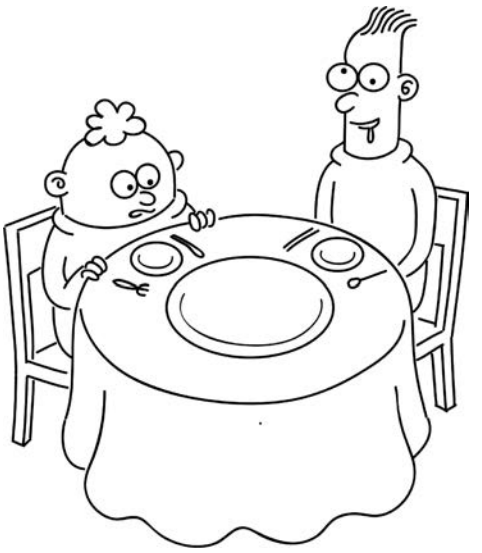
第 3 章

数 据 类 型



在第 2 章已经用到一些数据类型，比如 int 和 float 等。C 语言中的数据类型可以分为基本数据类型、派生数据类型和用户自定义数据类型。本章重点介绍基本数据类型，比如整数类型(整型)、浮点类型、字符类型等，以及数据类型之间的转换。


微课视频

3.1 C 语言中的数据类型

C 语言中的数据类型如图 3-1 所示。

解释说明：

(1) 基本数据类型，是 C 语言内置的数据类型。主要分为整数类型、浮点类型、字符类型和 void。

(2) 派生数据类型，是从基本数据类型衍生出来的数据类型。主要分为函数类型、数组类型和指针类型。

(3) 用户自定义数据类型，是用户自定义的数据类型。主要分为结构体、联合体、枚举

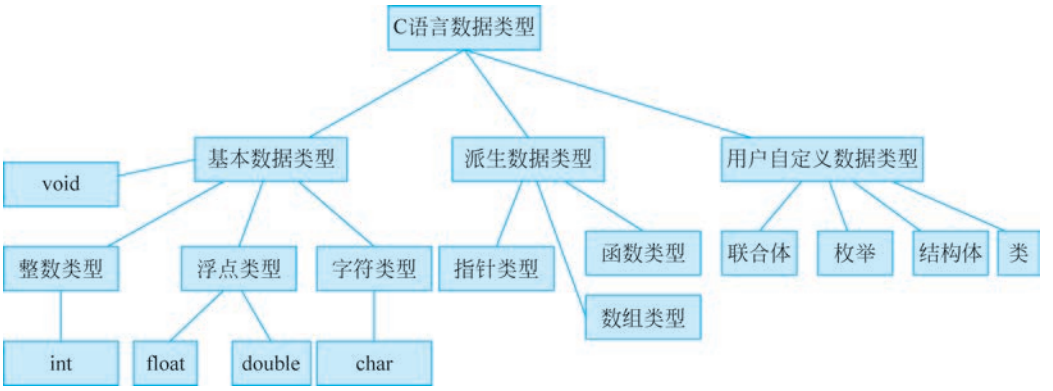


图 3-1 C 语言中的数据类型

和类。

其中：

① void：表示空数据，或者没有返回值，或者是没有分配内存空间的数据。

② 函数类型：函数本身也是一种数据类型。

3.2 整数类型

在 C 语言中使用 int 关键字声明整数类型数据，它用于存储整数数据，所占用的内存主要取决于编译器(32 位或 64 位)。通常，整数类型在使用 32 位编译器时占用 4 字节的内存空间，它的取值范围是 -2^{31}～$2^{31}-1$。

示例代码如下：

```
#include <stdio.h>

// 声明全局变量
int number1 = 100;                                          ①
char name[] = "Ben";

int main() {

    short int number2 = 500;
    printf("number1: %d\n", number1);
    printf("number1 所占用字节数: %lu\n", sizeof(number1));      ②

    printf("number2: %hd\n", number2);
    printf("number2 所占用字节数: %lu\n", sizeof(number2));

    return 0;
}
```

上述代码第①行声明整数类型变量 number1。

上述代码第②行 sizeof 函数用来计算 number1 的字节数。

上述示例代码运行结果如下：

```
number1: 100
number1 所占用字节数: 4
number2: 500
number2 所占用字节数: 2
```


微课视频

3.2.1 数据类型修饰符

读者是否注意到 3.2 节示例中声明变量 number2 的数据类型是 short int，short int 也是整型，被称为短整型。事实上，short 是基本数据类型的修饰符，这样的修饰符有 4 个。

这些修饰符的描述是正确的，它们的具体使用方式如下：

（1）unsigned：表示无符号的数据，所修饰的数据类型只能存储零或正数。

（2）signed：表示有符号的数据，所修饰的数据类型能存储零、负数或正数。

（3）short：所修饰的数据类型占用的内存空间要小一些，只能修饰 int 类型，即 short int。

（4）long：所修饰的数据类型占用的内存空间要大一些，主要用来修饰整数类型和浮点类型。

需要注意的是，C 语言标准并没有规定 short、long 和 long long 数据的确切大小，因此在不同的平台和编译器下，它们的大小可能会有所不同。但是，这些修饰符的使用方式和作用是一致的。

数据类型修饰符所占用字节数和取值范围如表 3-1 所示。

表 3-1 数据类型修饰符所占用字节数和取值范围

数据类型	所占用字节数	取值范围
short int	2	$-2^{15}\sim2^{15}-1$
unsigned short int	2	$0\sim2^{16}-1$
unsigned int	4	$0\sim2^{32}-1$
int	4	$-2^{31}\sim2^{31}-1$
long int	4	$-2^{31}\sim2^{31}-1$
unsigned long int	4	$0\sim2^{32}-1$
long long int	8	$-2^{63}\sim2^{63}-1$
unsigned long long int	8	$0\sim2^{64}-1$
signed char	1	$-128\sim127$
unsigned char	1	$0\sim255$

注意 字符型也是整型的一种。在计算机内部保存字符时使用的是 ASCII，例如字符'a'的 ASCII 值是 97，字符'A'的 ASCII 值是 65。

数据类型修饰符示例代码如下：

```
// 3.2.1 数据类型修饰符

#include <stdio.h>

int main() {
    // 定义变量并初始化
    unsigned short int number1 = 600;
    long int number2 = 700;
    unsigned long int number3 = 800;
    unsigned long long int number4 = 900;
    signed char number5 = 97;
    unsigned char number6 = 255;

    // 输出变量的值和所占用的字节数
    printf("number1: %u\n", number1);
    printf("number1 所占用字节数: %zu\n", sizeof(number1));

    printf("number2: %ld\n", number2);
    printf("number2 所占用字节数: %zu\n", sizeof(number2));

    printf("number3: %lu\n", number3);
    printf("number3 所占用字节数: %zu\n", sizeof(number3));

    printf("number4: %llu\n", number4);
    printf("number4 所占用字节数: %zu\n", sizeof(number4));

    printf("number5: %c\n", number5);
    printf("number5 所占用字节数: %zu\n", sizeof(number5));

    printf("number6: %u\n", number6);
    printf("number6 所占用字节数: %zu\n", sizeof(number6));

    return 0;
}
```

上述示例代码运行结果如下：

```
number1: 600
number1 所占用字节数: 2
number2: 700
number2 所占用字节数: 4
number3: 800
number3 所占用字节数: 4
number4: 900
number4 所占用字节数: 8
number5: a
number5 所占用字节数: 1
number6: 255
number6 所占用字节数: 1
```

注意上述示例运行时，number5 在计算机中保存的是 97，输出到控制台的是字符 a。


微课视频

3.2.2 数据溢出

某种数据类型的变量如果是有范围的，它保存的数值超出其范围，就会导致数据溢出。数据溢出虽然不会有编译错误，但会有警告！例如 unsigned char 数据类型最大的容纳值是 255，若将 300 赋值给它，在编译时就会有警告。

示例代码如下：

```
// 3.2.2 数据溢出
#include <stdio.h>

unsigned short int number1 = 600;
long int number2 = 700;
unsigned long int number3 = 800;
unsigned long long int number4 = 900;

signed char number5 = 97;
unsigned char number6 = 300;          // 数据溢出                    ①

int main() {
  printf("number1: %d\n", number1);
  printf("number1 所占用字节数: %lu\n", sizeof(number1));

  printf("number2: %ld\n", number2);
  printf("number2 所占用字节数: %lu\n", sizeof(number2));

  printf("number3: %lu\n", number3);
  printf("number3 所占用字节数: %lu\n", sizeof(number3));

  printf("number4: %llu\n", number4);
  printf("number4 所占用字节数: %lu\n", sizeof(number4));

  printf("number5: %d\n", number5);
  printf("number5 所占用字节数: %lu\n", sizeof(number5));

  printf("number6: %d\n", number6);
  printf("number6 所占用字节数: %lu\n", sizeof(number6));
  return 0;
}
```

上述代码第①行声明变量 number6 是 unsigned char 数据类型，unsigned char 数据类型最大的容纳值是 255，但是本例赋给 300 则会发生数据溢出。

上述示例代码运行结果如下：

```
number1: 600
number1 所占用字节数:2
number2: 700
number2 所占用字节数:4
number3: 800
```

```
number3 所占用字节数:4
number4: 900
number4 所占用字节数:8
number5: a
number5 所占用字节数:1
number6: 44
number6 所占用字节数:1
```

注意,number6 变量输出的结果是 44,这是数据溢出导致的。number6 变量被赋值 300,300 在计算机内部被存储为二进制数,又因为 number6 被声明为 unsigned char 数据类型,所以 300 被存储的二进制数只能保留低 8 位,高 4 位溢出,结果是十进制数 44。

3.2.3 整数的表示方式


微课视频

整数除了可以用十进制表示,还可以使用多种进制表示,例如:二进制、八进制和十六进制。

(1) 二进制数:以 0b 或 0B 为前缀。

(2) 八进制数:以 0 为前缀。

(3) 十六进制数:以 0x 或 0X 为前缀。

注意 二进制、八进制和十六进制前缀中是阿拉伯数字 0,不是英文字母 o。

示例代码如下:

```
// 3.2.3 整数的表示方式


#include <stdio.h>

int decimalInt = 28;
int binaryInt1 = 0b11100;
int binaryInt2 = 0B11100;
int octalInt = 034;
int hexadecimalInt1 = 0x1C;
int hexadecimalInt2 = 0X1C;

int main() {
    printf("decimalInt: %d\n", decimalInt);
    printf("binaryInt1: %d\n", binaryInt1);
    printf("octalInt: %d\n", octalInt);
    printf("binaryInt2: %d\n", binaryInt2);
    printf("hexadecimalInt1: %d\n", hexadecimalInt1);
    printf("hexadecimalInt2: %d\n", hexadecimalInt2);
    return 0;
}
```

上述示例代码运行结果如下:

```
decimalInt: 28
binaryInt1: 28
octalInt: 28
binaryInt2: 28
hexadecimalInt1: 28
hexadecimalInt2: 28
```


微课视频

3.3 浮点类型

在C语言中通过float、double及long double关键字声明浮点类型数据，如表3-2所示。

表3-2 浮点类型数据

数据类型	具体名称	占用字节数
float	单精度浮点型	4
double	双精度浮点型	8
long double	扩展双精度浮点型	Microsoft C编译器是8，GCC编译器是16

提示 Microsoft C编译器是微软提供的编译器，Visual Studio工具自带该编译器；GCC编译器(GNU Compiler Collection，GNU编译器套件)是由GNU开发的编译器。

浮点类型数据示例代码如下：

```
// 3.3 浮点类型
#include <iostream>

using namespace std;

float digit1 = 3.36;
double digit2 = 1.56e-2;

int main() {

    long double digit3 = 0.0;
    cout << "digit1 :" << digit1 << endl;
    cout << "digit1 所占用字节数:" << sizeof(digit1) << endl;

    cout << "digit2:" << digit2 << endl;
    cout << "digit2 所占用字节数:" << sizeof(digit2) << endl;

    cout << "digit3:" << digit3 << endl;
    cout << "digit3 所占用字节数:" << sizeof(digit3) << endl;
    return 0;
}
```

示例代码运行结果如下：

```
digit1 : 3.360000
digit1 所占用字节数: 4
digit2: 0.015600
digit2 所占用字节数: 8
digit3: 0.000000
digit3 所占用字节数: 16
```

3.4 字符类型


微课视频

字符类型表示单个字符，如表 3-3 所示，在 C 语言中通过 char 和 wchar_t 关键字声明字符类型数据。

表 3-3 字符类型数据

数据类型	名 称	占用字节数
char	窄字符	1
wchar_t	宽字符	2 或 4

示例代码如下：

```
// 3.4 - 1 字符类型


#include <stdio.h>

int main() {
    char ch1 = 'A';                                    ①
    wchar_t ch2 = L'A';                                ②

    printf("ch1: %c\n", ch1);
    printf("ch1 所占用字节数: %lu\n", sizeof(ch1));

    wprintf(L"ch2: %lc\n", ch2);
    printf("ch2 所占用字节数: %lu\n", sizeof(ch2));

    return 0;
}
```

上述代码第①行是声明窄字符数据类型变量 ch1。

代码第②行是声明宽字符数据类型变量 ch2，“L'A'”中的“L”表示该字符是宽字符数据类型，在内存中占用 2 字节。

上述示例代码运行结果如下：

```
ch1: A
ch1 所占用字节数: 1
ch2: A
ch2 所占用字节数: 2
```

另外，在 C 语言中，使用反斜杠(\)表示字符转义。反斜杠后面跟着一个或多个字符，

用于表示一些特殊的字符或符号。常见的转义字符如表 3-4 所示。

表 3-4 常见的转义字符

转义字符	说明	转义字符	说明
\t	水平制表符 Tab	\"	双引号
\n	换行符	\'	单引号
\r	回车符	\\	反斜杠

示例代码如下：

```
// 3.4 - 2 字符转义
#include <stdio.h>

int main() {
  char specialCharTab1[] = "Hello\tWorld.";
  char specialCharNewLine[] = "Hello\nWorld.";
  char specialCharQuotationMark[] = "Hello\"World.";
  char specialCharApostrophe[] = "Hello\'World\'.";
  char specialCharReverseSolidus[] = "Hello\\World.";
  printf("水平制表符 tab: %s\n", specialCharTab1);
  printf("换行符: %s\n", specialCharNewLine);
  printf("双引号: %s\n", specialCharQuotationMark);
  printf("单引号: %s\n", specialCharApostrophe);
  printf("反斜杠: %s\n", specialCharReverseSolidus);

  return 0;
}
```

上述示例代码运行结果如下：

```
水平制表符 tab: Hello World.
换行符: Hello
World.
双引号: Hello"World.
单引号: Hello'World'.
反斜杠: Hello\World.
```


微课视频

3.5 布尔类型

在 C 语言中，没有原生的布尔(bool)类型数据，即没有 bool 类型数据。通常，开发人员会使用整数类型来模拟布尔类型，其中 0 代表 false(假)，1 代表 true(真)。C 语言提供了<stdbool.h>头文件，定义了布尔类型和 true/false 常量。因此，在 C 语言中，开发人员可以通过包含<stdbool.h>头文件使用布尔类型数据。下面是一个使用布尔类型数据的示例。

```
// 3.5 布尔类型
#include <stdbool.h>                    ①
#include <stdio.h>
```

```
int main() {
    bool x = true;
    bool y = false;

    printf("x = %d, y = %d\n", x, y);
    printf("sizeof(bool) = %lu\n", sizeof(bool));

    return 0;
}
```

上述代码第①行包含头文件< stdbool. h >,后面的代码中才可以声明变量为布尔类型。

上述示例代码运行结果如下:

```
x = 1, y = 0
sizeof(bool) = 1
```

3.6 数据类型之间的转换

不同的数据类型是可以相互转换的,但转换比较复杂,本章只讨论基本数据类型之间的相互转换。基本数据类型之间的相互转换分为以下两种:

(1) 自动类型转换,也被称为隐式类型转换;

(2) 强制类型转换,也被称为显式类型转换。

3.6.1 自动类型转换


微课视频

自动类型转换是将小容量的数据类型赋值给大容量的数据类型,如图 3-2 所示,从下往上是自动类型转换,不会发生数据精度丢失;相反,从上往下需要强制类型转换,可能会发生数据精度丢失。

自动类型转换不仅在赋值时发生,在进行数值计算时也会发生。编程语言会自动将一个数据类型转换为另一个数据类型,以便进行计算,这种转换是自动进行的,程序员无须手动指定转换。例如,在将一个整数和一个浮点数相加时,整数会被自动转换为浮点数,以便进行计算。转换规则如表 3-5 所示,不同的数据类型之间有不同的转换规则,程序员需要了解这些规则以避免错误。

表 3-5 数据类型转换规则

操作数 1 的类型	操作数 2 的类型	转换后的类型
byte、short、char	int	int
byte、short、char、int	long	long
byte、short、char、int、long	float	float
byte、short、char、int、long、float	double	double

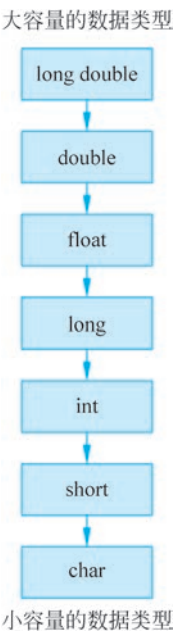


图 3-2 数据类型转换

示例代码如下:

```
// 3.6.1 自动类型转换

#include <stdio.h>

int main() {
    int a = 10;
    float b = 3.14;
    float c = a + b;   // 这里会发生自动类型转换,a 会被转换成 float 类型,以便与 b 进行计算
    printf("c = %f\n", c);
    return 0;
}
```

上述示例代码运行结果如下：

```
c = 13.140000
```


微课视频

3.6.2 强制类型转换

将大容量的数据赋值给小容量的数据时，需要强制类型转换，进行强制类型转换时会将数据高位截掉，所以可能会导致数据精度丢失。

强制类型转换的语法格式如下：

```
(目标类型) 表达式;
```

示例代码如下：

```
// 3.6.2 强制类型转换

#include <stdio.h>

int main() {
    // int 型变量
    int i1 = 10;
    short int b1 = (short int)i1;   //把 int 类型变量强制转换为 short int 类型   ①

    int i2 = (int)i1;
    int i3 = (int)b1;

    float c1 = i1 / 3;              // 小数部分被截掉                              ②
    printf("%f\n", c1);
    //把 int 类型变量强制转换为 float 类型
    float c2 = (float)i1 / 3;                                                     ③

    long long int yourNumber = 6666666666L;                                       ④
    printf("%lld\n", yourNumber);
    int myNuber = (int)yourNumber;                                                ⑤
    printf("%d\n", yourNumber);

    return 0;
}
```

(1) 上述代码第①行将 int 类型的变量 i 强制转换为 short int 类型，这显然没有必要，但在语法上是允许的。

(2) 代码第②行中变量 i 除以 3 的结果中有小数，但由于两个操作数都是 int 类型，所以小数部分被截掉了，结果是 3；该结果被赋值给 float 类型的变量 c1，最后 c1 保存的结果是 3.0。

(3) 代码第③行中整型 i1 与 float 类型进行运算，结果是 float 类型，不会截掉小数部分。

(4) 代码第④行声明一个很大的长整数 yourNumber。

(5) 代码第⑤行中，由于 yourNumber 数据很大，所以高位被截掉，导致数据精度丢失。

提示 %lld 是用于输出 long long 类型整数的格式转换控制符。

上述示例代码运行结果如下：

```
3.000000
8
6666666666
-1923267926
```

从运行结果可见，原本的 6666666666L 变成了负数，这是因为数据的高位被截掉，导致数据精度丢失。

3.7 动手练一练

1. 选择题

(1) 以下哪个数据类型可以存储的整数范围最大？(　　)

A. short int　　B. unsigned int

C. signed int　　D. long int

(2) 下面哪个数据类型可以存储的小数范围最大？(　　)

A. float　　B. double　　C. long double　　D. char

(3) 下面哪个数据类型不能存储负数？(　　)

A. unsigned int　　B. signed int　　C. char　　D. short int

(4) 假设一个字符占用 1 字节(byte)，那么下面哪个数据类型可以存储的字符数最多？(　　)

A. char　　B. signed char　　C. unsigned char　　D. wchar_t

2. 判断题

(1) 将小容量的数据赋值给大容量的数据时，数据类型是自动转换的。　　(　　)

(2) 将整型与浮点类型进行计算，结果还是整型。　　(　　)

3. 编程题

编写程序，计算整数 7 除以整数 5 的结果，将运算结果输出到控制台，并解释输出结果。