C++基础入门

1 C++初识

1.1 注释

- **作用**：在代码中加一些说明和解释，方便自己或其他程序员程序员阅读代码
- 两种格式与c相同：`/*……*/`以及`//`

1.2 变量与常量

- 变量：给一段指定的内存空间起名，方便操作这段内存
  - 语法：`数据类型 变量名 = 初始值;`
  - 注意：C++在创建变量时，必须给变量一个初始值，否则会报错
- 常量：用于记录程序中不可更改的数据
  - C++定义常量两种方式
    - **#define** 宏常量：`#define 常量名 常量值`
      - 通常在文件上方定义，表示一个常量
    - **const**修饰的变量`const 数据类型 常量名 = 常量值`
      - 通常在变量定义前加关键字const，修饰该变量为常量，不可修改

1.3 关键字

- 关键字是C++中预先保留的单词（标识符）
  - **在定义变量或者常量时候，不要用关键字**
  - C++关键字如下：

asm	**do**	**if**	**return**	**typedef**
auto	double	inline	short	typeid
bool	dynamic_cast	int	signed	typename
break	else	long	sizeof	union
case	enum	mutable	static	unsigned
catch	explicit	namespace	static_cast	using
char	export	new	struct	virtual
class	extern	operator	switch	void
const	false	private	template	volatile
const_cast	float	protected	this	wchar_t
continue	for	public	throw	while
default	friend	register	true	
delete	goto	reinterpret_cast	try	

delete **asm**	goto **do**	reinterpret_cast **if**	try **return**	**typedef**

> 提示：在给变量或者常量起名称时候，不要用C++得关键字，否则会产生歧义

1.4 标识符命名规则

- C++规定给标识符（变量、常量）命名时，有一套自己的规则
  - 标识符不能是关键字
  - 标识符只能由字母、数字、下划线组成
  - 第一个字符必须为字母或下划线
  - 标识符中字母区分大小写

> 建议：给标识符命名时，争取做到见名知意的效果，方便自己和他人的阅读

2 数据类型

C++规定在创建一个变量或者常量时，必须要指定出相应的数据类型，否则无法给变量分配内存

2.1 整型

- **作用**：整型变量表示的是整数类型的数据
- C++中能够表示整型的类型有以下几种方式，**区别在于所占内存空间不同**：

数据类型	占用空间	取值范围
short(短整型)	2字节	(-2^15 ~ 2^15-1)
int(整型)	4字节	(-2^31 ~ 2^31-1)
long(长整形)	Windows为4字节，Linux为4字节(32位)，8字节(64位)	(-2^31 ~ 2^31-1)
long long(长长整形)	8字节	(-2^63 ~ 2^63-1)

2.2 sizeof关键字

- **作用：**利用sizeof关键字可以统计数据类型所占内存大小
- **语法：** `sizeof( 数据类型 / 变量)`

> **整型结论**：short < int <= long <= long long

2.3 实型（浮点型）

- **作用**：用于表示小数

- 浮点型变量分为两种：
  - 单精度float
  - 双精度double
    - 两者的**区别**在于表示的有效数字范围不同。

数据类型	占用空间	有效数字范围
float	4字节	7位有效数字
double	8字节	15～16位有效数字

2.4 字符型

- **作用：**字符型变量用于显示单个字符
- **语法：**`char ch = 'a';`

> 注意1：在显示字符型变量时，用单引号将字符括起来，不要用双引号

> 注意2：单引号内只能有一个字符，不可以是字符串

- C和C++中字符型变量只占用1个字节。
- 字符型变量并不是把字符本身放到内存中存储，而是将对应的ASCII编码放入到存储单元
- ASCII码表格：

ASCII值	控制字符	ASCII值	字符	ASCII值	字符	ASCII值	字符
0	NUT	32	(space)	64	@	96	、
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	,	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k

ASCII值	控制字符	ASCII值	字符	ASCII值	字符	ASCII值	字符
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DCI	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	TB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	/	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	`
31	US	63	?	95	_	127	DEL

- ASCII 码大致由以下**两部分组**成：
  - ASCII 非打印控制字符： ASCII 表上的数字 **0-31** 分配给了控制字符，用于控制像打印机等一些外围设备。
  - ASCII 打印字符：数字 **32-126** 分配给了能在键盘上找到的字符，当查看或打印文档时就会出现。

2.5 转义字符

- **作用：**用于表示一些不能显示出来的ASCII字符
- 现阶段我们常用的转义字符有：`\n  \\  \t`

转义字符	含义	ASCII码值（十进制）
\a	警报	007
\b	退格(BS)，将当前位置移到前一列	008
\f	换页(FF)，将当前位置移到下页开头	012
\n	**换行(LF)，将当前位置移到下一行开头**	**010**
\r	回车(CR)，将当前位置移到本行开头	013
\t	**水平制表(HT) （跳到下一个TAB位置）**	**009**
\v	垂直制表(VT)	011
****	**代表一个反斜线字符""**	**092**
'	代表一个单引号（撇号）字符	039
"	代表一个双引号字符	034
\?	代表一个问号	063
\0	数字0	000
\ddd	8进制转义字符，d范围0~7	3位8进制
\xhh	16进制转义字符，h范围0~9，a~f，A~F	3位16进制

2.6 字符串型

- **作用**：用于表示一串字符
- **两种风格**
  - **C风格字符串**： `char 变量名[] = "字符串值"`
  - **C++风格字符串**： `string 变量名 = "字符串值"`

```
int main() {

    char str1[] = "hello world";//C风格
    cout << str1 << endl;

    string str = "hello world";//C++风格
    cout << str << endl;

    system("pause");

    return 0;
}
```

> 注意：C风格的字符串要用双引号括起来

> 注意：C++风格字符串，需要加入头文件`#include<string>`

2.7 布尔类型 bool

- **作用：**布尔数据类型代表真或假的值
- bool类型只有两个值：
  - true --- 真（本质是1）
  - false --- 假（本质是0）
- **bool类型占1个字节大小**

2.8 数据的输入输出

- **作用：用于从键盘获取数据或者相屏幕打印**
- **关键字：**cin、cout
- **语法：** `cin >> 变量 或者 cout << 待显示的数据`

3 运算符

3.1 算数运算符

- **作用**：用于处理四则运算
- 算术运算符包括以下符号：

运算符	术语	示例	结果
+	正号	+3	3
-	负号	-3	-3
+	加	10 + 5	15
-	减	10 - 5	5
*	乘	10 * 5	50
/	除	10 / 5	2
%	取模(取余)	10 % 3	1
++	前置递增	a=2; b=++a;	a=3; b=3;
++	后置递增	a=2; b=a++;	a=3; b=2;
--	前置递减	a=2; b=--a;	a=1; b=1;
--	后置递减	a=2; b=a--;	a=1; b=2;

> 总结：整数与整数进行运算结果也只能是整数

> 总结：在除法运算中，除数不能为0

> 总结：只有整型变量可以进行取模运算

> 总结：前置递增先对变量进行++，再计算表达式，后置递增相反

3.2 赋值运算符

- **作用**：用于将表达式的值赋给变量
- 赋值运算符包括以下几个符号：

运算符	术语	示例	结果
=	赋值	a=2; b=3;	a=2; b=3;
+=	加等于	a=0; a+=2;	a=2;
-=	减等于	a=5; a-=3;	a=2;
*=	乘等于	a=2; a*=2;	a=4;
/=	除等于	a=4; a/=2;	a=2;
%=	模等于	a=3; a%2;	a=1;

3.3 比较运算符

- **作用**：用于表达式的比较，并返回一个真值或假值
- 比较运算符有以下符号：

运算符	术语	示例	结果
==	相等于	4 == 3	0
!=	不等于	4 != 3	1
<	小于	4 < 3	0
>	大于	4 > 3	1
<=	小于等于	4 <= 3	0
>=	大于等于	4 >= 1	1

> 注意：C和C++ 语言的比较运算中， **“真”用数字“1”来表示， “假”用数字“0”来表示**

3.4 逻辑运算符

- **作用**：用于根据表达式的值返回真值或假值
- 逻辑运算符有以下符号：

运算符	术语	示例	结果
!	非	!a	如果a为假，则!a为真； 如果a为真，则!a为假。
&&	与	a && b	如果a和b都为真，则结果为真，否则为假。
||	或	a || b	如果a和b有一个为真，则结果为真，二者都为假时，结果为假。

4 程序流程结构

- C/C++支持最基本的三种程序运行结构：**顺序结构、选择结构、循环结构**
  - 顺序结构：程序按顺序执行，不发生跳转
  - 选择结构：依据条件是否满足，有选择的执行相应功能
  - 循环结构：依据条件是否满足，循环多次执行某段代码

4.1 选择结构

4.1.1 if语句

- **作用**：执行满足条件的语句
- if语句的三种形式
  - 单行格式if语句
  - 多行格式if语句
  - 多条件的if语句
- 结构与C一模一样

示例：

```
if(){//单行格式if语句

}

if(){//多行格式if语句

}else{

}

if(){//多条件的if语句

}else if(){

}else{

}
```

- **嵌套if语句**：在if语句中，可以嵌套使用if语句，达到更精确的条件判断

4.1.2 三目运算符

- **作用**： 通过三目运算符实现简单的判断
- **语法**：`表达式1 ? 表达式2 : 表达式3`
- **解释**：
  - 如果表达式1的值为真，执行表达式2，并返回表达式2的结果；
  - 如果表达式1的值为假，执行表达式3，并返回表达式3的结果

> 总结：和if语句比较，三目运算符优点是短小整洁，缺点是如果用嵌套，结构不清晰

4.1.3 switch语句

- **作用**：执行多条件分支语句
- **语法**：

```
switch(表达式)

{

    case 结果1：执行语句;break;

    case 结果2：执行语句;break;

    ...

    default:执行语句;break;

}
```

> 注意1：`switch`语句中表达式类型只能是整型或者字符型

> 注意2：`case`里如果没有break，那么程序会一直向下执行

> 总结：与if语句比，对于多条件判断时，switch的结构清晰，执行效率高，缺点是switch不可以判断区间

4.2 循环结构

4.2.1 while循环语句

- **作用**：满足循环条件，执行循环语句
- **语法**：`while(循环条件){ 循环语句 }`
- **解释**：只要循环条件的结果为真，就执行循环语句

4.2.2 do...while循环语句

- **作用**： 满足循环条件，执行循环语句
- **语法**： `do{ 循环语句 } while(循环条件);`
- **注意**：与while的区别在于do...while会先执行一次循环语句，再判断循环条件

4.2.3 for循环语句

- **作用**： 满足循环条件，执行循环语句
- **语法**：`for(起始表达式;条件表达式;末尾循环体) { 循环语句; }`

> 注意：for循环中的表达式，要用分号进行分隔

> 总结：while , do...while, for都是开发中常用的循环语句，for循环结构比较清晰，比较常用

4.3 跳转语句

4.3.1 break语句

- **作用:** 用于跳出==选择结构==或者==循环结构==
- break使用的时机：
    - 出现在switch条件语句中，作用是终止case并跳出switch

- 出现在循环语句中，作用是跳出当前的循环语句
- 出现在嵌套循环中，跳出最近的内层循环语句

4.3.2 continue语句

- **作用：**在==循环语句==中，跳过本次循环中余下尚未执行的语句，继续执行下一次循环

> 注意：continue并没有使整个循环终止，而break会跳出循环

4.3.3 goto语句

- **作用：**可以无条件跳转语句
- **语法：** `goto 标记;`
- **解释：**如果标记的名称存在，执行到goto语句时，会跳转到标记的位置

> 注意：在程序中不建议使用goto语句，以免造成程序流程混乱

5 数组

5.1 概述

- 所谓数组，就是一个集合，里面存放了相同类型的数据元素
- **特点1：**数组中的每个数据元素都是相同的数据类型
- **特点2：**数组是由连续的内存位置组成的

> 总结1：数组名的命名规范与变量名命名规范一致，不要和变量重名

> 总结2：数组中下标是从0开始索引

5.2.2 一维数组数组名

- 一维数组名称的**用途**：
  - 可以统计整个数组在内存中的长度（通过`sizeof（数组名）`）
  - 可以获取数组在内存中的首地址

> 注意：数组名是常量，不可以赋值

> 总结1：直接打印数组名，可以查看数组所占内存的首地址

> 总结2：对数组名进行`sizeof`，可以获取整个数组占内存空间的大小

5.3.1 二维数组定义方式

1. 二维数组定义的四种方式：
   1. `数据类型  数组名[ 行数 ][ 列数 ];`
   2. `数据类型  数组名[ 行数 ][ 列数 ] = { {数据1，数据2 } ，{数据3，数据4 } };`
   3. `数据类型  数组名[ 行数 ][ 列数 ] = { 数据1，数据2，数据3，数据4};`
   4. `数据类型  数组名[  ][ 列数 ] = { 数据1，数据2，数据3，数据4};`

> 建议：以上4种定义方式，利用第二种更加直观，提高代码的可读性

> 总结：在定义二维数组时，如果初始化了数据，可以省略行数

5.3.2 二维数组数组名

- 查看二维数组所占内存空间

- 获取二维数组首地址

> 总结1：二维数组名就是这个数组的首地址

> 总结2：对二维数组名进行`sizeof`时，可以获取整个二维数组占用的内存空间大小

6 函数

6.1 概述

作用：将一段经常使用的代码封装起来，减少重复代码

一个较大的程序，一般分为若干个程序块，每个模块实现特定的功能。

6.2 函数的定义

- 函数的定义一般主要有5个步骤：

  1、返回值类型

  2、函数名

  3、参数表列

  4、函数体语句

  5、return 表达式

- **语法：**

```
返回值类型 函数名 （参数列表）
{

        函数体语句

        return表达式

}
```

- 返回值类型 ： 一个函数可以返回一个值
- 函数名：给函数起个名称
- 参数列表：使用该函数时传入的数据
- 函数体语句：花括号内的代码，函数内需要执行的语句
- return表达式： 和返回值类型挂钩，函数执行完后，返回相应的数据

6.3 函数的调用

- **功能：**使用定义好的函数
- **语法：**`函数名（参数）`

> 总结：函数定义里小括号内称为形参，函数调用时传入的参数称为实参

6.4 值传递

- 所谓值传递，就是函数调用时实参将数值传入给形参
- 值传递时，如果形参发生，并不会影响实参

6.5 函数的常见样式

- 常见的函数样式有4种
    1. 无参无返
    2. 有参无返
    3. 无参有返
    4. 有参有返

6.6 函数的声明

- **作用：** 告诉编译器函数名称及如何调用函数。函数的实际主体可以单独定义。
- 函数的**声明可以多次**，但是函数的**定义只能有一次**

6.7 函数的分文件编写

- **作用：**让代码结构更加清晰
- 函数分文件编写一般有4个步骤
    1. 创建后缀名为`.h`的头文件
    2. 创建后缀名为`.cpp`的源文件
    3. 在头文件中写函数的声明
    4. 在源文件中写函数的定义

7 指针

7.1 指针的基本概念

- **指针的作用：** 可以通过指针间接访问内存
    - 内存编号是从0开始记录的，一般用十六进制数字表示
    - 可以利用指针变量保存地址

7.2 指针变量的定义和使用

- 指针变量定义语法：`数据类型 * 变量名;`
- 指针变量和普通变量的区别
    - 普通变量存放的是数据,指针变量存放的是地址
    - 指针变量可以通过" * "操作符，操作指针变量指向的内存空间，这个过程称为解引用

> 总结1： 我们可以通过 & 符号 获取变量的地址

> 总结2：利用指针可以记录地址

> 总结3：对指针变量解引用，可以操作指针指向的内存

7.3 指针所占内存空间

- 提问：指针也是种数据类型，那么这种数据类型占用多少内存空间?
- 总结：所有指针类型在32位操作系统下是4个字节，64位下是8个字节

7.4 空指针和野指针

- **空指针**：指针变量指向内存中编号为0的空间

    - **用途：**初始化指针变量
    - **注意：**空指针指向的内存是不可以访问的
- **野指针**：指针变量指向非法的内存空间

> 总结：空指针和野指针都不是我们申请的空间，因此不要访问

7.5 const修饰指针

- const修饰指针有三种情况
    1. const修饰指针 --- 常量指针
    2. const修饰常量 --- 指针常量
    3. const即修饰指针，又修饰常量
- `const int *p = &a;`
    - 常量指针
        - 指针的指向可以修改，但是指针指向的值不可以改

```
*p = 20;//错误，指针指向的值不可以改
p = &b; //正确，指针指向可以修改
```

- `int * const p = &a;`
    - 指针常量
        - 指针的指向不可以改，但是指针指向的值可以改

```
*p = 20;//正确，指向的值可以改
p = &b; //错误，指针指向不可以改
```

- `const int * const p = &a;`
    - 指针的指向和指针指向的值都不可以改

```
*p = 20;//错误，指向的值不可以改
p = &b; //错误，指针指向不可以改
```

> const修饰谁，谁就不能改变，*前值不变，p前指不变

7.6 指针和数组

- **作用：**利用指针访问数组中元素

7.7 指针和函数

- **作用：**利用指针作函数参数，可以修改实参的值

> 总结：如果不想修改实参，就用值传递，如果想修改实参，就用地址传递

8 结构体

8.1 结构体基本概念

- 结构体属于用户自定义的数据类型，允许用户存储不同的数据类型

8.2 结构体定义和使用

- **语法：** `struct 结构体名 { 结构体成员列表 };`
- 通过结构体创建变量的方式有三种：
  - struct 结构体名 变量名
  - struct 结构体名 变量名 = { 成员1值， 成员2值...}
  - 定义结构体时顺便创建变量

> 总结1：定义结构体时的关键字是struct，不可省略

> 总结2：创建结构体变量时，关键字struct可以省略

> 总结3：结构体变量利用操作符 "." 访问成员

8.3 结构体数组

- **作用：** 将自定义的结构体放入到数组中方便维护
- **语法：** `struct 结构体名 数组名[元素个数] = { {} , {} , ... {} }`

8.4 结构体指针

- **作用：** 通过指针访问结构体中的成员
  - 利用操作符 `->` 可以通过结构体指针访问结构体属性

8.5 结构体嵌套结构体

- **作用：** 结构体中的成员可以是另一个结构体
  - **例如：** 每个老师辅导一个学员，一个老师的结构体中，记录一个学生的结构体

> **总结：** 在结构体中可以定义另一个结构体作为成员，用来解决实际问题

8.6 结构体做函数参数

- **作用：** 将结构体作为参数向函数中传递
- 传递方式有两种：
  - 值传递
  - 地址传递

> 总结：如果不想修改主函数中的数据，用值传递，反之用地址传递

8.7 结构体中 const使用场景

- **作用：** 用const来防止误操作
- 示例

示例：

```
//const使用场景
//此处const修饰的是变量，属于指针常量，因此不允许修改指针所指向的数据的值
//此处使用指针的话就不会用形参申请额外的空间
void printStudent(const student *stu) //加const防止函数体中的误操作
{
    //stu->age = 100; //操作失败，因为加了const修饰
    cout << "姓名：" << stu->name << " 年龄：" << stu->age << " 分数：" << stu->score << endl;

}
```