C Language

Jianfeng He

第一章　C 程序设计初步

§1.1　C 程序的结构和处理过程

1.1.1　语言特点概览

1). **关键字及控制语句:** 30 多个关键字, 9 种控制语句
2). **运算符:** 共有 34 种运算符
3). **数据类型:** 大致有 8 种: 整型, 实型, 字符型, 数组类型, 指针类型, 结构体类型, 共用体类型, 枚举类型

1.1.2　基本程序结构

一下以一个输出 “hello, world” 的程序为例, 展现 C 程序的基本结构:

```c
#include <stdio.h>

// comment 1
/* comment 2-1
 * comment 2-2
 * comment 2-3 */

int main() {
    printf("hello, world");
    return 0;
}
```

将其特点总结如下:

1). 在程序的头部通过 `#include <stdio.h>` 调用标准库头文件.
2). 用双斜线 `//` 表示单行注释, 用双斜线插入星号 `/**/` 表示多行注释
3). 只有一个名为 `main` 的主程序. 在 `main` 的后面有一对小括号 `()`.
4). 主程序返回一个整数, 其值为 `0`. 其实这是 Linux 系统的一种惯例, 程序执行成功则返回 `0`, 失败则返回非 `0` 错误吗.
5). 每一条单独的语句都以分号 `;` 结尾, 忽略分号 `;` 是初学者最容易犯的错误.
6). 用一对大括号 `{}` 将主程序的内容括起.

1.1.3　C 程序处理过程

基本处理过程可以分为代码编写, 程序编译, 连接程序, 运行与调试等四个部分.

1). **代码编写:** 除了使用各种 IDE (intergrated development environment, 集成开发环境) 以外, 使用 Vim, EMacs, VSCode 等文本编辑器也是不错的选择. 为减少语法错误, 还可以利用 `ccls`

一类的静态语法检查工具.

2). **程序编译:** 常用的编译器有 `gcc`, `clang` 等. 若源程序没有语法错误, 在这一步将会生成二进制文件.

3). **链接程序:** 获得程序的目标文件后, 通过开发环境中的 "连接" 命令, 将目标文件和由开发环境提供的标准库函数, 启动代码等进行连接, 形成可执行文件 (后缀为 `.exe`). (教材上是如此说明的, 故而原样照抄. 不知道这是不是 Windows 环境下的特有操作? Linux 环境下似乎可以直接运行编译得到的二进制文件, 感觉没有接触过连接程序这一步.)

4). **运行及调试:** 在没有语法错误的情况下, 代码输出结果仍然有可能存在问题, 此时需要进行调试.

§1.2　C 语言的基本数据类型

1.2.1　C 语言的标识符

用户使用的变量名, 函数名等统称为标识符, 标识符的使用有如下规则:

1). 只能由数字, 字母, 下划线构成

2). 第一个符号不能是数字

3). 标识符对大小写敏感

4). 不能用保留字作为用户标识符

1.2.2　变量声明

与 Python, Perl, Julia 等动态语言不同, C 语言是静态语言, 变量在使用前必须进行声明 (后面将看到, 函数也是如此). 声明的一般形式为 (中括号内表示可选项):

```
<data_type> <var_1>[=<value_1>], <var_2>[=<value_2>], ...
```

可以看出, 变量声明的特点如下:

1). 可以一次声明多个变量

2). 可以在声明的同时对变量进行初始化

1.2.3　整型数据

在 16 位计算机中, 整型数据占 16 个 bit, 也就是 2 个 byte. 若带符号, 则可表示范围为 -32,786 ~ 32,767 ($-2^{15} \sim 2^{15}-1$) 的整数. 若为 32 位的计算机, 则可表示范围为 -2,147,483,648 ~ 2,147,483,647 ($-2^{31} \sim 2^{31}-1$).

C 语言中, 可以有十进制, 八进制, 十六进制三种整型常量:

1). **十进制整型常量:** 没有特殊标注时认为是十进制, 如: `452`.

2). **八进制整型常量:** 以 `0` 开头的是八进制整型常量, 如: `0777`, `-011`

3). **十六进制整型常量:** 以 `0x` 开头的是十六进制整型常量, 如: `0x123`, `0xff`

若按照数据位数来区分, 整型变量有 **基本整型**, **短整型**, **长整型**, **无符号整型** 四种, 依次通过关键字 `int`, `short`, `long`, `unsigned` 表示.

1.2.4　实型数据

实型数据又称 **浮点型数据**, 一般有两种表示方法: **小数形式** 与 **指数形式**.

小数形式既一般的小数书写方法, 无须特别强调; 指数形式以 `E` 或 `e` 表示指数部分, 如: `1.45e4`, `2.5639E5`. 对于指数形式:

Notation 1.1 :

1). 指数部分只能为整数, 如: `1.53E1.5` 是错误的表示法.

2). 指数部分可以为负数, 如: `4.53E-3`

3). 即使小数部分是 `1` 也不能省略, 如: `E4` 为错误的写法, 正确的写法应为 `1E4`

按占据的比特数的不同, 可以将实型数据分为 **单精度** 与 **双精度** 两种, 二者分别用标识符 `float` 与 `double` 来进行声明. 其中, 单精度数占 4 个 bit, 可提供 6 ∼ 7 位有校数字; 双精度占 8 个 bit, 可提供 15 ∼ 16 位有校数字.

1.2.5　字符型数据

字符型数据用关键字 `char` 进行声明. 字符型数据占 1 个字节.

转义码表见表 1.1:

表 1.1: escape char

转义字符	意义	功能解释
`\0`	NULL	字符串结束符
`\b`	退格	将光标左移一个字符
`\n`	换行	光标移向下一行的开始
`\f`	换页	(打印机) 换到下一页
`\t`	水平制表符	把光标移到下一个制表位
`\\`	反斜杠	输出反斜杠
`\"`	双引号	输出双引号
`\'`	单引号	输出单引号
`\a`	响铃	警报响铃
`\<num><num><num>`	ASCII 字符	输出八进制数字对应的 ASCII 字符
`\x<num><num>`	ASCII 字符	输出十六进制数字代表的 ASCII 字符

Notation 1.2 :

1). `printf('%c', 'textbackslash 101')` 与 `printf('%c', 101)` 的输出值分别是 `A` 与 `e`. 这是由于前者按八进制获取 ASCII 字符, 后者进行了隐式的转义, 按照十进制获取 ASCII 字符.

2). ASCII 编码中, 数字 0 ∼ 9 对应的编码是 48 ∼ 47; 大写字母 A ∼ Z 对应的编码是 65 ∼ 90; 小写字母 a ∼ z 对应的编码是 97 ∼ 122

3). 除了 `scanf()` 与 `pringf()`, 还可通过 `getchar()` 与 `putchar(<char>)` 来输入/输出单个字符.

1.2.6　字符串常量

通过将一串字符用双引号括起 `"<string>"` 来表示字符串常量. 应当注意, C 语言没有内置字符串变量, 所谓的字符串变量是通过字符列表来实现的, 其相当与一个元素为字符, 并以 `\0` 结尾的字符列表. 所以字符串变量对应的字符串长度, 实际上等于字符串列表的长度减一.

1.2.7　宏

严格来说, 宏并不是 C 语言的一种内置数据类型, 而是在编译前对源文件进行处理的语句. 预编译语句 `#define <A> <B>` 的效果是: 在编译时, 将源文件中的 `<A>` 替换为 `<B>`. 如: 可以用语句 `#define PI 3.1415926` 来定义圆周率 `PI`.

§1.3　输入与输出

1.3.1　格式化输出: printf()

1. 简单输出

```
printf("<string>");
```

这条语句将直接输出字符串 <string>

2. 基本语法

```
printf(%[-][*][m.n][l/h]<format_charactor>, <var_1>, <var_2>, ...);
```

其中各项的意义为:

1). %: 占位符, 一个变量对应一个占位符, 该占位符后面的那些字符表示对该变量的格式控制.

2). <format_charactor>: 控制数据输出的模式, 常用的格式输出符有见表 1.2

表 1.2: options of format charactor

format charactor	meaning
d	decimal, 以 10 进制形式输出带符号整数, 正数不输出负号
o	octal, 以 8 进制形式输出无符号整数, 无前缀 o
x	hexadecimal, 以 16 进制形式输出无符号整数, 无前缀 ox
u	unsign, 以 10 进制输出无符号整数
f	float, 以小数形式输出单、双精度实数
e	exponential, 以指数形式输出单、双精度实数
g	以 %f 或 %e 中较短的输出宽度输出
c	char, 输出单个字符
s	string, 输出字符串

3). l/h: 长度修正符, 用于指定对应位置输出数据是按长类型数据输出还是按短类型数据输出.

4). m,n: 域宽可选项, 用于指定对应输出项所占的输出宽度, 即指定用多少字符位置来显示对应输出数据.

5). *: 对应输出表列中两个连续数据项, 其意义是用前一个数据项的值作为后一个数据项输出的指定域宽.

6). -: 减号可选项, 用于指定对应输出数据的对齐方向. 当选用减号时, 输出数据左对齐, 否则右对齐.

Notation 1.3 :

1). 若 a 为 int 类型的数据, 则表达式 printf("%c", a) 将会输出 a 对应的 ASCII 码值.

2). 当输出格式为长型浮点数 %lf 时, 默认输出 6 位小数. (如果只是 %f, 似乎无论单、双精度, 仍是 6 位小数?)

3). 对于 32 位的编译器, %d 与 %ld 是相同的.

4). 对于整型数据, 若输出域宽大于数的长度, 将会用空格将域宽补全. 若再在域宽前加上 0, 则将以 0 来填充空格. 如: 以 %5d 的格式输出 152, 输出结果将会变为 ␣␣152, 如以 %05d 的形式, 则输出结果为 00152

5). 若以 %#o 或 %#x 的形式输出八进制或十六进制数, 则输出结果会对输出的进制进行标识, 八进制用 0, 十六进制用 0x. 如要输出 100, 采用四种输出格式将会对应如下结果:

表 1.3: 输出八进制与十六进制数

输入	格式控制字符	输出
100	%o	144
	%#o	0144
	%x	64
	%#x	0x64

§1.4　基本运算符和表达式运算

1.4.1　C 语言表达式的概念

1.4.2　赋值运算符

1. 类型转换　当赋值语句两侧的数据类型不一致时, 会自动将右侧数据转换为左侧数据. 具体转换规则有:

1). **实型 → 整型:** 不进行四舍五入, 直接舍去所有小数部分.
2). **整型 → 实型:** 整数部分不变, 小数部分为 0.
3). **字符型 → 整型:** 直接将字符的 ASCII 码放到整型变量的最低八位中.
4). **整型 → 字符型:** 将整型数据的低八位值对应的 ASCII 码赋值给字符变量.

1.4.3　算数运算符

1. 单目运算符

2. 双目运算符　共有五个: 加号 +, 减号 -, 乘号 *, 除号 /, 求模 %.

Notation 1.4 :

1). 两个整型数据相除, 结果仍然是整数, 取整方式为截取法.
2). 求模运算就是取余数运算, 参与求模运算的两个数必须都是整数, 运算的符号与第一个相同.

1.4.4　自增自减运算符

自增运算符 (++) 与自减运算符 (--) 用以对变量进行增量赋值. 允许的语法有:

```
++<var_name>;
--<var_name>;
<var_name>++;
<var_name>--;
```

Notation :

1). 前缀模式与后缀模式的返回值是不同的. 前缀模式返回变量进行了增量运算后的值, 后缀模式返回进行增量运算前的值. 如:

```
int i = 5, j = 5;
printf("%d, %d", ++i, j++);
```

该程序的返回值为:

```
6, 5
```

2). 自增、自减运算不能用于字面量

1.4.5　复合赋值运算符

1.4.6　逗号运算符

逗号运算符是优先级最低的运算符, 用于将多个表达式连接成一个表达式, 并从左到右进行计算, 并返回最后一个表达式的结果. 一般的语法为:

```
<expr1>,<expr2>,<expr3>...
```

Notation 1.5 :

1). 可以利用逗号表达式进行多个变量的赋值, 如:

```
a = 1, b = 2, c = 3;
```

Example 1.1 逗号表达式

可以用括号将逗号表达式进行分隔. 比较下面两个变量 x, y 的输出的差异:

```
int x, y;
x = 1, 2, 3, 4;
y = (1, 2, 3, 4);
printf("x = %d, y = %d", x, y)
```

输出结果为:

```
x = 1, y = 4
```

■

1.4.7　sizeof 运算符

sizeof 运算符的功能是返回其所测试的数据对象所占存储单元的字节数. 基本语法为:

```
sizeof(<var>)
```

其中 <var> 可以是:

1). 某个具体的变量名
2). 某种数据类型常量
3). 某种数据类型的名字. 这也包括数组、结构体等构造数据类型
4). 一个合法的 C 表达式. 此时不会进行具体的计算, 而只是得出计算结果的数据类型, 并返回该数据类型对应的存储空间大小.

1.4.8　数据类型转换

大致可分为**隐式转换**与**强制转换.**

1. 隐式转换

转换的方式如图 1.1所示. 其中:

1). 纵向: 进行运算时, 若存在优先级较高的部分, 则优先级低的向优先级高的转换
2). 横向: 无条件转换. (比如两个 char 型数据相加将会强制转换为 int 型数据?)

2. 显式转换

下面的语句将表达式的结果转换成指定的类型进行输出:

```
(<type>)(<expr>)
```

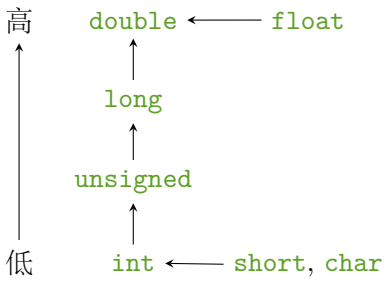

图 1.1: 隐式转换规则

Example 1.2 显示转换

```
double f = 4.56f;
int zf;
zf = (int)(f + 1);
printf("zf = %d, f = %.2lf\n", zf, f);
```

输出结果为:

```
zf = 5, f = 4.56
```

■

第二章　数组与字符串

§2.1　数组的定义及数组元素的引用

1. 数组的定义

```
<element_type> <array_name>[<array_length>];
```

Notation：

1). 数组索引从 0 开始

2). 通过 `sizeof(<array_name>)` 来获取数组的长度

2.1.1　二维和多维数组

1. 数组存储规则对于多维数组, 其存储规则可以概括为: 序号越靠右, 存储优先级越高. 此外, 若将数组的序号看成是一个多进制数, (e.g., `[8][9][2]` 看成数字 892), 并记该数为 a, 则存储顺序是按照 a 增加的顺序来的.