bennyhuo · 更新于 2021-01-05

上一节 【加餐】背景知识 3-2 数据类型之... 下一节

数据类型之 int 类型

既然要说问内存，那总得知道内存当中的数据时怎么存储的吧？简单来说，内存当中的每一个存储单元都是一个逻辑电路，通过通断来表示二进制的 0 或者 1，这也就是我们熟知的 1 比特，或者 bit，不过这个单位太小了，所以用 8 个比特来表示一个更大的单位，叫做字节，也就是 Byte。这个单位的命名也是颇具戏剧性，英文当中 bit 是一小口，Byte 是一大口，懂我意思吧？

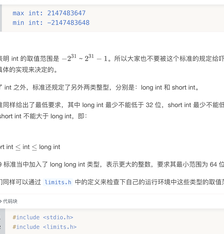
这里的比特也通常被称作位，因此我们常说的 32位、64位计算机，其实说的是 CPU 每次最大读取的数据长度，这个也被叫做字长，所以 32 位的 CPU 每次最长可以处理的数据时 4 个字节，64位的则是 8 个字节。

数据以二进制的形式交给计算机去处理非常自然，不过对于程序语言来说就不是特别的理想了。因为我们的程序语言是给人看的，人看见二进制估计要疯掉，我们需要的是整数、小数这样的表示法，所以 C 语言当中定义了几种基本类型，用来表示数值，我们依次来见识一下他们。

有符号整型

int 是 integer 的缩写，就是整数的意思，我们觉得整数嘛，可以很大很大，也可以很小很小，但计算机的存储是要钱的，每一位都是要钱的，不能由着性子来，因此整型数据实际上是有表示范围的。C 标准规

定 int 类型的范围最少是由两个字节即 16 位来表示，也就是说 int 类型的表示范围至少是 -2^{15} ~ $2^{15}-1$。



哇，这也太小了吧，微信运动的步数要是用 int 来存岂不是分分钟就溢出了？

这里请大家注意，这是 C 语言标准规定的范围，一方面 C 语言标准历史悠久，当时的计算机 16 位的 CPU 还是主流，另一方面 C 语言标准只是规定了下限，因此现在的 C 语言实现很多都远远大于这个范围，我们可以通过引入 `limit.h` 这个头文件来获取对应机器的整型值范围：

代码块

```c
#include <stdio.h>
#include <limits.h>

int main() {
  printf("max int: %d\n", INT_MAX);
  printf("min int: %d\n", INT_MIN);
  return 0;
}
```

在我的 64 位 Windows 的 MSVC 2019 环境中，程序输出的结果如下：

代码块

```
max int: 2147483647
min int: -2147483648
```

这表明 int 的取值范围是 -2^{31} ~ $2^{31}-1$，所以大家也不要被这个标准的规定给吓到，具体的范围还是由具体的实现来决定的。

除了 int 之外，标准还规定了另外两类整型，分别是：long int 和 short int。

标准同样给出了最低要求，其中 long int 最少不能低于 32 位，short int 最少不能低于 16 位，int 不能低于 short int 不能大于 long int，即：

$$\text{short int} \leqslant \text{int} \leqslant \text{long int}$$

C99 标准当中加入了 long long int 类型，表示更大的整数，要求其最小范围为 64 位。

我们同样可以通过 `limits.h` 中的定义来检查下自己的运行环境中这些类型的取值范围：

代码块

```c
#include <stdio.h>
#include <limits.h>

int main() {
  printf("max short: %d\n", SHRT_MAX);
  printf("min short: %d\n", SHRT_MIN);

  printf("max int: %d\n", INT_MAX);
  printf("min int: %d\n", INT_MIN);

  printf("max long: %ld\n", LONG_MAX);
  printf("min long: %ld\n", LONG_MIN);

  printf("max long: %lld\n", LLONG_MAX);
  printf("min long: %lld\n", LLONG_MIN);

  return 0;
}
```

打印的结果如下：

代码块

```
max short: 32767
min short: -32768
max int: 2147483647
min int: -2147483648
max long: 2147483647
min long: -2147483648
max long: 9223372036854775807
min long: -9223372036854775808
```

可以看出，我的电脑运行环境中：

$$\text{short int(16bit)} < \text{int(32bit)} = \text{long int(32bit)} < \text{long long int(64bit)}$$

以上整型类型中，short int 可以简写为 short，long int 则可以简写为 long，long long int 可以简写为 long long。

我们在打印输出整型范围的时候，用到了 printf 函数的格式化功能，运行时，`%d` 会被后面的 int 替代，`%ld` 则对应于 long int，`%lld` 则对应于 long long int，`%hd` 则对应于 short int。为什么不用 `%sd` 呢？因为 `%s` 是字符串的格式化符，因此只好使用 short 的第二个字母 h 来表示 short 了。

这里的 `d` 是 Decimal 的意思，表示十进制整型。如果要打印十六进制的整数表示，使用 `x`，八进制则使用 `o`。

整数值在内存当中的存储当然是二进制，但编写和阅读程序或者使用程序的是人，所以输入和输出是十进制是非常常见的需求，当然，对于程序员来讲，十六进制也是非常常见的需求，因为时间长了之后我们会非常习惯计算机的二进制表示法，而十六进制既能很好的反应这一本质，又能写起来非常高效，因此在实际的开发当中下面的例子也是非常常见的：

代码块

```c
#define UINT_MAX 0xffffffffU
```

大家先不要在意这个语法，只需要关注后面数字的写法。这里定义了 UINT_MAX 的值是 0xffffffffU，用的就是十六进制的表示法，只需要使用 `0x` 开头即可，相应的，我们也可以使用八进制表示整数，例如十进制的 `16` 用八进制表示是 `20`，在 C 语言当中写作 `020` 即可。

`0xffffffffU` 当中的 `U` 表明这是一个无符号整数，无符号证书我们将会在下面给大家介绍，在 C 语言源码当中我们写一个整数出来，如果范围在 int 类型的表示范围内，那么编译器会默认它为 int 类型，如果超出了 int 的范围，则会默认为 long int 类型，当然我们也可以强制让一个整数的类型为 long int，采用类似的方法在整数的后面加一个 `l` 或者 `L` 即可：

代码块

```
1l
100L
```

这两个整数都是 long int 类型，不过大家应该可以明显的看到，`1l` 看着特别迷惑人，某些字体中的 `1` 和 `l` 几乎一样，因此对于 long int 的后缀，我们强烈建议大家只使用 `L`，避免使用 `l`。

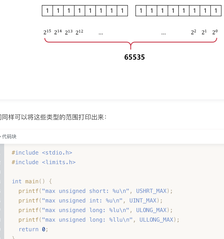
实际上现在有不少语言在设计这部分语法时已经禁止使用 `l` 来表示 long int 了，原因很简单，为了规避错误，C 语言的设计哲学是自由和相信程序员，不过实际的情况是随着编程行业的发展，程序设计日益复杂，在实践当中基于程序员的自由度越低越容易写出规范稳定且容易维护的语言，即便是自由度相对较高的语言，也纷纷在使用认为的编码规范来约束开发者。

无符号整型

无符号整型其实比较容易理解，前面提到的几类整型前面加一个 unsigned，即可构成对应的无符号类型，例如：

- unsigned short int
- unsigned int
- unsigned long int
- unsigned long long int

无符号整型的取值范围是 $0 \sim 2^n - 1$。其中 n 是位数，因此对于 16 位的 unsigned int 来说，其取值范围是 $0 \sim 2^{16} - 1$，其他的类似。



我们同样可以将这些类型的范围打印出来：

代码块

```c
#include <stdio.h>
#include <limits.h>

int main() {
  printf("max unsigned short: %u\n", USHRT_MAX);
  printf("max unsigned int: %u\n", UINT_MAX);
  printf("max unsigned long: %lu\n", ULONG_MAX);
  printf("max unsigned long: %llu\n", ULLONG_MAX);
  return 0;
}
```

输出的结果如下：

代码块

```
max short: 65535
max int: 4294967295
max long: 4294967295
max long: 18446744073709551615
```

对于无符号整型，我们在打印输出时，格式符把整型的 `d` 替换成 `u` 即可。