Типы данных

А. Г. Фенстер, fenster@fenster.name

20 марта 2010 г.

На этом занятии мы поговорим про различные типы данных в языке C.

1 Представление чисел в компьютере

Вспомним, как компьютер хранит целые и вещественные числа. Для хранения чисел используется двоичная система. Целые беззнаковые числа хранятся очевидным образом, целые числа со знаком представлены в дополнительном коде: битовое представление числа $(-k)$ совпадает с битовым представлением беззнакового числа $2^N - k$, где N — количество двоичных разрядов (бит) в данном типе. Такое представление выбрано в основном по той причине, что добавление отрицательных чисел не изменяет арифметические операции (в частности, автоматически выполняется $k + (-k) = 0$).

Вещественные числа в компьютере хранятся в формате *с плавающей запятой* (по-английски floating point, «плавающая точка», поскольку у них принято использовать для записи дробей точку, а не запятую). Число хранится в виде $\pm s \cdot 2^e$, где s — *мантисса* (в английском языке принят термин significand), e — *порядок* (exponent). Представление обычно нормализовано ($1 \leqslant s < 2$).

Рассмотрим соответствие типов языка C и перечисленных моделей представления чисел, а также разберём основные недостатки каждого из способов представления.

2 Целые числа

Размер типа `char` — 1 байт (`sizeof(char) == 1`). Заметим, что стандарты определяют байт как адресуемый элемент, достаточный для хранения одного символа, состоящий не менее чем из восьми бит (константа `CHAR_BIT` из `limits.h` даёт точное значение для данной реализации). Размеры остальных типов стандартами в явном виде не определяются, определены лишь соотношения между ними, а `limits.h` определяет пределы для данной реализации: `INT_MAX`, `INT_MIN`, `UINT_MAX` и прочие.

К стандартным знаковым целым типам относятся `signed char`, `short int`, `int`, `long int` и `long long int`. К стандартным беззнаковым целым типам относятся те же типы с добавленным словом `unsigned` (`unsigned char`, `unsigned short int` и т. д.).

Что касается типа `char` — реализация вправе выбирать, будет он знаковым или беззнаковым. В частности, у `gcc` это может быть задано ключом при компиляции.

Размер беззнакового типа совпадает с размером соответствующего знакового типа.

Стандарт языка C не указывает, каким из трёх основных способов (прямой код, обратный код, дополнительный код) представлены отрицательные числа. В процессорах x86 используется дополнительный код (two's complement), мы будем приводить примеры именно для такого способа хранения.

Из определения дополнительного кода следует, что соответствие между беззнаковыми и знаковыми типами такое:

`unsigned char`	0	1	2	...	127	128	129	...	253	254	255
`signed char`	0	1	2	...	127	−128	−127	...	−3	−2	−1

Аналогично соответствуют друг другу беззнаковый и знаковый `int` и прочие целые типы.

В языке C не происходит ошибки в случае выполнения переноса единицы в несуществующий старший бит в случае беззнакового типа и при переполнении и изменении знакового разряда знакового типа.

Для целых чисел определены битовые операции `&`, `|`, `^` (исключающее «или»),`~` и битовые сдвиги `<<`, `>>`. Важно понимать, что `&`, `|`, `~` отличаются от логических операций `&&`, `||` и `!`: например, `7 & 8 == 0`, но `7 && 8 == 1`.

Битовые операции можно использовать для установки нуля или еди-

ницы в определённый бит числа. Например, для установки 1 в бит k числа a можно использовать

```
a |= (1 << k);
```

для установки туда нуля — похожую операцию

```
a &= ~(1 << k);
```

а для проверки значения k-го бита числа a можно сравнить с нулём число

```
(a >> k) & 1
```

Всегда берите операции сдвига в скобки! Приоритет операции сдвига меньше, чем операции сложения, поэтому `a << b + c` соответствует `a << (b + c)`, что в большинстве случаев не является тем, что вы ожидаете.

Битовые операции также используются для установки и снятия флагов — констант, обычно равных степеням двойки: 1, 2, 4, 8, ... Это даёт возможность хранить в одной переменной типа `unsigned int` значения 32 различных флагов. Например, в условиях тотальной нехватки памяти в 32 битах можно сохранить информацию о 32 сданных (или несданных) экзаменах студента:

```
#define IS_CALCULUS_PASSED    1
#define IS_ALGEBRA_PASSED     2
#define IS_PROGRAMMING_PASSED 4
...

int student_result = 0;

/* сдали программирование */
student_result |= IS_PROGRAMMING_PASSED;

/* сдали ли алгебру? */
if (student_result & IS_ALGEBRA_PASSED)
    ...
```

3 Вещественные числа

Стандарт не определяет точного способа хранения вещественных чисел. Наиболее часто употребляется формат чисел с плавающей запятой, описанный в стандарте IEEE 754.

При вычислении выражений с вещественными числами возможны потери точности, о которых вам подробно рассказывали на лекциях по представлению данных. В качестве примера заметим, что цикл

```
float a = 1.0;
while (a != 0.0)
    a /= 2.0;
```

сделает вполне определённое конечное число итераций, а цикл

```
float a;
for (a = 0.0; a != 1.0; a += 0.1)
    ;
```

будет бесконечным.

4 Структуры и объединения

Перечислю основные факты о структурах, которые могут вызвать затруднение. Во-первых, имя структуры не является именем типа в C: объявив `struct s`, можно завести переменную типа `struct s`, но не типа `s` (в отличие от C++). Вполне допустимы анонимные структуры. В следующем примере две структуры по сути одинаковы, но присваивание между ними невозможно:

```
struct
{
    int a, b;
} a;

struct s
{
    int a, b;
} b;
```

Как видно из предыдущего примера, при определении структуры можно сразу определить переменную такого типа.

При хранении структур важно понимать, что размер структуры может быть больше, чем суммарный размер её элементов: элементы структуры выравниваются по определённым правилам. Например, структура

```
struct
{
    int a;
    char c;
    int b;
};
```

обычно будет занимать 12 байт (между tt char c и `int b` будут вставлены три дополнительных байта выравнивания). Байты выравнивания могут быть также добавлены в конец структуры, но не в начало: адрес первого элемента структуры всегда совпадает с адресом самой структуры.

Различные способы выравнивания могут привести к проблемам при чтении структур, ранее записанных в файл на компьютере с другой ОС или архитектурой.

Для обращения к элементу структуры используется точка, для обращения к элементу структуры через адрес структуры используется «стрелочка» `->`. Вообще говоря, в C запись `p->a` является более краткой записью для `(*p)->a` (скобки важны: это не то же самое, что `*p->a`).

Существует также особый вид элементов структуры: битовые поля, для которых указывается точное количество бит, выделенных под это поле. Например, такая структура точно соответствует типу `float` в стандарте IEEE 754:

```
struct float
{
    unsigned int significand: 23;
    unsigned int exponent: 8;
    unsigned int sign: 1;
};
```

Битовые поля позволяют реализовать флаги без объявления большого количества макросов (`#define`). Пример про экзамены может быть переписан так:

```
struct exam_list
{
    unsigned int IS_CALCULUS_PASSED: 1;
    unsigned int IS_ALGEBRA_PASSED: 1;
    unsigned int IS_PROGRAMMING_PASSED: 1;
    ...
};

struct exam_list student_result;
student_result.IS_PROGRAMMING_PASSED = 1;
```

Все члены объединения `union` размещаются в памяти, начиная с одного и того же адреса. Объединения позволяют экономить память, если в одних и тех же переменных может понадобиться хранить значения разных типов, и разбирать внутреннюю структуру типов. Например, можно записать

```
union Float
{
    struct float sf;
    float f;
} u;
```

и, присвоив для примера `u.f = 1.0`, смотреть на `u.sf.significand`, `u.sf.exponent` и `u.sf.sign`. Желающие могут таким образом изучить, как *на самом деле* представляются вещественные числа в стандарте IEEE 754.

5 Перечисления

Перечисления `enum` используются, если нужно создать тип, состоящий из заранее определённых константных значений:

```
enum suite { SPADE, HEART, DIAMOND, CLUB };
```

Значениям можно также указывать соответствующие числовые значения. В языке C константы перечисления имеют глобальную область видимости, поэтому использование их не очень удобно (эта проблема решена в C++ введением пространств имён).