Лекция 4. Встроенные типы, инструкции и операторы.

Встроенные типы

Группа	Тип	Литералы
Логический	`bool`	`true, false`
Символьный	`char, wchar_t, char16_t`	`‘a’, ‘\n’, ‘\t’, L’x’, u‘€’`
Целый	`int`	`0, 5, 0xef, 045, 0b101010`
	`short [int]`	`32767s`
	`long [int]`	`56L, -1234567L`
	`unsigned [int]`	`765u`
Вещественный	`long double`	`6.626e-34L`
	`double`	`2.71, .56, 2., 6e24`
	`float`	`3.1415f`

Размеры типов

```
1   1            == sizeof(char)    <= sizeof(short) <= sizeof(int) <= sizeof(long)
2   1            <= sizeof(bool)    <= sizeof(long)
3   sizeof(char) <= sizeof(wchar_t) <= sizeof(long)
4   sizeof(float)<= sizeof(double)  <= sizeof(long double)
5   sizeof(T)    == sizeof(signed T)== sizeof(unsigned T)
```

- Ограничения типа: `std::numeric_limits<type>`
- Не полагайтесь на размер, для этого есть `uint16_t, int64_t` и т.д.

Явное приведение типов

- Базовые типы неявно приводятся друг у другу. Многие с потерей значимости (warning).

```
 1.  float*       pfv;
 2.  const int*   piv;
 3.  short        sv;
 4.
 5.  int* a = (int*)pfv; // 1. C-style
 6.  char b(sv);         // 2. constructor
 7.
 8.  double pi  = 3.1415;
 9.  int    ipi = static_cast<int>(pi);
10.  //---
11.  int* c = reinterpret_cast<int*>(pfv);
12.  //---
13.  int* d = const_cast<int*>(piv);
14.  //---
15.  derived* d = dynamic_cast<derived*>(base);
```

Тип void

```
1.  void *p  = 0, *q = 0;
2.  bool  eq = (p == q);
3.  void  do_smth(int value);
4.  int   main   (void); // C-style
5.  void  nothing;       // wrong!
```

- Указатель на void:

```
1.  void send(const void* data, size_t size);
2.  //...
3.  X* ptr = ...;
4.  send(ptr, sizeof(X));
5.  //---------------------------------------
6.  void receive_X(const void* data, size_t size)
7.  {
8.      Assert(size == sizeof(X));
9.      X const* ptr = static_cast<X const*>(data);
10. }
```

null terminated strings

```
1.  const char str     [] = "Hello";              // native type
2.  char       can_edit[] = "can change this string";
3.  char*      c_style    = "old style string";  // don't change it
4.
5.  const char ml [] = "this is"
6.                     "multiline\
7.                      string";
```

- Одинаковые литералы могут размещаться в одном месте
- Спец. символы: ‘\t’ ‘\r’ ‘\n’ ‘\a’

null terminated string	std::string
char* a = “…”, *b = “…”;	string a = “…”, b = “…”;
strcmp(a, b) == 0;	a == b;
strcpy(a,b);	a = b;

Перечисления enum (до C++11)

```
1.  enum msg_type
2.  {
3.      mt_setup,
4.      mt_request = 0x10,
5.      mt_response
6.  };
```

- Нет неявного приведения из целого типа
- Пользовательский тип - можно, например, переопределить операторы

```
1.  int process_msg(msg_type type, const void* data)
2.  {
3.      switch(type)
4.      {
5.      case mt_setup   : return process_setup   (*static_cast<setup   *>(data));
6.      case mt_request : return process_request (*static_cast<request *>(data));
7.      case mt_response: return process_response(*static_cast<response*>(data));
8.      default: throw std::runtime_error("unknown message");
9.      }
10. } // hidden error! where?
```

Синоним имени typedef

```
1.  typedef
2.      std::map<std::string, student>
3.      students_t;
4.
5.  // the same:
6.  using students_t = std::map<std::string, student>;
7.
8.  // works even for templates:
9.  template<class key>
10. using students_t = std::map<key, student>;
```

- Задает синоним имени, не задает новый тип
- Делает код короче – проще читать
- Платформонезависимые типы
- «Протаскивает» тип через шаблон
- Имеет внутреннюю компоновку

И еще раз про объявления

Необязательный спецификатор (e.g. extern, virtual)	Базовый тип	Объявляющая часть	Необязательный инициализатор
---	`const char`	`array[]`	`= “Hello, World!”`

- *Идентификаторы*: буквы, цифры, _ (начало не с цифры). Длина не ограничена стандартом, но часто реализацией
- Объявление через запятую – остается только базовый тип (не делайте так!):

```
1.  int* x, y = 1, *const z = 0;
2.  // int* x; int y = 1; int *const z = 0;
```

Области действия и видимости

```
1.  int uno; // = 0
2.
3.  void foobar()
4.  {
5.      int  due; // = 'trash‘
6.      int& quatro = due;
7.
8.      int uno = 1;
9.      ::uno = 2;
10.     // uno != ::uno
11.
12.     if (true)
13.     {
14.         double due = 8.31;
15.         T tre = func(&tre);
16.         quatro = 2; // due <line 5> == 2
17.     }
18. }
```

Операторы

Семантика	Синтаксис
Область видимости	`[namespace|class]::name`
Выбор члена	`object.member` `ptr->member`
Доступ по индексу	`pointer[]`
Вызов функции	`expr(expr-list)`
Постфиксный инкремент	`lvalue++`
Идентификатор типа	`typeid({expr|type})`
Преобразование типов	`static_cast<type>(expr)`
Взятие размера	`sizeof({expr|type})`
Унарные префиксные операторы	`~lvalue`
Разыменование	`*expr`
Выделение/освобождение памяти	`new type (expr-list)`

Операторы, часть 2

Семантика	Синтаксис
Бинарные арифметические операторы	expr * expr
Сдвиги	expr << expr
Бинарные операторы сравнения	expr < expr
Бинарные логические операторы	expr || expr
Условное выражение	expr ? expr : expr
Присваивания	expr = expr expr <<= expr expr += expr
Генерация исключения	throw expr
Запятая	expr, expr

Выражения

- Результат:
  - Расширение к большему (например, *)
  - Логическое выражение bool
  - Там, где можно, lvalue (a=b=c)
- Порядок вычисления в общем случае не определен

```
1.  f(a) + f(b)
2.  // what would you get here?
3.  if (var & mask == 0){...}
4.  if (0<=x<=99)       {...}
```

- Исключение: ленивое вычисление &&,|| на встроенных типах, а также оператор ,

Инкремент/Декремент

- Префиксный: изменил, вернул новое значение
- Постфиксный: изменил, вернул старое значение

```
1.  int a = ++x;
2.  int b = x += 1;
3.
4.  int c = x++;
5.  int t;
6.  int d = (t = x, x += 1, t);
7.
8.  // prefer prefix ++
9.  for (auto it = cont.begin(); it != cont.end(); ++it);
```

Инструкция (statement) выбора

```
1.  if (ptr p = get_pointer(...))
2.      if (connected(p))
3.          disconnect(p);
4.      else
5.          connect   (p);
```

- switch
  - окончание ветки: break, return, throw, exit(0)
  - для определения переменных потребуются скобки {}

```
1.  switch(id)
2.  {
3.  case btn_yes:
4.      apply_changes();
5.  case btn_no :
6.      close_doc    ();
7.      break;
8.  case btn_cancel:
9.      continue_editing();
10.     break;
11. default:
12.     Log("Unexpected btn " << id);
13. }
```

Циклы

```
1.
2.  for  (init; condition; modification) {}
3.  while(condition) {}
4.  do{} while(condition)
5.
6.  // examples
7.  for (size_t i = 0, size = strlen(src); i < size; ++i)
8.      dest[i] = src[i];
9.
10. while(*dest++ = *src++); // don't do that
```

- Управление циклом:
  - досрочный выход: `break, return, throw, exit(0)`
  - продолжение: `continue`

Комментарии

```
1.  // one line
2.  /* multi
3.       line */
```

- Лучше вообще без комментариев, чем нерелевантные комментарии
- Если можно выразить кодом, не пишите комментарии
- Сложный алгоритм – дай ссылку на статью
- Желательно написать:
  - сложный код (например, оптимизация)
  - а-ля разметка (окончание namespace)
  - в заголовке файла (зачем он, copyright)

Вопросы?