C/C++ 语言

C/C++ 数据

张晓平

武汉大学数学与统计学院

Table of contents

整型数据

整数的存储方式

- 数据都是以二进制的形式存储。
- 整数以补码的方式存储。

  1. 正数的补码是其本身
  2. 负数的补码：将其绝对值的二进制形式按位取反再加 1。

整数的存储方式

0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0

图 1: 正数 10 的存储方式

10 的原码

0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0

取反

1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1

加 1，得-10 的补码

1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0

图 2: 负数-10 的存储方式

int 型

int 型表示有符号整数，其取值范围依赖于系统。

- C：使用头文件 limits.h
  - 它专用于检测整型数据类型的取值范围。
- C++：使用头文件 limits
  - 它提供了 std::numeric_limits 模板类，用于替代 C 语言，检测各种基本数据类型的取值范围及其他信息。

int 型

```c
// int_info.c
#include<stdio.h>
#include<limits.h>
int main(void)
{
  printf("range of int is %d ~ %d\n", INT_MIN, INT_MAX
  );
  printf("sizeof int = %lu bytes\n", sizeof(int));
  return 0;
}
```

int 型

```c
// int_info.c
#include<stdio.h>
#include<limits.h>
int main(void)
{
  printf("range of int is %d ~ %d\n", INT_MIN, INT_MAX
  );
  printf("sizeof int = %lu bytes\n", sizeof(int));
  return 0;
}
```

```
$ gcc  int_info.c
$ ./a.out
range of int is -2147483648 ~ 2147483647
sizeof int = 4 bytes
```

int 型

```
// int_info.cpp
#include<cstdio>
#include<limits>
using namespace std;
int main()
{
  printf( "range of int is %d ~ %d\n",
          numeric_limits<int>::min(),
          numeric_limits<int>::max() );
  printf( "sizeof int = %lu bytes\n", sizeof(int) );
  return 0;
}
```

int 型

```cpp
// int_info.cpp
#include<cstdio>
#include<limits>
using namespace std;
int main()
{
  printf( "range of int is %d ~ %d\n",
          numeric_limits<int>::min(),
          numeric_limits<int>::max() );
  printf( "sizeof int = %lu bytes\n", sizeof(int) );
  return 0;
}
```

```
$ g++  int_info.cpp
$ ./a.out
range of int is -2147483648 ~ 2147483647
sizeof int = 4 bytes
```

int 变量的声明

关键字 int 用于声明基本的整型变量，书写格式为

```
int  var;
int  var1, var2;
```

要声明多个变量，

- 可以逐个声明每个变量；
- 也可在 int 后跟一个变量名列表，各个变量之间用逗号隔开。

int 变量的赋值

int 变量的赋值有如下三种方式：

1. 先声明，后赋值

```
int n;
n = 1;
```

2. 先声明，后通过 scanf 函数赋值

```
int n;
scanf("%d", &n);
```

3. 初始化变量

```
int n = 1;
```

int 变量的初始化

初始化变量就是为变量赋一个初始值。

```
int a = 1;
int b = 2, c = 3;
int d, e = 4;  // valid, but not good
```

请避免在一个声明语句中同时出现初始化和未初始化的变量。

`int` 变量的初始化

声明语句为变量创建、标定存储空间并为其指定初始值。

int 变量的初始化

声明语句为变量创建、标定存储空间并为其指定初始值。

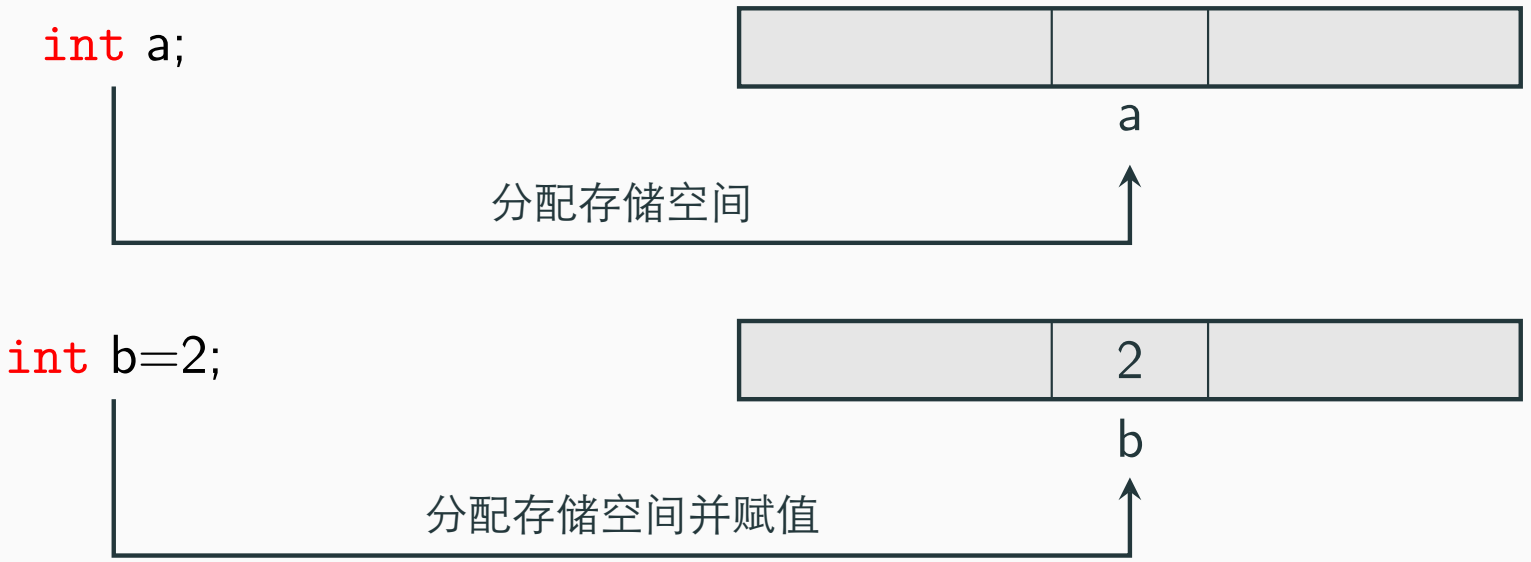

图 3: 定义和初始化变量

int 值的打印

```c
// print1.c: display some properties of printf()
#include<stdio.h>
int main(void)
{
  int a = 10;
  int b = 2;
  printf("Doing it right: ");
  printf("%d - %d = %d\n", a, b, a-b);
  printf("Doing it wrong: ");
  printf("%d - %d = %d\n", a);
  return 0;
}
```

int 值的打印

```c
// print1.c: display some properties of printf()
#include<stdio.h>
int main(void)
{
  int a = 10;
  int b = 2;
  printf("Doing it right: ");
  printf("%d - %d = %d\n", a, b, a-b);
  printf("Doing it wrong: ");
  printf("%d - %d = %d\n", a);
  return 0;
}
```

```
$ gcc print1.c
$ ./a.out
Doing it right: 10 - 2 = 8
Doing it wrong: 10 - 73832 = 771
```

int 值的打印

第二次调用 print() 时，程序使用 a 为第一个 %d 提供打印值，然后用内存中的任意值为其余两个 %d 提供打印值。

注意

使用 printf() 时，格式说明符的个数与要显示值的数目必须相同。

八进制数和十六进制数的打印

在 C 中，有专门的前缀指明进制。

- 前缀 `0x` 或 `0X` 表示十六进制数
  - 16 的十六进制表示为 `0x10` 或 `0X10`。
- 前缀 `0` 表示八进制数
  - 16 的八进制表示为 `020`。

打印八进制数和十六进制数

```c
// bases.c: print 100 in decimal, octal and hex
#include <stdio.h>
int main(void)
{
  int x = 100;
  printf("dec = %d; oct = %o; hex = %X\n", x, x,
   x);
  printf("dec = %d; oct = %#o; hex = %#X\n", x,
  x, x);
  return 0;
}
```

打印八进制数和十六进制数

```c
// bases.c: print 100 in decimal, octal and hex
#include <stdio.h>
int main(void)
{
  int x = 100;
  printf("dec = %d; oct = %o; hex = %X\n", x, x,
   x);
  printf("dec = %d; oct = %#o; hex = %#X\n", x,
  x, x);
  return 0;
}
```

```
$ gcc bases.c
$ ./a.out
dec = 100; oct = 144; hex = 64
dec = 100; oct = 0144; hex = 0x64
```

八进制数和十六进制数的打印

进制	格式说明符	格式说明符（显示前缀）
十进制	%d	
八进制	%o	%#o
十六进制	%x 或 %X	%#x 或 %#X

其他整数类型

C/C++ 提供 4 个附属关键字修饰 `int`： `short`、`long`、`unsigned` 和 `signed`。

类型	含义	占位符
`short (int)`	适用于小数值	`%hd, %ho, %hx`
`long (int)`	适用于大数值	`%ld, %lo, %lx`
`long long (int)`	适用于更大数值	`%lld, %llo, %llx`
`unsigned (int)`		`%u`
`unsigned long (int)`		`%lu`
`unsigned long long (int)`		`%llu`

其他整数类型

C/C++ 提供 4 个附属关键字修饰 `int`：`short`、`long`、`unsigned` 和 `signed`。

类型	含义	占位符
`short (int)`	适用于小数值	`%hd, %ho, %hx`
`long (int)`	适用于大数值	`%ld, %lo, %lx`
`long long (int)`	适用于更大数值	`%lld, %llo, %llx`
`unsigned (int)`		`%u`
`unsigned long (int)`		`%lu`
`unsigned long long (int)`		`%llu`

关键字 signed 可以和任何有符号类型一起使用，使数据类型更加明确。如 short、short int、signed short 和 signed short int 表示同一种类型。

其他数据类型

问题

为什么会出现多种整数类型?

其他数据类型

问题

为什么会出现多种整数类型?

- 有些 CPU 的自然字大小，若认为没有表示更大数的需要，会将 long 类型和 `int` 类型定义相同的长度。
- 很多场合不要用到太大的整数，于是创建了更节省空间的 short 类型。

整数的上溢

问题

若整数太大，超出整数类型的范围会发生什么?

整数的上溢

```
 1 #include <stdio.h>
 2 #include <limits.h>
 3 int main(void)
 4 {
 5   int i = INT_MAX;
 6   unsigned int j = UINT_MAX;
 7   printf("i = %d, i+1 = %d, i+2 = %d\n",
 8       i, i+1, i+2);
 9   printf("j = %u, j+1 = %u, j+2 = %u\n",
10       j, j+1, j+2);
11   return 0;
12 }
```

整数的上溢

```
$ gcc intOverflow.c
$ ./a.out
i = 2147483647, i+1 = -2147483648, i+2 =
-2147483647
j = 4294967295, j+1 = 0, j+2 = 1
```

整数的上溢

当达到最大值时，将溢出到起始点。

- 对于 unsigned int 类型，起始点是 0；
- 对于 int 类型，起始点为-2147483648。

注意：当整数溢出时，编译器不会给出任何提示，故编程时必须谨慎对待此类问题。