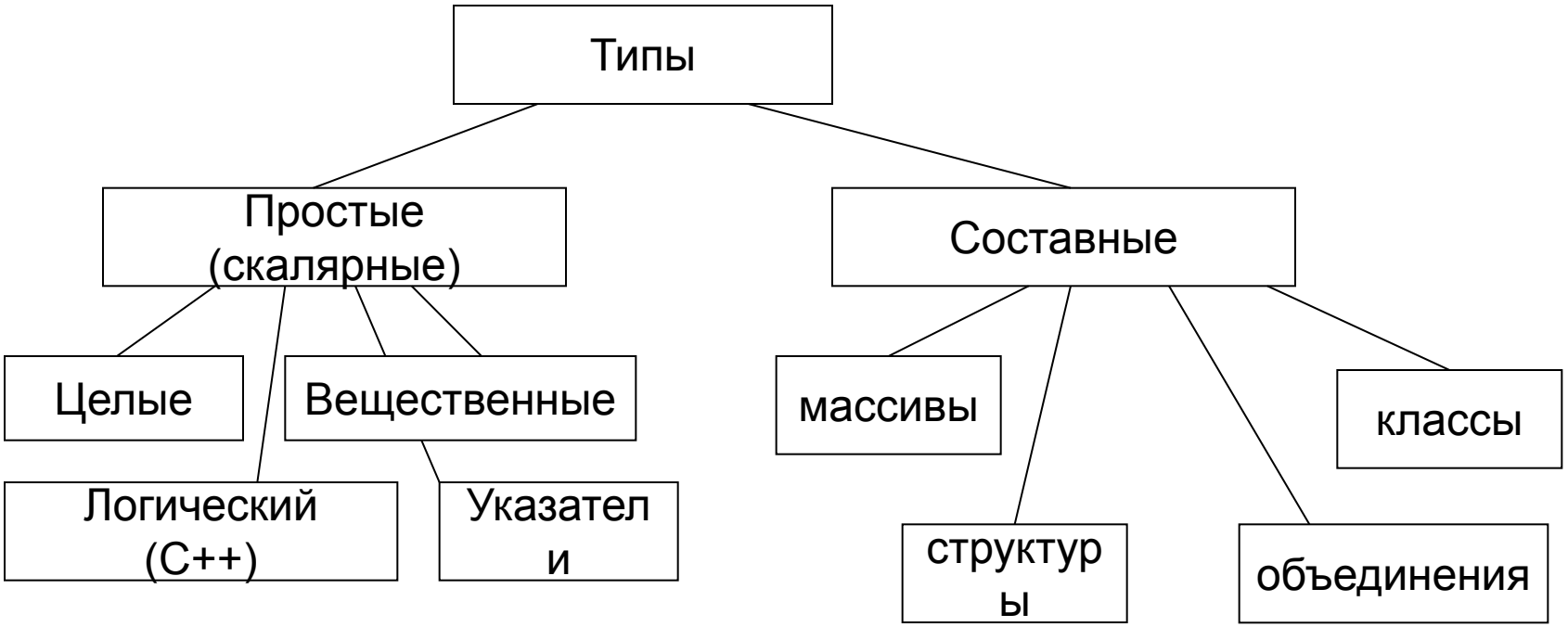
Лексемы C++

- Имена (не рекомендуется начинать с _)
- Ключевые слова
- Знаки операций (одно и двухсимвольные)
- Константы
- Разделители

Комментарии

- однострочные // комментарий
- многострочные /* длинные */

Типы данных C++



Базовые типы

	C/C++	Pascal (Delphi)
Целые	char	shortint
	int (short int)	integer
	unsigned char	byte
	unsigned int (short)	word
	long int	longint
	unsigned long int	cardinal
Вещест.	float	single
	double	double
	long double	extended

Специальные типы

- bool – логический (true/false) – в C++

  В C целое значение =0 – ложь,
  не равно 0 - истина
- void – пустой. Используется для обозначения функций без значений и нетипизированных указателей

Константы

- Целые:
  - десятичные 123, 0, 98
  - восьмеричные 01, 015
  - шестнадцатиричные 0xA1, 0X00FF
- Вещественные 5.8, .2e-3
- Символьные ‘A’, ‘xy’, ‘\n’, ‘\123’, ‘\\’
- Строковые “привет”, “1 \n 2”

Структура программы

<директивы препроцессора>

<функции>

Функция имеет вид

<тип> <имя> (<список параметров>)

{ <операторы>

}

Выполнение начинается с функции main

Пример программы

```
#include <iostream.h>
int main()
{ int a, b; //описание переменных
  cin >> a >> b; //ввод
  cout << “сумма” << a+b; //вывод
  return 0; //возврат
}
```

Описание переменных

[класс памяти] [const] тип имя [инициализатор]

Модификатор const используется при описании констант.

Инициализатор задает начальное значение переменной в виде “= значение” или в круглых скобках “(значение)”.

Примеры:

```
long int n =1000, k;
char c(‘A’);
const float pi = 3.1415926;
```

Классы памяти

Класс памяти определяет область видимости и время жизни переменной.

Область видимости может быть локальной (внутри блока) и глобальной (во всем файле).

Время жизни – временным (до конца блока) и постоянным (до выхода из программы).

Класс auto (по умолчанию) – автоматическая переменная. Локальная и временная.

Класс register – регистровая. Аналог auto, только переменная хранится в регистре процессора.

Класс static– статическая переменная. Постоянная. Может быть локальной и глобальной.

Класс extern– внешняя переменная. Определяется в другом файле программы.

Операции

- Арифметические **+ - * / %** Преобразования автоматические
- Увеличение и уменьшение **++ --**
  префиксное (++a) возвращает новое значение,
  постфиксное (a++) возвращает старое значение.
- Отношения **< > <= >= == !=** Результат – true или false
- Логические **&&** (и) **||** (или) **!** (отрицание)
- Присваивание **=** Результат – выражение. Выполняется справа налево: a = b = c = 5
- Присваивание с операцией **+= *=** и т.д.
  a+=b эквивалентно a = a + b
- Последовательное выполнение **,** результат - самое правое выражение
- Условная операция условие ? выр.1 : выр. 2
  max = (a>b) ? a : b;

Операторы

- Выражение
- Условный
- Выбора
- Циклы
- Передачи управления

В конце оператора всегда ставится ;

Условный оператор

if (выражение) опер.1; [else опер.2;]

Пример. Найти максимум и минимум из двух чисел.

```
int main()
{ int a,b,min, max; cin >> a >> b;
  if (a<b) { min = a; max = b;}
   else {min = b; max = a; }
  cout << min << max;
  return 0;
}
```

Условный оператор. Типичные ошибки

- Отсутствие фигурных скобок

```
if (a<b)  min = a; max = b;
```

- Использование = вместо ==

```
if (a=5) cout << a;
```

- Проверка диапазона

```
if (-1 <= x <=1) cout << “есть arcsin”;
Правильно так:
if (-1 <= x && x <=1)
      cout << “есть arcsin”;
```

Оператор выбора

```
switch (выражение) {
    case конст.1: список операторов 1
    case конст.2: список операторов 2
    …
    default: операторы
}
```

Производит переход на первый оператор из списка, соответствующего константе, равной значению выражения.

Операторы из нижестоящих списков будут выполняться, если не сделать выход оператором break.

Пример на оператор выбора

По номеру месяца определить время года

```
…
switch (m) {
case 1: case 2: case 12:
     cout<<“Зима”; break;
case 3: case 4: case 5:
     cout<<“Весна”; break;
case 6: case 7: case 8:
     cout<<“Лето”; break;
default: cout<<“Осень”;
}
```

Цикл с предусловием

while (выражение) оператор

Цикл выполняется так:

1. Вычисляется выражение
2. Если оно истинно (не 0) выполняется оператор
3. Снова вычисляется выражение
4. Если оно ложно – выход из цикла.

Пример: вычисление факториала n

```
f = k = 1;
while (k<=n) f *= k++;
```

Цикл с постусловием

do оператор while (выражение)

Цикл выполняется так:

1. Выполняется оператор
2. Вычисляется выражение
3. Если оно истинно (не 0) снова выполняется оператор
4. Если оно ложно – выход из цикла.

Пример: вычисление факториала n

```
f = k = 1;
do f *= k++; while (k<=n);
```

Цикл с параметром

for (инициализация; условие выполнения;
модификация) оператор;

Инициализация выполняется перед началом цикла

Модификация - в конце каждой итерации

Пример: вычисление факториала

```
for (int k = f = 1; k<=n; k++) f *= k;
```

либо `for (int k = f = 1; k<=n; f *= k++);`

Бесконечный цикл for (; ;) { ….}

Операторы передачи управления

- `goto метка;`
  Помеченный оператор –
  метка: оператор;
- `break;` выход из цикла или оператора выбора
- `continue;` переход к следующей итерации цикла
- `return [выражение];` выход из функции с возвратом значения

Указатели

Содержит адрес памяти, в которой хранятся данные определенного типа

```
тип * имя;
```

Звездочка относится к имени

```
int a, *b, c;
```

Безтиповый указатель `void *p;`

Константные указатели

Модификатор const относится либо к указателю, либо к значению

```
int i;
const int c = 1;
const int *pc = &c; //указатель на константу
int* const pc = &i; //указатель-константа
```

Инициализация указателей

- с помощью операции & (адрес)

  int a=5;

  int * p = &a;
- значением другого указателя

  int * r = p;
- явным адресом памяти

  char *vp = (char *)0xB8000000;
- пустым значением (нулем)

  int * r = 0;

Динамические переменные

Создание

```
int *n = new int;
int *m = new int (10); // *m=10
int *r = new int [10]; // массив
```

Удаление

```
delete n;
delete [ ] r;
```

Операции с указателями

- доступ к переменной, на которую указывает указатель *

  *n=100; r[1] = 20;

- Арифметические операции (прибавление константы, вычитание, увеличение, уменьшение) учитывают размер данных

  r++; // увеличивает r на 2

Ссылки

Ссылка – синоним имени, указанного при ее инициализации

```
тип & имя;
int a;
int &b=a;
```

Массивы

Массивы в языке C++ описываются следующим образом:

```
тип_элементов имя [размер];
```

Размер массива задается константным выражением. Индексы элементов – целые числа, начиная с нуля. Например,

объявление

```
int a[100];
```

определяет массив a с элементами a[0], a[1], …, a[99].

Инициализация массивов

При описании массива можно указать начальные значения элементов через запятую в фигурных скобках. При инициализации размер массива можно не указывать. Например,

```
int p[] = {0,1,2,3,4,5};
```

К элементам массива можно обратиться, указав имя и в квадратных скобках индекс. Контроль выхода за пределы массива не производится.

Вывод на экран

```
for (int i=0; i<100; i++)  cout <<a[i]<<" ";
```

Переменная типа массив является константным указателем на первый элемент массива. Выражения a[i] и *(a+i) эквивалентны.

Многомерные массивы

Многомерные массивы описываются как массивы, элементами которых также являются массивы. Следующий пример описывает матрицу из m строк и n столбцов.

```
const int m = 4, n = 5;
float matr[m][n];
```

Инициализация

```
int mas [3][2] = { {1,1}, {0,2}, {1,0}};
```

Обработка

```
for (i=0; i<m; i++) {
for (j=0; j<n; j++)  cout << a[i][j] <<‘ ‘;
      cout << endl; }
```

Динамические массивы

Массивы, размер которых меняется во время выполнения программы, описываются как указатели и создаются с помощью операции new. Следующий фрагмент программы описывает создание массива, размер которого вводится с клавиатуры.

```
int *a, k;
cin >> k;
a = new int [k];
for (int i=0; i<k; i++)  cin >>a[i];
```

Строки

Специальный строковый тип в языке Си отсутствует.

Строка в Си представляется одномерным массивом элементов типа char; последним элементом массива должен быть символ ‘\0’ (такой массив называется ASCIIZ – строкой).

Строки можно вводить и выводить с помощью стандартных потоков или функций ввода-вывода gets(s) и puts(s).

Следующий пример демонстрирует ввод и вывод строк

```
char s[100], p[]=”Введено ”;
cin >>s;
cout <<p<<s;
```

Строка может быть описана как указатель на char и размещена в динамической памяти.

Функции для работы со строками

Некоторые функции работы со строками (Заголовочный файл <*string.h*>)

1. Длина строки

int strlen(char*< строка>)

2. Конкатенация.

char* strcat(char*< строка-приемник>, char*< строка-источник>)

3. Копирование

char* strcpy(char*< строка-приемник>, char*< строка-источник>)

4. Сравнение

int strcmp(char*< строка1>, char*< строка2>)

Возвращает: 0, если <строка1> совпадает со <строкой 2>,
число <0, если <строка1> **<** <строки 2>,
число >0, если <строка1> **>** <строки 2>.

5. Поиск символа в строке. Возвращает указатель на найденный символ.

char* strchr(char*< строка>, int <символ>)

6. Поиск подстроки в строке. Возвращает указатель на найденную строку.

char* strchr(char*< строка>, char*< подстрока>)

Функции преобразования

Некоторые функции преобразования данных “строка ↔ число” (Заголовочный файл *<stdlib.h>*)

Преобразование строки в double, int, long

```
double atof (char*< строка>)
int atoi (char*< строка>)
long atol (char*< строка>)
```

Преобразование int, long, unsigned long в строку

```
char*  itoa (int <число>, char*< строка>, int
   <основание сист. сч.>)
char*  ltoa (long <число>, char*< строка>, int
   <основание сист. сч.>)
char* ultoa (unsigned long <число>, char*< строка>, int
   <основание сист. сч.>)
```

Работа со строками как с указателями

Пример: копирование строки s в строку t

```
Неэффективный способ
for (int i = 0; i<=strlen(s); i++) t[i] = s[i];

Эффективный способ
char *st = s, *dt = t;
while (*st != 0) *dt++ = *st++;
*dt=0;
```