Элементы языка C++

Основы информатики и программирования

Язык C++ и вопросы совместимости с C

Тесно связаны, но имеют существенные различия!

C не является подмножеством C++

- нетривиальные программы на C не компилируются на C++ без изменений

Си:

```
void *ptr;
/* Неявное преобразование из void* в int* */
int *i = ptr;
int *j = malloc(5 * sizeof *j);
```

C++:

```
void *ptr;
int *i = (int *)ptr;
int *j = (int *)malloc(5 * sizeof *j);
```

Приведение типов в C++

Приведение типов в C++

Статическая типизация

Приведение типов в стиле языка C (new_type)exp

Функциональная нотация int i = int(a);

Явное и неявное приведение типов
- Примеры неявных приведений
- В арифметических операциях
- Безопасные
- Опасные (потеря точности, ошибка)

```
int c = true;        // 1
double d = false;    // 0
```

```
int a = 3.4;         // 3
int b = 3.6;         // 3
```

```
unsigned char a = -5;           // 251
unsigned short b = -3500;       // 62036
unsigned int c = -50000000;     // 4244967296
```

```
bool a = 1;      // true
bool b = 0;      // false
bool c = 'g'; // true
bool d = 3.4;    // true
```

Приведение типов в C++

static_cast<type>(value)

Явные приведения типов

```
#include <iostream>

int main()
{
    double sum {100.2};
    unsigned int hours {8};
    unsigned int revenuePerHour { static_cast<unsigned int>(sum/hours) };  // revenuePerHour = 12
    std::cout << "Revenue per hour = " << revenuePerHour<< std::endl;
}
```

```
#include <iostream>

int main()
{
    double sum {100.2};
    unsigned int hours {8};
    unsigned int revenuePerHour { (unsigned int)sum/hours};     // revenuePerHour = 12
    std::cout << "Revenue per hour = " << revenuePerHour<< std::endl;
}
```

const_cast<type>(value)

Удаляет или добавляет квалификаторы const и volatile с исходного типа данных.

```
p = const_cast<int*>(pc);
```

const - константность
volatile - значение переменной может меняться без явного выполнения присваивания

```
pc= const_cast<const int*>(p);
pv = const_cast<volatile int*>(p);
```

Приведение типов в C++

reinterpret_cast<type>(value)

Используется для приведения несовместимых типов

Приведение типов в C++

dynamic_cast<type>(value)

Предназначен для приведения полиморфных типов по иерархии наследования

Пространства имен

Пространства имен

```
1  namespace first
2  {
3      int a;
4  }
5
6  namespace second
7  {
8      double a;
9  }
10
11 int main ()
12 {
13     first::a = 2;
14
15     second::a = 6.453;
16     // ...
17
18 }
```

Учебники

Группировка функционала в отдельные контейнеры — блок кода со своим набором компонент(функция, классов и т.д.)

```
namespace имя_пространства_имен
{
    // код пространства имен
}
```

Если пространство имен не указано - умолчанию применяется **глобальное пространство имен.**

!!! функция main должна быть определена в глобальном пространстве имен.

std::cout

Одно пространство имен может содержать другие пространства.

std::string message{"hello"};

Пространства имен

Директива using

Позволяет ссылаться на любой компонент пространства имен без использования его имени.

Так как могут возникнуть конфликты имен, можем подключать отдельные компоненты:

```
using console::print;
```

Или создать псевдоним:

```
namespace mes = console::messages;
```

```
#include <iostream>

namespace console
{
    const std::string message{"hello"};
    void print(const std::string& text)
    {
        std::cout << text << std::endl;
    }
}

using namespace console;    // подключаем все компоненты пространства консо

int main()
{
    print(message); // указывать пространство имен не требуется
}
```

Заголовочные файлы стандартной библиотеки

Формат заголовочных файлов

```cpp
#include <cstdio>
#include <cmath>
#include <algorithm>

using namespace std;

int main ()
{
    double a, b;

    a = 1.2;
    b = 10.5;
    a = sin (a);

    printf("%f %f", a, max(a, b));

    return 0;
}
```

Ссылки

Ссылка как синтаксический сахар

```
1  int main ()
2  {
3       double a = 3.1415927;
4
5      double &b = a;
6
7      b = 89;
8
9      printf("a = %d\n", a);
10
11     return  0;
12 }
```

!!! Инициализация ссылки обязательна

Ссылки

Псевдоним ссылочного типа

```
1  using RT = T&;
2
```

```
1  typedef T& RT;
2
```

Копия ссылки

```
1 T x;
2 T &rx = x;
3 T &rx2 = x;
```

Ссылки на функцию

```
void Foo(int);
void(&rf)(int) = Foo;
rf(42); // тоже самое, что и Foo(42);
```

Ссылки на массив

```
int a[4];
int(&ra)[4] = a;
```

Ссылки в параметрах

```
1  void change (double &r, double s)
2  {
3      r = 100;
4      s = 200;
5  }
6
7  int main ()
8  {
9      double k, m;
10
11     k = 3;
12     m = 4;
13
14     change (k, m);
15
16     printf("k = %d, m = %d\n", k, m);
17
18     return 0;
19 }
```

Леводопустимые выражения с ссылками

```
double *biggest (double *r,
double *s)
{
    if (*r > *s) return r;
    else         return s;
}

int main ()
{
    double k = 3;
    double m = 7;

    (*(biggest (&k, &m))) = 10;
    (*(biggest (&k, &m))) ++;
}
```

```
double *silly_function ()
{
    static double r = 342;
    return &r;
}

int main ()
{
    double *a;

    a = silly_function();

    double &b = *a;

    b += 1;
    b = b * b;
    b += 4;
    //...
}
```

Ссылки и указатели

Не являются полностью взаимозаменяемыми.

Указатель может иметь значение nullptr, ссылки не могут быть нулевыми.

Нельзя создавать массивы ссылок и нет ссылочного аналога нетипизированного указателя void*.

Указатели оказаться незаменимыми в низкоуровневых решениях, где используется арифметика указателей.

При перегрузке операторов также часто нельзя обойтись без параметров ссылочного типа.

Указатели являются С-архаизмом

Исключения

Исключения

```
1     char zero []      = "zero";
2     char pair []      = "pair";
3     char notprime [] = "not prime";
4     char prime []     = "prime";
5
6     try
7     {
8        if (a == 0) throw zero;
9        if ((a / 2) * 2 == a) throw pair;
10       for (int i = 3; i <= sqrt (a); i++)
11       {
12          if ((a / i) * i == a) throw
13          notprime;
14       }
15       throw prime;
16    }
17    catch (char *conclusion)
18    {
19       fprintf(stderr, "A mistake was
         happen!\n");
      }
```

Перегрузка функций и операций

Значения аргументов по умолчанию

```
1  double test (double a, double b = 7)
2  {
3      return a - b;
4  }
5
6  int main ()
7  {
8      cout << test (14, 5) << endl;    // Displays 14 - 5
9      cout << test (14) << endl;       // Displays 14 - 7
10
11     return 0;
12 }
```

Перегрузка функций

```
1  double test (double a, double b)
2  {
3      return a + b;
4  }
5
6  int test (int a, int b)
7  {
8      return a - b;
9  }
```

```
1  int main ()
2  {
3      double    m = 7,  n =
4  4;
4      int       k = 5,  p =
3;     printf("%f %d", test(m, n),
5             test(k, p));
6
7      return  0;
8
9
10 }
```

Перегрузка операций

```
1  struct vect
2  {
3      double x;
4      double y;
5  };
6
7  vect operator * (double a, vect b)
8  {
9      vect r;
10
11     r.x = a * b.x;
12     r.y = a * b.y;
13
14     return r;
15 }
```

```
1  int main ()
2  {
3      vect k,
   m;
4      k.x = 2;
5      k.y = -1;
6
7      m = 3.1415927 * k;
8
9      printf("(%f, %f)\n", m.x, m.y);
10
11     return 0;
12
13 }
```

Шаблоны

Шаблоны

```
template <class ttype>
ttype minimum (ttype a, ttype b)
{
    ttype r;

    if (b < a) r = b; else r = a;

    return r;
}
```

```
int main ()
{
    int i1, i2, i3;
    i1 = 34;
    i2 = 6;
    i3 = minimum (i1,
i2);
    double d1, d2, d3;
    d1 = 7.9;
    d2 = 32.1;
    d3 = minimum (d1,
    d2);
    // ...

}
```

Динамическая память

Выделение динамической памяти

Общий синтаксис:

```
pointer-variable = new data-type;
```

Примеры:

```
int *p = NULL;
p = new int;
```

или

```
int *p = new int;
```

Инициализация

```
int *p = new int(25);
float *q = new float(75.25);
```

Выделение памяти под массив

Общий синтаксис:

```
pointer-variable = new data-type[size];
```

Примеры:

```
int *p = new int[10];
```

Инициализация

```
int *p = new int(25);
float *q = new float(75.25);
```

Инициализация (C++11)

```
int *p = new int[10]{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

Ошибка выделения памяти

Если необходимый объем памяти выделить невозможно, будет выброшено исключение std::bad_alloc:

```
1  int main ()
2  {
3      try
4      {
5        int* myarray= new int[1000];
6      }
7      catch (exception& e)
8      {
9        cout << "Standard exception: " << e.what() << endl;
10     }
11     return 0;
12 }
```

Ошибка выделения памяти

Однако, можно обработать в стиле C:

```
1 int *p = new(nothrow) int;
2
3 if (!p) {
4         fprintf(stderr, "Memory allocation failed\
n");
5         exit(EXIT_FAILURE);
6 }
```

Освобождение динамической памяти

Общий синтаксис в случае одного объекта:

```
1 delete pointer-variable;
```

Примеры:

```
1 delete p;
2 delete q;
```

Общий синтаксис в случае динамического массива:

```
1 delete[] pointer-variable;
```

Инициализация

```
1 delete[] p;
```

Учебники

https://metanit.com/cpp/