14. ПРАВИЛА ПРЕОБРАЗОВАНИЯ СТАНДАРТНЫХ ТИПОВ

В любых случаях выполняются два преобразования:

 имя массива преобразуется к указателю на его первый элемент; имя функции преобразуется к указателю на эту функцию.

14.1. Явные преобразования

Разрешены любые преобразования стандартных типов одного к другому. При преобразовании более длинного типа к более короткому происходит потеря разрядов; при преобразовании более короткого целочисленного типа к более длинному свободные разряды заполняются 0 (если короткий тип – беззнаковый) или происходит размножение знакового разряда (для типа со знаком).

Разрешены любые преобразования типов указателей, а также ссылок. Явное преобразование типов делается посредством операции приведения типов (cast). Всего есть 5 типов таких операторов:

1. c-style cast:

(имя_типа**)** операнд // Традиционная форма;
или имя_типа **(**операнд**)** // функциональная форма.

Здесь **имя_типа** задаёт тип, а **операнд** является величиной, которая должна быть преобразована к заданному типу.

Отметим, что во второй форме **имя_типа** должно быть простым идентификатором, например, полученным с помощью **typedef**:

```
double d = (double)5; int i =
int(d); int *ip = &i; float *fp =
(float*) ip; typedef float* FP;
fp = FP(ip);
```

c-style cast не проверяется во время компиляции, поэтому может быть неправильно использован, например, при преобразовании типов const или изменении типов данных без учёта их диапазонов (что может привести к переполнению).

2. static_cast

Операцию static_cast лучше всего использовать для преобразования одного фундаментального типа данных в другой:

```
int i1 = 13; int i2 = 37; double x = static_cast<double> (i1) /
i2; // Такой синтаксис!
```

Основным преимуществом static_cast является проверка преобразования во время компиляции, что уменьшает возможность возникновения непреднамеренных проблем. static_cast имеет меньшее влияние, чем C-style cast, поэтому, используя static_cast, нельзя случайно изменить тип const или сделать другие преобразования, которые не имеют смысла.

3. const_cast
4. reinterpret_cast

Что касается этих двух операций, то их желательно избегать, потому что они полезны только в редких случаях и могут создать немало проблем, если их использовать неправильно.

5. dynamic_cast

dynamic_cast обычно применяется в связи с использованием указателей и наследования в классах.

14.2. Неявные преобразования стандартных базовых типов

Для стандартных базовых типов компилятор может выполнять любые преобразования одного типа к другому:

```
int i = 'A';                // i = 65;
char c = 256;          // Теряются 8 старших битов; c станет равно '\0';
int j=-1; long l = j;
long m = 32768;       // Двоичное представление числа 32768
// содержит единственную единицу в 15 разряде. short int k = m;
// k = -32768, так как 15-й разряд для short –
                              // знаковый.
unsigned u=m;       // u = 32768
double d = 0.999999; long n =
d;                  // n = 0
```

При выполнении небезопасных неявных преобразований типов компилятор будет выдавать предупреждения. Например,

```
int i = 53;
char ch = i;                // неявное преобразование
```

Преобразование переменной типа int (4 байта) в тип char (1 байт) потенциально опасно – компилятор выдаст предупреждение. Чтобы сообщить ему, что вы намеренно делаете что-то, что потенциально

опасно (но хотите сделать это в любом случае), используйте оператор static_cast:

```
int i = 49;
char ch = static_cast<char>(i);
```

В следующем случае компилятор предупредит, что преобразование из double в int может привести к потере данных:

```
int i = 70; i
= i / 3.6;
```

Чтобы сообщить компилятору, что мы сознательно хотим сделать это, следует написать:

```
int i = 70;
i = static_cast<int>(i / 3.6);
```

При **выполнении арифметических операций** также происходит неявное преобразование типов. Правила здесь такие:

а) типы char, short, enum преобразуются к типу int, a unsigned short – к unsigned int; тип float преобразуется к double;

б) затем, если один из операндов имеет тип long double, то и второй преобразуется к long double;

в) иначе, если один из операндов имеет тип double, то и второй преобразуется к double;

г) иначе, если один из операндов имеет тип unsigned long, то и второй преобразуется к unsigned long;

д) иначе, если один из операндов имеет тип unsigned, то и второй преобразуется к unsigned;

е) иначе, если один из операндов имеет тип long, то и второй преобразуется к long;

ж) иначе оба операнда имеют тип int. Пример

1:

```
int g = 10, t = 5; double
t2 = t * t / 2;
double s = g * t2;                    // s станет равно  120; double s0
= g * t* t / 2.0;          // s0 станет равно 125.
```

Пример 2:

Функция **atoi** (упрощенная), которая ставит в соответствие строке цифр её числовой эквивалент:

```
int atoi ( char s[ ] ){int i, n = 0; for (i = 0; s[i] >= '0' && s[i] <= '9';
i++) n = 10 * n + s[i] - '0';                    // Преобразование char
в int. return n;}
```

14.3. Преобразование производных стандартных типов

Для указателей разрешено неявное преобразование указателя на любой тип к указателю на тип void. Все другие преобразования должны быть явными.

```
int *ip; void *vp = ip; ip = vp;  // Ошибка!
ip = (int*) vp;        // Теперь верно. double
*fp = ip;       // Ошибка. float *fp =
(double*) ip;          // Верно.
```

Константа 0 может быть неявно преобразована к указателю на любой тип. При этом гарантируется, что такой указатель не будет ссылаться ни на один объект. Значение стандартной константы NULL равно 0 для всех видов указателей.