КОНСТРУИРОВАНИЕ ПРОГРАММ

Лекция № 04. Типы, выражения и операторы языка С

Преподаватель: Поденок Леонид Петрович, 505а-5

+375 17 293 8039 (505а-5)

+375 17 320 7402 (ОИПИ НАНБ)

prep@lsi.bas-net.by

ftp://student:2ok*uK2@Rwox@lsi.bas-net.by/

Кафедра ЭВМ, 2021

2021.09.22

Оглавление

Взаимодействие и правила поведения на занятиях....3
Типы....4
Типы в языке программирования С....5
Булев (логический) тип....5
Стандартные целочисленные типы со знаком....6
Стандартные целочисленные типы без знака....7
Целочисленный ранг преобразования типов....8
Вещественный тип с плавающей точкой....9
Комплексный тип с плавающей точкой....10
Символьный тип....11
Перечисление....12
Тип void....13
Массив....14
Структурный тип....15
Тип объединения....16
Тип функции....18
Указатель....19
Квалификация типов....22
Представления типов (самостоятельно)....24
Общие соображения....24
Представление целых типов....26
Совместимость типов....28
Выравнивание объектов....29
Преобразования типов (самостоятельно)....31
Арифметические операнды....32
Логические величины, символы и целые числа....32
Булев тип....33
Целые числа со знаком и без знака....33
Действительный тип с плавающей точкой и целочисленный типы....34
Действительные типы с плавающей точкой....35
Действительные и комплексные....35
Обычные арифметические преобразования....36

Взаимодействие и правила поведения на занятиях

Язык общения — русский
Фото- и видеосъемка запрещается, болтовня и прочее мычание тоже
Использование мобильных гаджетов может вызвать проблемы

prep@lsi.bas-net.by
ftp://student@lsi.bas-net.by/

Старосты групп отправляют на **prep@** со своего личного ящика сообщение, в котором указывают свой телефон и ящик, к которому имеют доступ все студенты группы. В этом же сообщении в виде вложения приводят списки своих групп (**.odt** или **.doc**). Если у группы нет ящика, его следует создать. Все студенты группы должны мониторить этот ящик.

Формат темы этого сообщения: «**010901 Фамилия И.О. Список группы**»
Формат темы для **prep@** вообще: «**010901 Фамилия И.О. Суть сообщения**»

Правила составления сообщений
- текстовый формат сообщений;

Удаляется на сервере присланное в ящик prep@ все, что:
- без темы;
- имеет тему не в формате;
- содержит html, xml и прочий мусор;
- содержит рекламу, в том числе и сигнатуры web-mail серверов;
- содержит ссылки на облака и прочие гуглопомойки вместо прямых вложений.

Типы

Значение величины, хранящейся в объекте или возвращаемой функцией, определяется типом выражения, используемого для доступа к этому объекту или возвращаемому значению.

Самым простым таким выражением является идентификатор, объявленный как объект, поскольку в объявлении идентификатора указывается его тип.

Типы подразделяются на

- *типы объектов* (типы, которые описывают объекты);
- *типы функций* (типы, которые описывают функции).

Полный и неполный типы

В различных точках в единице трансляции тип объекта может быть *неполным* (без достаточной информации для определения размера объектов этого типа) или *полным* (имеющим достаточную информацию).

Тип может быть неполным или полным во всей единице трансляции или может изменять состояния в разных точках в единице трансляции.

```
struct foo_s {
    struct foo_s *next; // здесь неполный тип struct foo_s
    struct foo_s *prev;
    int a;
};
// а теперь тип struct foo_s полон (завершен, закончен)
```

Типы в языке программирования C

- булев (логический) тип;
- стандартный целочисленный тип со знаком;
- стандартный целочисленный тип без знака;
- вещественный тип с плавающей точкой;
- комплексный тип с плавающей точкой;
- символьный тип;
- перечислимый тип;
- тип `void`;
- тип массива;
- структурный тип;
- тип объединения;
- тип функции;
- тип указателя;
- атомарный тип;

Булев (логический) тип

Объект, объявленный, как имеющий тип **`_Bool`**, хранит логические значения 0 и 1.

```
_Bool in_word = 0;
_Bool thread_in_progress = 0;
```

Стандартные целочисленные типы со знаком

Существует пять ***стандартных целочисленных типов со знаком***, обозначаемых как

```
signed char
short int          short
int
long int           long
long long int      long long
```

Также могут существовать ***расширенные*** целочисленные типы со знаком, определенные конкретной реализацией.

Стандартные и расширенные целочисленные типы со знаком совокупно называются *целочисленными типами со знаком*[1].

Объект, объявленный как имеющий тип **`signed char`**, занимает тот же объем памяти, что и «простой» объект типа **`char`**.

Объект типа «простой» **`int`** имеет ***естественный размер, предлагаемый архитектурой среды выполнения*** (достаточно большой, чтобы содержать любое значение в диапазоне от **`INT_MIN`** до **`INT_MAX`**, как определено в заголовке **`<limit.h>`**).

```
short int si;
long long lli;
```

1 Ключевые слова, определенные реализацией, должны иметь форму идентификатора, зарезервированного для любого использования, как описано в 7.1.3. Зарезервированные идентификаторы

Стандартные целочисленные типы без знака

Для каждого из целочисленных типов со знаком существует соответствующий (но другой) целочисленный тип без знака (обозначаемый ключевым словом **unsigned**), который использует тот же объем памяти (включая информацию о знаке) и имеет те же требования к выравниванию.

```
unsigned char
unsigned short int          unsigned short
unsigned int                unsigned
unsigned long int           unsigned long
unsigned long long int      unsigned long long
```

Тип **_Bool** и целочисленные типы без знака, которые соответствуют стандартным целочисленным типам со знаком, являются *стандартными целочисленными типами без знака (беззнаковыми типами)*.

Целочисленные типы без знака, которые соответствуют расширенным целочисленным типам со знаком, являются *расширенными целочисленными типами без знака*.

Стандартные и расширенные целочисленные типы без знака совокупно называются *целочисленными типами без знака*.

Стандартные целочисленные типы со знаком и стандартные целочисленные типы без знака совокупно называются *стандартными целочисленными типами*; расширенные целочисленные типы со знаком и расширенные целые типы без знака совокупно называются *расширенными целочисленными типами*.

Целочисленный ранг преобразования типов

Каждый целочисленный тип имеет *ранг целочисленного преобразования*, определяемый следующим образом:

- никакие два из разных целочисленных типов со знаком не могут быть одинакового ранга, даже если они имеют одинаковое представление;
- ранг целочисленного типа со знаком больше ранга целочисленного типа со знаком меньшей точности;
- ранг **`long long int`** больше ранга **`long int`**, который больше ранга **`int`**, который больше ранга **`short int`**, который больше ранга **`signed char`**;
- ранг любого целого типа без знака равняется рангу соответствующего целого типа со знаком, если таковой имеется;
- ранг любого стандартного целочисленного типа больше, чем ранг любого расширенного целочисленного типа той же ширины;
- ранг **`char`** равняется рангу как **`signed char`**, так и рангу **`unsigned char`**;
- ранг **`_Bool`** меньше, чем ранг любого из стандартных целочисленных типов;
- ранг любого перечислимого типа равняется рангу совместимого целочисленного типа.

Ранги для знаковых целочисленных типов, например, находятся в след. отношениях:

```
signed char < short int < int < long int < long long int
```

Диапазон неотрицательных значений *целочисленного типа со знаком* является поддиапазоном соответствующего *целочисленного типа без знака*, и представление одного и того же значения в каждом типе одинаково (старший бит в обоих случаях равен 0).

Вычисления с участием беззнаковых операндов никогда не могут переполниться, поскольку результат, который не может быть представлен результирующим целочисленным типом без знака, уменьшается по модулю на <type>_MAX + 1 (модулярная арифметика).

Вещественный тип с плавающей точкой

Существует три *вещественных типа с плавающей точкой* (*real floating types*), обозначаемых как

```
float
double
long double
```

Набор значений типа **`float`** является подмножеством набора значений типа **`double`**. Набор значений типа **`double`** является подмножеством набора значений типа **`long double`**.

```
float < double < long double
```

Комплексный тип с плавающей точкой

Существует три *комплексных типа* (*complex types*), обозначаемых как

```
float _Complex
double _Complex          // gcc: _Complex, но не ISO
long double _Complex
```

Комплексные типы являются условным функционалом, поддержка которого реализациями не является необходимостью

Действительные и комплексные типы с плавающей точкой совокупно называются *типами с плавающей точкой*.

Для каждого типа с плавающей точкой существует соответствующий действительный тип, который всегда является действительным типом с плавающей точкой.

Для действительных типов с плавающей точкой это тот же самый тип.

Для комплексных типов это тип, заданный удалением ключевого слова **`_Complex`** из имени типа.

Каждый комплексный тип имеет те же требования к представлению и выравниванию, что и тип массива, содержащий ровно два элемента соответствующего действительного типа, где первый элемент равен действительной части, а второй элемент — мнимой части комплексного числа.

Символьный тип

Три типа

```
char
signed char
unsigned char
```

вместе называются *типами символов (символьными типами)*.

Тип **char** всегда определен таким образом, чтобы он имел тот же диапазон, представление и поведение, какие имеют либо **signed char** либо **unsigned char**.

Объект, объявленный, как имеющий тип **char**, может хранить любой член базового набора символов среды исполнения.

Если элемент базового набора символов среды исполнения хранится в объекте **char**, его значение гарантированно будет неотрицательным.

Если какой-либо другой символ хранится в объекте **char***, результирующее значение определяется реализацией, но должно находиться в диапазоне значений, которые могут быть представлены в этом типе.*

Тип **char**, целочисленные типы со знаком и без знака, а также типы с плавающей точкой совокупно называются *базовыми типами*.

Базовые типы являются завершенными (полными) типами объектов.

Перечисление

Перечисление содержит (включает в состав) набор именованных целочисленных константных значений.

Каждое отдельное перечисление образует отличный от других *перечислимый тип*.

```
enum color {
    red,
    green,
    blue
};

enum car {
    honda,
    toyota
};
```

Тип **char**, целочисленные типы со знаком и без знака, а также перечисляемые типы совокупно называются *целочисленными типами*.

Целочисленные и вещественные типы с плавающей точкой совокупно называются *действительными типами*.

Домены типа

Целочисленные и типы с плавающей точкой вместе называются *арифметическими типами*. Каждый арифметический тип принадлежит одному домену типа — *домен действительного типа* содержит действительные типы, *домен комплексного типа* содержит комплексные типы.

Тип void

Тип **`void`** содержит пустой набор значений.
Это неполный тип объектов, который никогда не может быть завершенным.

Массив

Тип массива или просто массив (array type) описывает непустой набор расположенных смежно объектов-членов конкретного типа, который называется *типом элемента массива*.

```
int array[];
int matrix[4][4];
struct foo_s *foo_table[TABLE_SIZE];
```

Всякий раз, когда указывается тип массива, тип его элементов должен быть полностью определен.

Типы массивов характеризуются типом элементов и количеством элементов в массиве.

Тип массива, как говорят, является производным от типа его элемента, и если тип его элемента равен **T**, тип массива иногда называют «*массивом* **T**».

Конструирование типа массива из типа элемента называется «*образованием типа массива*».

Структурный тип

Структурный тип описывает **последовательно распределенный** непустой набор объектов-членов, каждый из которых имеет *необязательно* указанное имя и, возможно, свой отдельный тип.

```
struct foo_s {                  // элемент двунаправленного списка
    struct foo_s *next;         // указатель на объект своего типа
    struct foo_s *prev;         // указатель на объект своего типа
    int           object_type; // тип объекта
    void         *object_p;     // указатель на хранимый в списке объект
};

struct bar_s {
    struct {                    // неименованная структура
        int       iv;
        const int civ;
    } s1;
    struct {
        int iv1;
        int iv2;
        int iv3;
    } s2;
};
```

Тип объединения

Тип объединения описывает **перекрывающийся** непустой набор объектов-членов, каждый из которых имеет *необязательно* указанное имя и, возможно, отдельный тип.

```
union tag_u {
    int count;
    int type_code;
}
```

Пример

Следующее является действительным фрагментом:

```
union {
    struct {
        int    alltypes;
    } n;
    struct {
        int    type;
        int    intnode;
    } ni;
    struct {
        int    type;
        double doublenode;
    } nf;
} u;

u.nf.type       = 1;
u.nf.doublenode = 3.14;
/* ... */
if(u.n.alltypes == 1)
if(sin(u.nf.doublenode) == 0.0)
/* ... */
```

Тип функции

Тип функции описывает функцию с указанным типом возврата.

Тип функции *характеризуется* типом возвращаемого значения, а также количеством и типами ее параметров.

Тип функции *является* производным от ее возвращаемого типа, и если ее возвращаемым типом является **T**, тип функции иногда называется «*функцией, возвращающей* **T**».

```
int main(int argc, char *argv[], char *envp[]) {
    ....
    return 0;
}

int main(void) { // Может и не иметь параметров
    ....
    return 0;
}
```

```
int bobik(int     idx,       // индекс
          char   *nickname, // имя
          double  height,   // рост
          double  weight)   // вес
{
    /* .... */
}
```

Указатель

Тип указателя может быть получен как из типа функции, так и из типа объекта.

Данные типы называются *ссылочным типом* (*referenced type*).

Тип указателя описывает объект, значение которого предоставляет ссылку на объект ссылочного типа.

Тип указателя, полученный из ссылочного типа **T**, иногда называют «*указателем на* **T**».

Конструирование типа указателя из ссылочного типа называется «*образованием типа указателя*».

Тип указателя является полным типом объекта.

Объявления и определения указателей.

```
int   *num_p;          // указатель на строку символов
char  *name = "ascii"; // указатель на строку символов
char  *argv[];         // массив указателей на строки
int  (*foo)();         // указатель на функцию, возвращающую int
```

Атомарный тип описывает тип, обозначенный конструкцией **`_Atomic (`***type-name***`)`**.

Атомарные типы являются необязательной особенностью, которую различные компиляторы могут и не поддерживать.

Все вышеприведенные методы конструирования производных типов могут применяться рекурсивно.

Арифметические типы и типы указателей совокупно называются ***скалярными типами***.

Тип массива и структурный тип совокупно называются ***агрегатными типами***.

Тип массива неизвестного размера является незавершенным типом. Типа массива завершается, когда в последующем объявлении (с внутренним или внешним типом связывания) для идентификатора этого типа указывается размер.

Тип структуры или тип объединения неизвестного содержимого является незавершенным типом.

Тип структуры или тип объединения завершается при объявлении структуры или объединения с тем же тегом позже, где определяется их содержимое в той же самой области видимости.

Если тип не является неполным, а также не является типом массива переменной длины, он имеет *неизменяемый (константный) размер*.

Тип характеризуется своей *категорией типов*, которая является либо самым внешним производным из производных типов, либо самим типом, если тип не состоит из производных типов.

Квалификация типов

Любой тип, упомянутый до сих пор, является *неквалифицированным типом (unqualified type)*.

Каждый неквалифицированный тип имеет несколько квалифицированных версий своего типа, соответствующих комбинациям одного, двух или всех трех квалификаторов **`const`**, **`volatile`** и **`restrict`**.

Квалифицированные или неквалифицированные версии типа — это отдельные типы, принадлежащие к одной категории типов и имеющие одинаковые требования к представлению и выравниванию[2].

Производный тип не квалифицируется квалификаторами (если таковые имеются) типа, из которого он получен — квалификатор в объявлении производного типа, например, массива, относится к типу элемента, а не к массиву.

Существует квалификатор **`_Atomic`**. Наличие квалификатора **`_Atomic`** обозначает атомарный тип.

Размер, представление и выравнивание атомарного типа не обязательно совпадает с размерами соответствующего неквалифицированного типа.

Указатель на **`void`** имеет те же требования к представлению и выравниванию, что и указатель на символьный тип.

Аналогично, указатели на квалифицированные или неквалифицированные версии совместимых типов имеют одинаковые требования к представлению и выравниванию.

Все указатели на типы структур и на типы объединений имеют одинаковые требования к представлению и выравниванию. Указатели на другие типы могут не иметь одинаковых требований к представлению или выравниванию.

---

2 Одни и те же требования к представлению и выравниванию подразумевают взаимозаменяемость в качестве аргументов функций, возвращаемых значений функций и членов объединений

ПРИМЕР 1

Тип, обозначенный как «**float ***», имеет тип «указатель на **float**».

Его категория типа — указатель, а не плавающий тип.

Версия этого типа (указателя) с квалификатором **const** обозначается как «**float * const**», тогда как тип, обозначенный как «**const float ***», не является квалифицированным типом — его тип является «указателем на const-квалифицированный **float**» и является указателем на квалифицированный тип.

Следует различать:

квалифицированный указатель на тип — **T * const** **(T * const ID)**

и

указатель на квалифицированный тип — **const T *** **(const T *ID)**

ПРИМЕР 2

Тип, обозначенный как «**struct tag (*[5])(float)**» имеет тип «массив указателей на функцию, возвращающую **struct tag**».

Массив имеет размерность пять, а функция имеет один параметр типа **float**.

Его категория типов — массив.

Представления типов (самостоятельно)

Общие соображения

Представления всех типов **точно не определяются**, кроме случаев, указанных в этом подпункте.

1) За исключением битовых полей, объекты состоят из смежных последовательностей из одного или нескольких байтов, количество, порядок и кодирование которых либо явно указаны, либо определяются реализацией.

2) Значения, хранящиеся в битовых полях без знака, и объекты типа **`unsigned char`** представляются с использованием чисто двоичной записи.

3) Значения, хранящиеся в объектах без битовых полей любого другого типа объектов, состоят из n x **`CHAR_BIT`** бит, где n — размер объекта этого типа, в байтах.

4) Значение может быть скопировано в объект типа **`unsigned char[n]`** (например, с помощью **`memcpy()`**); результирующий набор байтов называется *объектным представлением значения*.

5) Значения, хранящиеся в битовых полях, состоят из **m** битов, где **m**-размер, указанный для битового поля. Объектное представление представляет собой набор из **m** битов, которые битовое поле содержит в адресуемом запоминающем устройстве, удерживающем его.

6) Два значения (кроме **`NaNs`**) с одним и тем же объектным представлением при сравнении считаются равными, однако значения, которые при сравнении считаются равными, могут иметь разные объектные представления.

7) Некоторые объектные представления могут не представлять значения объектного типа.

Если хранимое значение объекта имеет такое представление и читается lvalue-выражением, которое имеет не символьный тип, поведение не определено.

8) Если такое представление создается побочным эффектом, который изменяет весь или любую часть объекта lvalue-выражением, которое не имеет символьного типа, поведение не определено. Такое представление называется *представлением-ловушкой*[3].

9) Когда значение хранится в объекте типа структуры или типа объединения, в том числе в объекте-члене, байты объектного представления, соответствующие любым байтам-заполнителям (padding), могут принимать неопределенные значения. Такое случается потому, что при присвоении структуры байты-заполнители могут не копироваться.

Когда значение сохраняется в члене объекта типа *объединения*, байты объектного представления, которые не соответствуют этому члену, но соответствуют другим членам, принимают неопределенные значения.

Если к значению, имеющему более одного объектного представления применяется оператор, то используемое объектное представление не влияет на значение результата.

10) Объекты **x** и **y** с одинаковым эффективным типом **T** могут иметь одинаковое значение при обращении к ним как к объектам типа **T**, но иметь различные значения в другом контексте. В частности, если для типа **T** определен оператор **==**, то **x == y** не означает, что **memcmp(&x, &y, sizeof(T)) == 0**.

Более того, **x == y** не обязательно означает, что **x** и **y** имеют одинаковое значение; другие операции над значениями типа **T** могут их отличать.

---

3 См. Trap representation

Представление целых типов

Для целочисленных типов без знака, отличных от **`unsigned char`**, биты представления объекта делятся на две группы — биты значений (значащие биты) и биты-заполнители. Если имеется **N** значащих битов, каждый бит представляет собой различную степень двойки между 1 и 2^{N-1} , таким образом объекты этого типа способны представлять значения от 0 до 2^{N-1}, используя обычное двоичное представление.

Значения любых битов-заполнителей не конкретизируются.

Для целочисленных типов со знаком биты представления объекта делятся на три группы

- биты значения;
- биты-заполнители;
- бит знака.

Наличие битов-заполнителей не является обязательным — их может и не быть.

Всегда существует ровно один знаковый бит.

Каждый значащий бит имеет такое же значение, что и тот же бит в объектном представлении соответствующего беззнакового типа (если в знаковом типе есть **M** битов значения и **N** в беззнаковом типе, то **M ⩽ N**). Если знаковый бит равен нулю, он не влияет на результирующее значение.

Если знаковый бит равен единице, значение представляется одним из следующих способов:

- соответствующее значение со знаковым битом, равным 0 становится отрицательным (как по знаку, так и по величине);
- **дополнение до двух — дополнительный код;**
- дополнение до единицы — обратный код.

Что из этого применимо, определяется реализацией, как и то, является ли значение с знаковым битом 1 и всеми битами значения, установленными в ноль (для первых двух вариантов) или со знаковым битом и всеми битами, установленными в значение 1 (для дополнения до единицы) представлением-ловушкой или нормальным значением.

В случае дополнения до единицы, если представление со всеми единицами, установленными в 1, является нормальным значением, оно называется отрицательным нулем.

Значения бит-заполнителей не регламентируются.

Допустимое (не являющееся ловушкой) представление объекта целочисленного типа со знаком, где знаковый бит равен нулю, является допустимым представлением объекта соответствующего типа без знака и представляет то же значение.

Для любого целочисленного типа объектное представление, где все биты равны нулю, является представлением нулевого значения в этом типе.

Точность целочисленного типа — это количество битов, которые он использует для представления значений, исключая любые знаки и биты заполнения.

Ширина целочисленного типа остается неизменной, но включает в себя любой знаковый бит; таким образом, для целочисленных типов без знака точность и ширина одинаковы, в то время как для целочисленных типов со знаком ширина на единицу больше точности.

Совместимость типов

Два типа имеют *совместимый тип*, если их типы одинаковы.

Два типа структуры, объединения или перечисления, объявленные в отдельных файлах, совместимы, если их теги и члены удовлетворяют следующим требованиям:

Если один объявлен с тегом, другой должен быть объявлен с тем же тегом.

Если оба являются завершенными типами где-либо в пределах их соответствующих единиц трансляции, то применяются следующие дополнительные требования:

- между их членами должно быть взаимно-однозначное соответствие, так что каждая пара соответствующих членов объявляется с совместимыми типами;
- если один член пары объявлен со спецификатором выравнивания, другой объявлен с эквивалентным спецификатором выравнивания;
- если один член пары объявлен с именем, другой объявлен с тем же именем.

Для структур соответствующие члены должны быть объявлены в одинаковом порядке.

Для структур или объединений соответствующие битовые поля должны иметь одинаковую ширину.

Для перечислений соответствующие члены должны иметь одинаковые значения.

Все объявления, которые ссылаются на один и тот же объект или функцию, должны иметь совместимый тип, в противном случае поведение не определено.

Выравнивание объектов

Завершенные (полные) типы объектов имеют требования к выравниванию, которые накладывают ограничения на адреса, по которым могут размещаться объекты данного типа. Выравнивание — это определенное реализацией целочисленное значение, представляющее собой количество байт между последовательными адресами, по которым может быть размещен данный объект. Тип объекта накладывает требование выравнивания на каждый объект данного типа. Более строгое выравнивание может быть запрошено с помощью ключевого слова **`_Alignas`**.

Фундаментальное выравнивание представлено выравниванием, меньшим или равным наибольшему выравниванию, поддерживаемому реализацией во всех контекстах, которое равно **`_Alignof (max_align_t)`**.

Расширенное выравнивание представлено выравниванием, большим, чем **`_Alignof (max_align_t)`**. Определяется реализацией, поддерживаются ли какие-либо расширенные выравнивания и контексты, в которых они поддерживаются.

Выравнивания представляются в виде значений типа **`size_t`**. Допустимые выравнивания включают только те значения, которые возвращаются выражением **`_Alignof`** для фундаментальных типов, плюс дополнительный набор значений, определенный реализацией, который может быть пустым.

Каждое действительное значение выравнивания является неотрицательной целочисленной степенью двойки.

Выравнивания имеют порядок от более слабого к более сильному или более строгому выравниванию. Более строгие выравнивания имеют большие значения выравнивания. Адрес, который удовлетворяет требованию выравнивания, также удовлетворяет любому более слабому действительному требованию выравнивания.

Требование к выравниванию полного типа можно запросить с помощью выражения **`_Alignof`**. Типы **`char`**, **`sign char`** и **`unsigned char`** имеют самые слабые требования к выравниванию.

Сравнение выравниваний имеет смысл и дает очевидные результаты:

- два выравнивания равны, если равны их числовые значения;
- два выравнивания отличаются, если их числовые значения не равны.
- в том случае, когда одно выравнивание больше другого, оно представляет более строгое выравнивание.

Преобразования типов (самостоятельно)

Некоторые операторы автоматически преобразуют значения операндов из одного типа в другой — выполняют *неявные преобразования*.

В данном разделе указываются результаты таких неявных преобразования, а также результаты, получаемые в результате операции *приведения* (cast operation) или *явного преобразования* (explicit conversion).

За исключением отдельных случаев, преобразование значения операнда в совместимый тип не вызывает изменений в значении или представлении.

Арифметические операнды

Логические величины, символы и целые числа

Следующее может использоваться в выражении везде, где могут использоваться **int** или **unsigned int**:

- объект или выражение целочисленного типа (кроме **int** или **unsigned int**), чей ранг целочисленного преобразования меньше или равен рангу **int** и **unsigned int**.

- битовое поле типов **_Bool**, **int**, **sign int** или **unsigned int**.

Если **int** может представлять все значения исходного типа (что для битового поля ограничено шириной), значение преобразуется в **int**; в противном случае оно конвертируется в **unsigned int**. Эти преобразования называются целочисленными распространениями (повышениями, продвижениями) типа.

Целочисленные повышения применяются только: как часть обычных арифметических преобразований к определенным выражениям аргументов, к операндам унарных операторов +, - и ~, а также к обоим операндам операторов сдвига. Все остальные типы не меняются при целочисленных повышениях типа.

Целочисленные повышения типа сохраняют значение, включая знак.

Булев тип

При преобразовании любого скалярное значения в **`_Bool`**, результат равен 0, если значение равно 0; в противном случае результат будет равен 1.

Целые числа со знаком и без знака

Если при преобразовании значения одного целочисленного типа в другой целочисленный тип, отличный от **`_Bool`**, это значение может быть представлено новым типом, оно не изменяется.

В противном случае, если новый тип является беззнаковым, значение преобразуется путем многократного сложения или вычитания числа на единицу большего, чем максимальное значение, которое может быть представлено в новом типе, до тех пор, пока значение не окажется в диапазоне нового типа.

В противном случае, если новый тип является знаковым, и значение не может быть представлено в нем, результат определяется реализацией, либо инициируется *сигнал*.

При преобразовании любого скалярное значения в **`_Bool`**, результат равен 0, если значение равно 0; в противном случае результат будет равен 1.

Действительный тип с плавающей точкой и целочисленный типы

Если конечное значение действительного типа с плавающей точкой преобразуется в целочисленный тип, отличный от **`_Bool`**, дробная часть отбрасывается (т.е. значение усекается по направлению к нулю). Если значение целой части не может быть представлено целочисленным типом, поведение не определено.

Если значение целочисленного типа преобразуется в действительный тип с плавающей точкой, и если преобразуемое значение может быть точно представлено в новом типе, оно не изменяется.

Если преобразуемое значение находится в диапазоне значений, которые могут быть представлены, но не могут быть представлены точно, результатом является либо ближайшее более высокое, либо ближайшее нижнее представимое значение, в зависимости от реализации

Если конвертируемое значение находится вне диапазона значений, которые могут быть представлены, поведение не определено. Результаты некоторых неявных преобразований могут быть представлены в большем диапазоне и с большей точностью, чем требуется для нового типа.

Действительные типы с плавающей точкой

Когда значение действительного типа с плавающей точкой преобразуется в действительный тип с плавающей точкой, и если преобразуемое значение может быть точно представлено в новом типе, оно не изменяется.

Если преобразуемое значение находится в диапазоне значений, которые могут быть представлены, но не могут быть представлены точно, результатом является либо ближайшее более высокое, либо ближайшее нижнее представимое значение, выбранное способом, в зависимости от реализации.

Если преобразуемое значение находится вне диапазона значений, которые могут быть представлены, поведение не определено.

Результаты некоторых неявных преобразований могут быть представлены в большем диапазоне и точности, чем это требуется для нового типа.

Действительные и комплексные

Когда значение действительного типа преобразуется в комплексный тип, действительная часть значения комплексного результата определяется по правилам преобразования в соответствующий действительный тип, а мнимая часть значения комплексного результата представляет собой положительный или беззнаковый ноль.

Если значение комплексного типа преобразуется в значение действительного типа, мнимая часть комплексного значения отбрасывается, а значение действительной части преобразуется в соответствии с правилами преобразования для соответствующего действительного типа.

Обычные арифметические преобразования

Многие операторы, ожидающие операнды арифметического типа, производят преобразования и выдают типы результатов аналогичным образом. Целью является определить общий действительный тип для операндов и результата. Каждый из указанных операндов преобразуется без изменения типа домена в тип, соответствующий действительный тип которого является общим действительным типом для обоих. Если явно не указано иное, общий действительный тип также является соответствующим действительным типом результата, чей домен типов является областью типов операндов, если они одинаковы, и сложной в противном случае. Такое поведение называется обычным арифметическим преобразованием.

Если соответствующий действительный тип одного из операндов является **`long double`**, другой операнд преобразуется, без изменения домена типа, в тип, соответствующий действительный тип которого является **`long double`**.

В противном случае, если соответствующий действительный тип одного из операндов является **`double`**, другой операнд преобразуется без изменения домена типа в тип, соответствующий действительный тип которого является **`double`**.

В противном случае, если соответствующий действительный тип одного из операндов является **`float`**, другой операнд преобразуется, без изменения домена типа, в соответствующий действительный тип которого является **`float`**. Например, сложение **`double _Complex`** и **`float`** влечет за собой только преобразование операнда типа **`float`** в операнд типа **`double`** (и приводит в результате к **`double _Complex`**).

В противном случае для обоих операндов выполняются целочисленные преобразования. Затем к полученным операндам применяются следующие правила:

- если оба операнда имеют один и тот же тип, то дальнейшее преобразование не требуется.

- в противном случае, если оба операнда имеют целочисленные типы со знаком или оба имеют целочисленные типы без знака, операнд с типом меньшего ранга преобразования преобразуется в тип операнда с большим рангом.

- в противном случае, если операнд целочисленного типа без знака имеет ранг, больший или равный рангу типа другого операнда, тогда операнд целочисленного типа со знаком преобразуется в тип операнда целочисленного типа без знака.

- в противном случае, если тип операнда целочисленного типа со знаком может представлять все значения типа операнда целочисленного типа без знака, тогда операнд целочисленного типа без знака преобразуется в тип операнда целочисленного типа со знаком.

- в противном случае оба операнда преобразуются в целочисленный тип без знака, соответствующий типу операнда целочисленного типа со знаком.

Значения операндов с плавающей точкой и результатов выражений с плавающей точкой могут быть представлены в большем диапазоне и точности, чем требуется типом. Таким образом, типы не изменяются. Для удаления дополнительного диапазона и точности, тем не менее, все еще необходимы операторы приведения (cast) и присваивания.