简易C语言教程

一、C语言的初步使用

1.选择合适的编译器

与其他编程语言不同，C语言的编译器种类繁多，甚至有不同的语言标准。因此，在使用C语言时，首先要确定合适的编译器和语言标准。

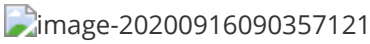
本书的代码采取的是C99语言标准。在编译器的选择上，读者可以根据自己的情况自行决定，推荐使用Visual Studio。

2.第一个C语言程序

我们先写一个简单的C语言程序。

```c
#include <stdio.h>
int main(void)
{
    int num;
    num = 0;
    printf("Hello World!\n");
    printf("num = %d\n",num);
    return 0;
}
```

在Visual Studio上写好代码后，我们点击调试->开始执行，或者按Ctrl+F5运行上述代码。可以看到，屏幕上打印出了一行"Hello World!"。



我们可以看出这个C程序的基本构成：

1）指令和头文件：对应上述代码中的`#include <stdio.h>`，它的作用相当于把stdio.h文件中的所有内容都输入到该行所在的位置。

2）函数：对应上述代码中的`int main(void)`及其大括号下的部分。C程序一定要有一个main函数（特殊情况暂不考虑），它是C程序的核心。你可以创建多个函数，但main函数必须是开始的函数。main前面的int代表的是main函数的返回类型，说明main返回的值是整数。main小括号（()）内的内容代表的是传入函数的信息，其中main(void)代表的是函数不传入任何信息。main大括号（{}）内的内容代表的是函数执行的命令。我们可以在函数里写合适的代码，来完成我们想要的指令。

对于一般的C程序而言，除了main函数之外，可能还会有其他多个函数。这些函数是C程序的构造块，它们相互协调相互作用，共同构成了一个完整的C程序。

接下来，我们来详细看一下main函数里面的内容。它主要由两部分组成：声明和语句。

1）声明：对应程序里的`int num;`这行代码完成了两件事，一是说明在函数中有一个名为num的变量，二是表明num这个变量是一个整数。int是一种数据类型，它代表的是整数；除了int之外，还有其他的数据类型。下表整理了一些比较常用的数据类型：

名称	数据类型
int	整数型
float	浮点型
double	双精度浮点型
char	字符型

同时int也是C语言中的关键字。在C语言中，关键字是语言定义的单词，不能用作其它用途，因此不能把int作为变量名；num是一个标识符，是变量的名称。需要注意的是，在C语言中，所有变量都必须先声明才能使用。

2）语句：语句是C程序的基本构件块。一条语句就相当于一条完整的计算机指令。在上述的例子中，main函数里除了变量声明外，其他的部分都是语句。语句又分为简单语句和复合语句。

这里面比较特殊的是return语句，对应例子中的return 0。它代表的是函数的返回值（执行函数中的代码所得到的结果）。对于没有返回值的函数（void类型）,return语句可以不加，而对于其他类型的函数，return语句是需要加的。

return语句可以有多个，可以出现在函数里的任意位置，但每次调用函数都只能有一个return语句被执行，执行之后，return后面的语句就都不会执行了。

需要注意的是，语句和声明的末尾，都需加上“;”，以表明语句\变量的结束。

下面将详细讲解语句的概念和使用规则。

二、运算符、表达式、语句

为了理解语句，首先需要了解运算符和表达式是什么。

1.运算符

1.1 基本运算符和其他运算符

运算符的作用是用来表示算术运算。数学上的加减乘除符号(+,-,*,/)在C语言里都属于运算符，并且是基本运算符。除此之外，还有其他的基本运算符。总结后表格如下：

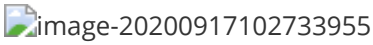
基本运算符	作用
=	赋值运算符，把等号右边的值赋给左边变量
+	做加法运算，把两侧的值相加
-	做减法运算，把左侧的值减掉右侧的值。除此之外，它也可以让值取相反数。
*	做乘法运算
/	做除法运算
()	括号，和数学中的用法一样

需要特别注意的是，在C语言中，除法运算的结果会因为数据类型的不同而不同！比如，浮点数除法的结果是浮点数，而整数除法的结果是整数。比如我们运行以下代码：

```c
#include "stdio.h"

int main(void) {
    printf("5/4 = %d\n",5/4);
    return 0;
}
```

正确的结果应该是5/4 = 1.25，然而程序运行后得到的却是5/4 = 1：



这种情况被称为“截断”。在做除法运算的时候，要特别牢记这一点。

除了基本运算符，还有一些其他比较常用的运算符，整理表格如下：

运算符	作用
%	求模运算符，用以求出左侧整数除以右侧整数后得到的余数.
++	递增运算符，将其运算对象递增1。该运算符可以放在变量前（前缀模式），也可放在变量后（后缀模式）；两种方式的区别在于递增行为发生的时间不同。
--	递减运算符，将其运算对象递减1。该运算符可以放在变量前（前缀模式），也可放在变量后（后缀模式）；两种方式的区别在于递增行为发生的时间不同。
< (<=)	关系运算符中的小于（小于等于），如果左侧变量的值小于（小于等于）右侧变量的值，则取值为1，否则为0。
> (>=)	关系运算符中的大于（大于等于），如果左侧变量的值大于（大于等于）右侧变量的值，则取值为1，否则为0。
==	关系运算符中的等于，如果左侧变量的值等于右侧变量的值，则取值为1，否则为0。
!=	关系运算符中的不等于，如果左侧变量的值不等于右侧变量的值，则取值为1，否则为0。

对于递增、递减运算符，前缀后缀模式会对代码产生不同的影响，比如我们运行下方代码：

```c
#include "stdio.h"

int main(void) {
    int i = 0;
    int j = 0;
    printf("i = %d\n",i++);
    printf("j = %d\n",++j);
    return 0;
}
```

得到的结果为：

可以看到打印出来的i，j两个值并不相同。因此我们需要特别注意递增递减运算符前后缀形式的不同。

1.2 运算符优先级

不同运算符的优先级是不同的。优先级高的运算符首先被程序执行，优先级相同的从左到右执行（=运算符除外）。因此我们需要了解各个运算符的优先级。整理后得到表格如下：

优先级	运算符	名称
1	()	括号
1	++	递增运算符（后缀形式）
1	--	递减运算符（后缀形式）
2	-	负号运算符
2	++	递增运算符（前缀形式）
2	--	递减运算符（前缀形式）
3	/	除法运算符
3	*	乘法运算符
3	%	求模运算符
4	+	加法运算符
4	-	减法运算符
5	<< (>>)	左移（右移）
6	> (>=)	大于（大于等于）
6	< (<=)	小于（小于等于）
7	==	等于
7	!=	不等于

2.表达式

表达式由运算符和运算对象组成。以下的例子都是表达式：

```
4
23
a*b+c-d/3+100
p = 3*a
x = q++
x > 3
```

C语言中，表达式的一个最重要的特性就是每个表达式都有一个值。这个值是按照运算符优先级的顺序执行获得的。在上述的例子中，前三个都比较清晰。对于有赋值运算符（=）的表达式，其值和赋值运算符左侧变量的值相同。对于有不等式的表达式，如果不等式成立，则值为1，否则为0。

3.语句

3.1 简单语句

上面我们简单介绍了运算符和表达式，接下来我们来正式介绍语句。

语句是C程序的基本构建块，它有很多类型，其中较为常见的赋值表达式语句——它是由赋值表达式构成的，比如：

```
a = 3*5 + 2;
```

函数表达式语句会引起函数调用。比如，调用`printf()`函数可以打印结果。

这里需要注意的是：赋值和函数调用都是表达式，这些语句本质上都是表达式语句。

3.2 复合语句

以上介绍的基本上都是简单语句，下面将介绍相对复杂一点的复合语句。

复合语句使用花括号括起来的一条或者多条语句，也被称之为块。比较常见的复合语句类型有：循环语句和分支语句。

3.3 循环语句

循环语句，顾名思义，就是让程序反复执行一段或者多段命令。循环语句一般都是复合语句，其中比较常见的两种类型有while语句和for语句。

3.3.1 while语句

while语句的通用格式如下：

```
while(expression){
    statement
}
```

其中expresssion代表的是表达式，它的作用是判断是否继续循环执行while语句内（花括号内的内容）的语句。如果表达式的值为1，则循环继续（从头执行花括号内的语句）；如果表达式的值为0，则循环结束，往下执行语句（花括号外的语句）。statement代表的是while语句内包含的语句。

通常，我们一般用关系表达式（含关系运算符）作为while循环的判断条件。如果我们想要终止while循环，则必须让while循环内的表达式的值有所变化。比如下面的代码就是错误的，它会导致程序陷入无限循环：

```
int index = 2;
while(index < 3){
    printf("hello world!\n");
}
```

如果不想让程序想入无限循环，则我们可以写成如下形式：

```
int index = 2;
while(index < 3){
    printf("hello world!\n");
    index++;
}
```

3.3.2 for语句

for语句的通用格式如下：

```
for(exp1;exp2;exp3){
    statement;
}
```

可以看到，for语句小括号里面含有三个表达式。第一个表达式exp1是初始化，它的作用在于初始化计数器的值；第二个表达式是测试条件，如果表达式为假（值为0），则结束循环；第三个表达式是执行更新，在每次循环结束后求值，更新计数。statement代表的是for语句里面包含的语句。

3.3.3 嵌套循环

嵌套循环指的是在一个循环内包含另一个循环。嵌套循环常用于按行和列显示数据。下面是一个嵌套循环的例子：

```
for(i=0;i<10;i++){
    for(j=0;j<10;j++){
        printf("i=%d,j=%d\n",i,j);
    }
}
```

3.4 if语句

基本的if语句的通用格式如下：

```
if(expression){
    statement;
}
```

if小括号内的表达式expression用于判断。如果表达式为真（1），则执行花括号内的statement（语句）；如果为假，则不执行花括号内的语句。

简单形式的if语句可以让程序选择执行或者不执行一条或者多个语句。除此之外，我们还可以用if和else，让程序在两个语句块中选择其一执行。其格式如下：

```
if(expression){
    statement1;
}else{
    statement2;
}
```

这段代码的含义是：如果expression为真，则执行statement1，否则执行statement2。

另外，我们还可以把if语句进行多层嵌套，比如可以写成如下格式：

```
if(expression1){
    statement1;
}else if(expression2){
    statement2;
}else{
    statement3;
}
```

这段代码的含义是：如果expression1为真，则执行statement1,否则继续判断expression2的真假——如果expression2为真，则执行statement2，否则执行statement3。

3.5 逻辑运算符

在讲完循环语句和if语句后，我们可以继续深入了解一种特殊的运算符——逻辑运算符。顾名思义，它是用来表明逻辑的。

逻辑运算符共有三种：与、或、非。具体如下：

逻辑运算符	含义
&&	与
||	或
!	非

假设exp1和exp2是两个简单的关系表达式，那么：

- 当且仅当exp1和exp2都为真时，exp1&&exp2才为真
- 如果exp1和exp2二者有其一为真，则exp1||exp2为真
- 如果exp1为真，则!exp1为假；如果exp1为假，则!exp1为真

三、数组和指针

1.数组

数组是由数据类型相同的一系列元素组成的。需要使用数组时，我们首先需要声明数组，告诉编译器数组内有多少元素以及其元素的类型。比如`int group[10]`或者`float group2[7]`，中括号内的数字表示的是数组的长度。除此之外，我们还可以用以都逗号分隔的值列表（用花括号括起来）来初始化数组，比如`int group[3] = {1,2,3}`。

在初始化数组后，我们可以对数组的某一位置的元素进行赋值。比如这样：

```
int a[3];
a[0] = 1;
```

其中`a[0] = 1;`里的中括号内的数字表示的是数组的下标，指明对应数组下标位置的数组元素。需要注意的是，数组下标是从0开始的，而且存在上界。因此，我们必须保证数组下标在有效范围内。对于`int a[3];`这一长度为3的数组来说，它的下标最大值为2。

2.多维数组

有时我们想存储矩阵或者表格形式的数据，这时我们就需要用到多维数组。

我们先拿二维数组举例。如果我们想要初始化二维数组，则首先要初始化一维数组。比如我们想要初始化一个二维数组，则可以这样表示：

```
int group[2][3] = {
    {1,2,3},{4,5,6}
};
```

可以看到最外层的花括号内含有2个长度为3的数值列表。可以理解为group这个二维数组含有两个长度为3的一维数组。如果花括号内的数值列表的长度小于对应的数组下标，则程序会默认把其他的元素初始化为0。比如：

```
int group[2][3] = {
    {1,2},{4,5,6}
};
```

可以看到，最外层花括号内的第一个数值列表的长度只有2，小于对应的数组下标3，因此程序会默认在花括号内第一个数组里数组下标大于1的元素设置为0。也就是等价为：

```
int group[2][3] = {
    {1,2,0},{4,5,6}
};
```

初始化时也可以省略内部的花括号，只保留最外层的花括号。只要保证初始化的数值个数正确，初始化的效果与上面相同。但如果初始化的数值不够，则按照先后顺序逐行进行初始化，直到用完所有值。后面没有值初始化的元素会被统一设置为0。比如：

```
int group[2][3] = {
    5,6,7,8
};
```

实际上等同于：

```
int group[2][3] = {
    {5,5,7},{8,0,0}
};
```

对于二维数组的讨论同样也适用于三维数组以及更多维的数组。比如我们可以声明一个三维数组：

```
int box[10][20][30];
```

可以理解为box这个三维数组中，含有10个大小为20×30的二维数组。

3.指针

指针是一个值为内存地址的变量。正如char类型的变量的值为字符，int类型的变量的值为整数，指针变量的值为地址。

假如一个指针变量名为ptr，可以编写如下语句：

```
ptr = &a;
```

其中“&”为地址运算符，表示取右侧向量的内存地址。对于这条语句，我们可以说ptr指向了a。ptr和&a的区别在于ptr是变量，而&a是常量。

要创建指针变量，先要声明指针变量的类型。假设想要把ptr声明为存储int类型变量地址的指针，就需要用到间接运算符“*”。

假设已知ptr指向b，如：

```
ptr = &b;
```

然后使用间接运算符*找出存储b中的值，如：

```
val = *ptr;
```

二者相结合相当于下面的语句：

```
val = b;
```

声明指针时，必须指定指针所指向变量的类型。比如：

```
int * pi; //pi是指向int类型变量的指针
char * ch; //ch是指向char类型变量的指针
```

类型说明符表明了指针所指向对象的类型，星号（*）表明声明的变量是一个指针。