LECTURE 6 期中复习

工学院18-19学年秋季学期计算概论（邓习峰班）课后辅导

讲师：陈婉雯

日期：2018/11/10

提纲

- 程序：数据+指令
- 数据
  - 数据类型：基本数据类型、拓展类型（结构体）
  - 常量/变量
  - 读取：输入、输出
  - 存储：数组、指针
- 指令
  - 表达式和语句
  - 程序结构
  - 函数

数据类型

- 存储单位：位（0/1）、字节（8位）、字（32/64位）
- 存储长度：
  - 整型（4字节）字符型（1字节）双精度浮点数（8字节）
- 强制数据类型转换
  - (类型说明符)(表达式)
  - 注意运算优先级
- 自动类型转换规则
  - 浮点数赋给整型，该浮点数小数被舍去
  - 整数赋给浮点型，数值不变，但是被存储到相应的浮点型变量中

整型

- 不同的进制
  - 默认为10进制，10，20
  - 以0开头为8进制，045，021
  - 以0x开头为16进制，0x21458adf
- 长整型 long int、无符号整型 unsigned int、短整型 short int
- 2 ≤ short ≤ int ≤ long
- 输出控制符
  - %hd： short int 类型
  - %d： int 类型
  - %ld： long int 类型
  - %u：无符号整型

字符型

- 字符型：单引号表示 ‘A’
  - 字符串：双引号 “A”（实际存储：”A\0”）
  - 字符串实际是字符数组
- ASCII数值、大小写字母转换
  - ‘0’=48, ‘A’=65, ‘a’=97
  - ’A’+32=‘a’

表2-4 转义字符

转义字符	意 义	ASCII码值（十进制）
\a	响铃(BEL)	007
\b	退格(BS)	008
\f	换页(FF)	012
\n	换行(LF)	010
\r	回车(CR)	013
\t	水平制表(HT)	009
\v	垂直制表(VT)	011
\\	反斜杠	092
\?	问号字符	063
\'	单引号字符	039
\"	双引号字符	034
\0	空字符(NULL)	000
\ddd	任意字符	三位八进制
\xhh	任意字符	二位十六进制

常量：不能改变的量

- 整型常量：0,100,1000
- 实型常量：
  - 十进制小数：0.123 （小数点两边有一个为0的话可以不写）
  - 指数：12.34e3（代表12.34*10^3）
  - e或E代表以10为底的指数；e或E前必须有数字，后面必须为整数
- 字符常量：用单引号表示
  - ‘A’, ‘o’,...
- 字符串常量：用双引号表示
  - “Hello, world!\n”
- 符号常量：#define PI 3.14156
- const关键字

变量：可以改变的量

- 变量是程序可操作的存储区的名称
- C 中每个变量都有特定的类型，类型决定了变量存储的大小和布局，该范围内的值都可以存储在内存中
- 运算符可应用于变量上
- 变量的名称可以由字母、数字和下划线字符组成。它必须以字母或下划线开头
- 大写字母和小写字母是不同的，因为 C 是大小写敏感的
- 先声明（类型、标识符），初始化，再使用

运算符

算术运算符

下表显示了 C 语言支持的所有算术运算符。假设变量 A 的值为 10，变量 B 的值为 20，则：

运算符	描述	实例
+	把两个操作数相加	A + B 将得到 30
-	从第一个操作数中减去第二个操作数	A - B 将得到 -10
*	把两个操作数相乘	A * B 将得到 200
/	分子除以分母	B / A 将得到 2
%	取模运算符，整除后的余数	B % A 将得到 0
++	自增运算符，整数值增加 1	A++ 将得到 11
--	自减运算符，整数值减少 1	A-- 将得到 9

关系运算符

下表显示了 C 语言支持的所有关系运算符。假设变量 **A** 的值为 10，变量 **B** 的值为 20，则：

运算符	描述	实例
==	检查两个操作数的值是否相等，如果相等则条件为真。	(A == B) 不为真。
!=	检查两个操作数的值是否相等，如果不相等则条件为真。	(A != B) 为真。
>	检查左操作数的值是否大于右操作数的值，如果是则条件为真。	(A > B) 不为真。
<	检查左操作数的值是否小于右操作数的值，如果是则条件为真。	(A < B) 为真。
>=	检查左操作数的值是否大于或等于右操作数的值，如果是则条件为真。	(A >= B) 不为真。
<=	检查左操作数的值是否小于或等于右操作数的值，如果是则条件为真。	(A <= B) 为真。

逻辑运算符

下表显示了 C 语言支持的所有关系逻辑运算符。假设变量 A 的值为 1，变量 B 的值为 0，则：

运算符	描述	实例
&&	称为逻辑与运算符。如果两个操作数都非零，则条件为真。	(A && B) 为假。
||	称为逻辑或运算符。如果两个操作数中有任意一个非零，则条件为真。	(A || B) 为真。
!	称为逻辑非运算符。用来逆转操作数的逻辑状态。如果条件为真则逻辑非运算符将使其为假。	!(A && B) 为真。

位运算符

位运算符作用于位，并逐位执行操作。&、 | 和 ^ 的真值表如下所示：

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

杂项运算符 ↦ sizeof & 三元

下表列出了 C 语言支持的其他一些重要的运算符，包括 **sizeof** 和 **? :**。

运算符	描述	实例
sizeof()	返回变量的大小。	sizeof(a) 将返回 4，其中 a 是整数。
&	返回变量的地址。	&a; 将给出变量的实际地址。
*	指向一个变量。	*a; 将指向一个变量。
? :	条件表达式	如果条件为真 ? 则值为 X : 否则值为 Y

赋值运算符

下表列出了 C 语言支持的赋值运算符：

运算符	描述	实例
=	简单的赋值运算符，把右边操作数的值赋给左边操作数	C = A + B 将把 A + B 的值赋给 C
+=	加且赋值运算符，把右边操作数加上左边操作数的结果赋值给左边操作数	C += A 相当于 C = C + A
-=	减且赋值运算符，把左边操作数减去右边操作数的结果赋值给左边操作数	C -= A 相当于 C = C - A
*=	乘且赋值运算符，把右边操作数乘以左边操作数的结果赋值给左边操作数	C *= A 相当于 C = C * A
/=	除且赋值运算符，把左边操作数除以右边操作数的结果赋值给左边操作数	C /= A 相当于 C = C / A
%=	求模且赋值运算符，求两个操作数的模赋值给左边操作数	C %= A 相当于 C = C % A
<<=	左移且赋值运算符	C <<= 2 等同于 C = C << 2
>>=	右移且赋值运算符	C >>= 2 等同于 C = C >> 2
&=	按位与且赋值运算符	C &= 2 等同于 C = C & 2
^=	按位异或且赋值运算符	C ^= 2 等同于 C = C ^ 2
|=	按位或且赋值运算符	C |= 2 等同于 C = C | 2

类别	运算符	结合性
后缀	() [] -> . ++ - -	从左到右
一元	+ - ! ~ ++ - - (type)* & sizeof	从右到左
乘除	* / %	从左到右
加减	+ -	从左到右
移位	<< >>	从左到右
关系	< <= > >=	从左到右
相等	== !=	从左到右
位与 AND	&	从左到右
位异或 XOR	^	从左到右
位或 OR	|	从左到右
逻辑与 AND	&&	从左到右
逻辑或 OR	||	从左到右
条件	?:	从右到左
赋值	= += -= *= /= %=>>= <<= &= ^= |=	从右到左
逗号	,	从左到右

表达式

- An expression is a sequence of operators and operands that specifies computation of a value
- 算术表达式
  - 注意除法、求余
- 赋值表达式：左值（不能是常量）、右值
- 自增/自减表达式
  - ++在前先加后用，++在后先用后加
- 逗号表达式
  - 最右边的值是表达式的值
- ……

函数：一组一起执行一个任务的语句

- 主函数main()
  - 只能有一个，程序从main()开始运行
- 函数定义
  - 返回类型、函数名称、参数、函数主体
- 函数声明
  - 告诉编译器函数名称及如何调用函数，函数的实际主体单独定义
- 函数参数：实际参数、形式参数、参数传递

调用类型	描述
传值调用	该方法把参数的实际值复制给函数的形式参数。在这种情况下，修改函数内的形式参数不会影响实际参数。
引用调用	通过指针传递方式，形参为指向实参地址的指针，当对形参的指向操作时，就相当于对实参本身进行的操作。

默认情况下，C 使用**传值调用**来传递参数。一般来说，这意味着函数内的代码不能改变用于调用函数的实际参数。

输入输出

- printf(格式控制，输出表列)
- scanf(格式控制，地址表列) &：取地址符
- 函数库：<stdio.h>
- 输入格式要和格式控制严格相同
- 格式控制：格式声明+普通字符

表 4.3 转换说明符及作为结果的打印输出

转换说明	输出
%a	浮点数、十六进制数字和 p-记数法（C99）
%A	浮点数、十六进制数字和 P-记数法（C99）
%c	一个字符
%d	有符号十进制整数
%e	浮点数、e-记数法
%E	浮点数、E-记数法
%f	浮点数、十进制记数法
%g	根据数值不同自动选择%f 或%e。%e 格式在指数小于-4 或者大于等于精度时使用
%G	根据数值不同自动选择%f 或%E。%E 格式在指数小于-4 或者大于等于精度时使用
%i	有符号十进制整数（与%d 相同）
%o	无符号八进制整数
%p	指针
%s	字符串
%u	无符号十进制整数
%x	使用十六进制数字 0f 的无符号十六进制整数
%X	使用十六进制数字 0F 的无符号十六进制整数
%%	打印一个百分号

表 4.4 printf()的修饰符

修饰符	含义
标记	表 4.5 描述了 5 种标记（-、+、空格、#和 0），可以不使用标记或使用多个标记 示例："%-10d"
数字	最小字段宽度 如果该字段不能容纳待打印的数字或字符串，系统会使用更宽的字段 示例："%4d"
.数字	精度 对于%e、%E 和%f 转换，表示小数点右边数字的位数 对于%g 和%G 转换，表示有效数字最大位数 对于%s 转换，表示待打印字符的最大数量 对于整型转换，表示待打印数字的最小位数 如有必要，使用前导 0 来达到这个位数 只使用.表示其后跟随一个 0，所以%.f 和%.0f 相同 示例："%5.2f"打印一个浮点数，字段宽度为 5 字符，其中小数点后有两位数字

表 4.5 printf() 中的标记

标记	含义
-	待打印项左对齐。即，从字段的左侧开始打印该项项 示例："%-20s"
+	有符号值若为正，则在值前面显示加号；若为负，则在值前面显示减号 示例："%+6.2f"
空格	有符号值若为正，则在值前面显示前导空格（不显示任何符号）；若为负，则在值前面显示减号 +标记覆盖一个空格 示例："%6.2f"
#	把结果转换为另一种形式。如果是%o 格式，则以 0 开始；如果是%x 或%X 格式，则以 0x 或 0X 开始；对于所有的浮点格式，#保证了即使后面没有任何数字，也打印一个小数点字符。对于%g 和%G 格式，#防止结果后面的 0 被删除 示例："%#o"、"%#8.0f"、"%+#10.3e"
0	对于数值格式，用前导 0 代替空格填充字段宽度。对于整数格式，如果出现-标记或指定精度，则忽略该标记

示例

```
int main(void)
{
    const double RENT = 3852.99;  // const 变量

    printf("*%f*\n", RENT);
    printf("*%e*\n", RENT);
    printf("*%4.2f*\n", RENT);
    printf("*%3.1f*\n", RENT);
    printf("*%10.3f*\n", RENT);
    printf("*%10.3E*\n", RENT);
    printf("*%+4.2f*\n", RENT);
    printf("*%010.2f*\n", RENT);

    return 0;
}
```

```
*3852.990000*
*3.852990e+03*
*3852.99*
*3853.0*
*  3852.990*
* 3.853E+03*
*+3852.99*
*0003852.99*
```

逻辑结构

- 顺序结构
- 选择结构
  - if
  - switch 注意要break
    - 在case后的各常量表达式的值不能相同
    - 在case后，允许有多个语句，可以不用{}括起来
    - 各case和default子句的先后顺序可以变动，而不会影响程序执行结果
    - default子句可以省略不用
- 循环结构
  - while
  - do-while
  - for
- 注意continue、break

指针

- 指针：值为内存地址的变量
- 取地址符：& 求变量的内存地址，后面跟一个变量名
- 间接运算符：* 取内存地址内的值，后面跟一个地址/指针名
- 数组名：数组首元素的地址（数组名不能++）
- 指针加1：递增至它所指向类型的大小
- 数组的两种表示法：
  - int days[MONTHS];
  - 指针表示法：*(days+index)　　days+index
  - 数组表示法：days[index]　　&(days[index])

```
*(dates + 2)   // dates 第 3 个元素的值
*dates + 2     // dates 第 1 个元素的值加 2
```

指针与内存分配

- int value;
  - 系统自动预留相应的内存空间存储value
- int *add;
  - 我们只知道add会是一个地址（无符号整型），但系统不会自动预留内存空间，add也没有指向特定的内存位置
- 函数库<stdlib.h>
- malloc(size):
  - 找到合适的、大小为size字节的空闲内存块
  - 返回动态分配内存块的首字节地址
  - 要把这个地址赋给指针变量，通过指针访问
  - 要记得强制类型转换
- free()：把内存归还到内存池中，使得内存可以重复使用

const和指针

- 常量指针
  - int const* p; const int* p;
  - 能够把const/非const值赋给常量指针，但不能把const值赋给普通指针
  - 常量指针指向的对象不能通过这个指针来修改，可是仍然可以通过原来的声明修改
  - 常量指针可以被赋值为变量的地址，之所以叫常量指针，是限制了通过这个指针修改变量的值
  - 指针还可以指向别处，因为指针本身只是个变量，可以指向任意地址

```
double rates[5] = {88.99, 100.12, 59.45, 183.11,
const double locked[4] = {0.08, 0.075, 0.0725,
const double * pc = rates; // 有效
pc = locked;                //有效
pc = &rates[3];             //有效
```

指向 const 的指针不能用于改变值。考虑下面的代码：

```
double rates[5] = {88.99, 100.12, 59.45, 183.11, 340.5};
const double * pd = rates;      // pd 指向数组的首元素
```

第 2 行代码把 pd 指向的 double 类型的值声明为 const，这表明不能使用 pd 来更改它

```
*pd = 29.89;       // 不允许
pd[2] = 222.22;    //不允许
rates[0] = 99.99; // 允许，因为 rates 未被 const 限定
```

const和指针

- 指针常量
  - int* const p;
  - 指针指向的地址不能改变，但指针地址内存的值可以改变

```
double rates[5] = {88.99, 100.12, 59.45, 183.11, 340.5};
double * const pc = rates; // pc指向数组的开始
pc = &rates[2];            // 不允许，因为该指针不能指向别处
*pc = 92.99;               // 没问题 -- 更改rates[0]的值
```

- 指向常量的常指针
  - const int * const p;
  - 既不能改变指针指向的地址，也不能改变地址内存里面的值

```
double rates[5] = {88.99, 100.12, 59.45, 183.11, 340.5};
const double * const pc = rates;
pc = &rates[2];    //不允许
*pc = 92.99;       //不允许
```

函数指针

- 指向函数的指针：通常用作另外一个函数的参数
- 声明：函数返回类型和形参类型
  - 返回值类型 (* 指针变量名) ([形参列表]);
  - int func(int x); /* 声明一个函数 */
  - int (*f) (int); /* 声明一个函数指针 */
  - f=func; /* 将func函数的首地址赋给指针f */
- 调用：
  - a=f(x); 等价于 a=func(x);

结构体

- 定义结构变量：创建结构体，分配存储空间
- struct stuff Alex;
- 初始化结构：
  - 直接访问结构成员
  - struct stuff Alex = {20,abcd,4.5};
- 访问结构成员：
  - Alex.number、Alex.code
- 假如创建的是结构体指针：
  - struct stuff *Alex;
  - Alex = (struct stuff *) malloc(sizeof(struct stuff));
- 访问方式变为：
  - (*Alex).number、Alex->number

链表

- 链表的定义
- 链表的基本操作
  - 创建链表
  - 往链表里添加、删除成员
  - 显示（遍历）链表
  - 删除（释放）链表