

C++ 再温习

ZeitHaum

2023 年 2 月 20 日

目录

1 语法基础	1
1.1 include	1
1.2 main	1
1.3 标识符	1
1.4 语句	1
1.5 终止符	1
1.6 头文件名	2
1.7 名称空间	2
1.8 define	2
1.9 测试代码	3
2 输入输出	4
2.1 性能	4
2.2 占位符	5
2.3 测试代码	5
3 变量	6
3.1 内存空间	6
3.2 整数	7
3.3 bool	7
3.4 char	7
3.5 void	8
3.6 const	8
3.7 浮点数	8
3.7.1 基本概述	8
3.7.2 浮点常量	8
3.7.3 浮点数输出	8
3.8 类型转换	9
3.8.1 隐式类型转换	9
3.9 强制类型转换	9
3.10 auto	10
3.11 测试代码	10

4 运算	11
4.1 自增自减运算符	11
4.2 逗号运算符	11
4.3 赋值语句	11
4.4 运算优先级	11
4.5 结合性	11
4.5.1 a[i++] 和 a[++i]	12
4.6 除法的类型转换	12
4.7 测试代码	12
5 流程控制语句	13
5.1 if-else 控制语句	13
5.1.1 if	13
5.1.2 else	13
5.1.3 else if	13
5.2 逻辑运算	13
5.2.1 &&、 、!	13
5.2.2 细节	14
5.3 ctype 字符函数库	14
5.4 ?:	15
5.5 switch-case 语句	15
5.6 break 和 continue	16
5.7 测试代码	16
6 循环	17
6.1 for	17
6.1.1 基本结构	17
6.1.2 和逗号运算符使用的技巧	17
6.1.3 基于范围的循环	18
6.2 while	18
6.3 do-while	18
6.4 测试代码	18
7 C++ 高级数据类型	19

7.1	一维数组	19
7.1.1	基本概述	19
7.1.2	数组越界	19
7.1.3	数组初始化	19
7.1.4	C++11 的数组初始化	20
7.2	多维数组	20
7.3	字符串	20
7.3.1	C 风格字符串	20
7.3.2	C++ 风格字符串	21
7.3.3	字符串 I/O: 读取一行	21
7.3.4	其他形式的字符串	22
7.4	结构体	22
7.4.1	结构体初始化	23
7.4.2	结构数组	23
7.4.3	结构中的位字段	23
7.5	共用体	24
7.6	枚举	24
7.6.1	声明和赋值	24
7.6.2	指定枚举的值	24
7.7	测试代码	24
	参考资料	27

工欲善其事，必先利其器。

1 语法基础

1.1 include

将文件的内容复制到该处，include 引入的文件也称为头文件。**增加 include 的内容只会增加编译时间，对运行时间几乎没有影响。**

1.2 main

供系统或外部程序调用，return 0 表示运行成功，main 函数默认返回 return 0。

1.3 标识符

作为变量名的一组字符，具备以下特征：

1. 允许出现英文、数字、下划线。
2. 数字不能出现在第一个字符。
3. 不允许和 C++ 关键词同名。C++ 关键字可见于：[C++ 的关键字（保留字）完整介绍](#)

1.4 语句

需要执行的操作。

1.5 终止符

C++ 中即 “;”，其是语句的组成部分。

1.6 头文件名

表 2.1 头文件命名约定			
头文件类型	约定	示例	说明
C++旧式风格	以.h结尾	iostream.h	C++程序可以使用
C旧式风格	以.h结尾	math.h	C、C++程序可以使用
C++新式风格	没有扩展名	iostream	C++程序可以使用，使用 namespace std
转换后的 C	加上前缀 c，没有扩展名	cmath	C++程序可以使用，可以使用不是 C 的特性，如 namespace std

1.7 名称空间

避免重名。使用编译指令语句 using 导入。具备两种格式：

1. using namespace std;(全局)
2. using std::cin;(局部)

1.8 define

也称为宏定义语句，本质是文本替换。可以识别变量。例如：

```
1 #define pi 3.14159
2 #define sum(x,y) (x)+(y)
```

注意由于是文本替换，有可能引起运算顺序上的错误。如 2*sum(1,2) 使用上述文本替换后会错误地先计算乘法。

define 可以递归调用，如：

```
1 #define pi 3.1415926
2 #define calc_circle_area(x) pi*(x)*(x)
```

define 可以通过 # 函数将标识符转化为字符串，或者通过 ## 将两个标识符合并为新标识符。如：

```
1 #define tostring(a) (#a)
2 #define mergestring(a,b) (a##b)
```

调用第一个宏可以得到值为 a 的字面量，调用第二个宏可以得到标识符为 ab 的变量（需要提前定义，否则会报错。）

可以使用 #undef 结束宏定义，在 #undef 后的代码将无法使用对应的宏。

```
1 #undef pi
```

define 的另一大应用是条件编译。首先定义一个没有值的变量，如

```
1 #define DEBUG
```

在代码中可以通过 `#if defined(DEBUG)` 或 `#ifdef DEBUG` 语句对代码进行选择执行。其对应的取反操作为 `#if !defined(DEBUG)` 和 `#ifndef DEBUG`。表示其余情况用 `#else`，结束语句是 `#endif`。

1.9 测试代码

对本部分内容的测试代码：

```
1 #include <iostream>
2 #include <cmath>
3 using namespace std;
4 #define pi 3.1415926
5 #define calc_circle_area(x) pi*(x)*(x)
6 #define toString(a) (#a)
7 #define mergestring(a,b) (a##b)
8 #define DEBUG
9
10 void testfunc(){
11     cout<<M_PI<<endl;
12     cout<<calc_circle_area(3.0)<<endl;
13     int x = 1;
14     int y = 2;
15     int xy = 3;
16     cout<<toString(x)<<endl;
17     cout<<mergestring(x,y)<<endl;
18     #undef pi
19     //cout<<pi<<endl; //未定义语句，将会报错。
20     #if defined(DEBUG)
21         cout<<"Enter DEBUG!"<<endl;
22     #else
23         cout<<"No DEBUG defined."<<endl;
24     #endif
25 }
```

```
26     #ifndef DEBUG
27         cout<<"Donot define DEBUG."<<endl;
28     #endif
29 }
30
31 int main(){
32     // cout<<M_PI;
33     testfunc();
34 }
```

2 输入输出

2.1 性能

std::cin 和 std::cout 相较于 C 语言的 scanf 和 printf 速度较慢。

使得 std::cin 速度较慢的原因之一为兼容 stdio 的开关, 可使用 std::ios::sync_with_stdio(false); 关闭。此时 C++ 的输入输出和 C 语言输入输出解除绑定, 此时不能同时使用 cin 和 scanf 或同时使用 cout 和 printf。

其次可以解除输入流和输出流的绑定。此时如果需要先 cout 再 cin 需要手动 flush。如

```
1 cin.tie(0);
2 cout<<"Hello,C++!";
3 int s;
4 cin>>s;
```

此时执行上述代码就可能会先输入 s 再输出字符串。注意:endl 会强制清空缓存, 因此下述代码不会出现问题:

```
1 cin.tie(0);
2 cout<<"Hello,C++!"<<endl;
3 int t;
4 cin>>t;
```

其余更多优化 (getchar、putchar、fread、fwrite、mmap) 等参见 [OI-wiki-OI 优化](#).

2.2 占位符

如下表 (注意:C++ 没有**%b** 输出二进制的方式。):

符号	含义	符号	含义
%d	十进制整数	%g	小数或者科学计数
%f	浮点数	%i	十进制、八进制、十六进制数
%s	字符串	%o	八进制数
%c	字符	%x	十六进制数
%p	指针	%%	%
%e	科学计数法	%u	无符号数
%[]	正则表达式读取字符集	%n	当前已有的字符数

%n 的用法举例如下:

```
1 int n;
2 scanf("%s%n",&str,&n);
3 cout<<toString(str)<<": "<<str<< " "<<toString(n)<<": "<<n<<"\n";
```

在控制台输入 “hello,world”, 终端输出:

```
str:hello,world n:11
```

%[] 是 C 语言的正则表达式用法, 当遇到第一个不满足 [] 内正则表达式的字符便会停止读取。注意需要导入 stdio 包 (不能同时导入 iostream 包)。一个简单的筛选数字的例子如下:

```
1 char x[80];
2 scanf("%[0-9]",x);
3 printf("%s",x);//注意:x已经是指针, 不能再次取地址。
```

输入 “123456789test”, 其输出只会读取前几个数字:

```
123456789test
123456789
```

2.3 测试代码

```
1 #include <iostream>
2 #include <stdio.h>
3 #define toString(a) #a
4 using namespace std;
5
6 void testfunc(){
7     ios::sync_with_stdio(false);
8     cin.tie(0);
9     cout<<"Hello,C++!";
10    int s;
11    cin>>s;
12    cout<<"Hello,C++!"<<endl;
13    int t;
14    cin>>t;
15    char str[200];
16    int n;
17    scanf("%s%n",&str,&n);
18    cout<<toString(str)<<":"<<str<<" "<<toString(n)<<":"<<n<<"\n";
19    // char x[80];
20    // scanf("%[0-9]",x);
21    // printf("%s",x);//注意:x已经是指针,不能再次取地址。
22 }
23
24 int main(){
25     testfunc();
26 }
```

3 变量

3.1 内存空间

不同类型的变量占据的内存空间不同。几个常见的 2 的幂次:

1. $2^{15} = 32768$

2. $2^{16} = 65536$

3. $2^{31} = 2147483648$

4. $2^{63} \approx 9.2 \times 10^{18}$

使用 `sizeof` 可以查看某个变量所占据的空间 (单位字节), 库 `climits` 可以查看各种类型可以表达的最大值和最小值。

3.2 整数

`short` 是 `short integer` 的简称, `long` 是 `long integer` 的简称。C++ 对不同整数的位数定义如下:

- `short` 至少 16 位;
- `int` 至少与 `short` 一样长;
- `long` 至少 32 位, 且至少与 `int` 一样长;
- `long long` 至少 64 位, 且至少与 `long` 一样长。

C++ 默认将证书字面量当作 `int` 类型。

整数类型的省略和顺序调整:

等价的类型表述

在不引发歧义的情况下, 允许省略部分修饰关键字, 或调整修饰关键字的顺序。这意味着同一类型会存在多种等价表述。

例如 `int`, `signed`, `int signed`, `signed int` 表示同一类型, 而 `unsigned long` 和 `unsigned long int` 表示同一类型。

3.3 bool

`bool` 类型占据 1 字节的空间。

3.4 char

占据 1 字节空间。ASCII 码中转义字符的定义如下:

表 3.2 C++转义序列的编码

字符名称	ASCII 符号	C++代码	十进制 ASCII 码	十六进制 ASCII 码
换行符	NL (LF)	\n	10	0xA
水平制表符	HT	\t	9	0x9
垂直制表符	VT	\v	11	0xB
退格	BS	\b	8	0x8
回车	CR	\r	13	0xD
振铃	BEL	\a	7	0x7
反斜杠	\	\\	92	0x5C
问号	?	\?	63	0x3F
单引号	'	\'	39	0x27
双引号	"	\"	34	0x22

3.5 void

表示无类型。

3.6 const

表示常量。

3.7 浮点数

3.7.1 基本概述

C++ 规定了三种浮点数:float、double、long double。在不同的机器中位数一般不一致,通常 float 有 32 位, double 64 位, long double 128 位。可使用 cfloat 库中的关键字查看。

3.7.2 浮点常量

浮点常量默认为 double 类型,如果需要声明为 float 或 long double,需要分别添加后缀 F 和 L(不区分大小写)。

3.7.3 浮点数输出

这里只介绍一两种通用的方法,更多的方法暂时省略。**1. 保留指定位数**使用 C 语言的格式化输出即可。

```
1 float pi = 3.1415926535;
2 printf("pi = %.6f\n",pi);
```

2. 输出末尾 0 使用 C 语言的格式化输出即可，注意 C 语言不支持 long double。

```
1 double pi2 = 3.15;
2 printf("pi = %.6f\n",pi2);
```

3. 舍去多余尾数如果是保留到整数位，可以使用 floor 函数或者 ceil 函数，如下例子：

```
1 float e = 2.718;
2 printf("%.0f\n",floor(e));//向下取整，需要引入cmath库
3 printf("%.0f\n",ceil(e));//向上取整
```

否则可以先左移再进行取整后再右移。

```
1 float test = 29.4525277;
2 printf("%.2f",floor(test*1e2)/1e2);//保留两位小数(去尾法)
3 printf("%.2f",floor(test*1e2)/1e2);//保留两位小数(进一法)
```

3.8 类型转换

3.8.1 隐式类型转换

为避免混乱，尽量避免进行隐性类型转换，尤其是无符号和有符号的转换。隐性类型转换的规则（需要特别留意 bool 和 char 的转换）：

- 先将 char, bool, short 等类型提升至 int（或 unsigned int，取决于原类型的符号性）类型；
- 若存在一个变量类型为 long double，会将另一变量转换为 long double 类型；
- 否则，若存在一个变量类型为 double，会将另一变量转换为 double 类型；
- 否则，若存在一个变量类型为 float，会将另一变量转换为 float 类型；
- 否则（即参与运算的两个变量均为整数类型）：
 - 若两个变量符号性一致，则将位宽较小的类型转换为位宽较大的类型；
 - 否则，若无符号变量的位宽不小于带符号变量的位宽，则将带符号数转换为无符号数对应的类型；
 - 否则，若带符号操作数的类型能表示无符号操作数类型的所有值，则将无符号操作数转换为带符号操作数对应的类型；
 - 否则，将带符号数转换为相对应的无符号类型。

3.9 强制类型转换

有两种方式，如 (long)a 和 long(a)。强制类型转换不会改变原本变量的值，而是创建一个新的变量用于参与表达式运算。

3.10 auto

auto 是 C++ 11 加入的特性，用于推测类别。注意 auto 不会进行类型转换，给予字面量时总是和字面量的默认类型一致。如 0 是 int, 0.0 是 double。

3.11 测试代码

```
1 #include<iostream>
2 #include<climits>
3 #include<cmath>
4 using namespace std;
5
6 void testfunc(){
7     bool b = false;
8     cout<<sizeof(b)<<endl;
9     cout<<LONG_LONG_MAX<<endl;
10    float pi = 3.1415926535;
11    printf("pi = %.6f\n",pi);
12    double pi2 = 3.15;
13    printf("pi = %.6f\n",pi2);
14    float e = 2.718;
15    printf("%.0f\n",floor(e));//向下取整，需要引入cmath库
16    printf("%.0f\n",ceil(e));//向上取整
17    float test = 29.4525277;
18    printf("%.2f\n",floor(test*1e2)/1e2);//保留两位小数(去尾法)
19    printf("%.2f\n",ceil(test*1e2)/1e2);//保留两位小数(进一法)
20 }
21
22 int main(){
23     testfunc();
24 }
```

4 运算

4.1 自增自减运算符

操作依次从左往右操作。如 `i++` 是先返回 `i` 再自增。尽量避免自增或自减运算符和其他运算符的嵌套，因为 C++ 未明确定义表达式中自增返回值的时机，不同的 C++ 编译器可能返回不同的结果。可参考 C++ Primerplus 中关于顺序点的讨论 (P134，第六版)。

4.2 逗号运算符

起分割运算符的作用，优先级最低，返回最右边的值。**不建议在普通情况下使用，可能引起误会。**

```
exp1, exp2, exp3; // 最后的值为 exp3 的运算结果。

Result = 1 + 2, 3 + 4, 5 + 6;
//得到 Result 的值为 3 而不是 11，因为赋值运算符 "="
//的优先级比逗号运算符高，先进行了赋值运算才进行逗号运算。

Result = (1 + 2, 3 + 4, 5 + 6);

// 若要让 Result 的值得到逗号运算的结果则应将整个表达式用括号提高优先级，此时
// Result 的值为 11。
```

4.3 赋值语句

赋值语句返回值是变量赋值之后的值，一般出现在误将“`==`”记为“`=`”的情况下。

4.4 运算优先级

总体上是!`>` 算术 `>` 关系 (如大于)`>` 与 `>` 或 `>` 赋值运算符。详细可参见[百度百科-运算优先级](#)。

4.5 结合性

先考虑运算优先级，再考虑结合性。结合性分为左结合和右结合，C 语言中单目运算符、条件运算符、赋值语句是右结合以外，其他基本均是左结合性。可参考帖子[C++ 运算符结合性（左结合，右结合）](#)

4.5.1 `a[i++]` 和 `a[++i]`

作为一个例子，背住即可。先计算 `i++`(或者 `[]` 内的东西)，再计算 `[]`。
例子：

```
1 int a[5] = {0,1,2,3,4};
2 int i = 0;
3 while(true){
4     if(i>=4) break;
5     cout<<a[i++]<<endl;
6 }
7 int j = 0;
8 while(true){
9     if(j>=4) break;
10    cout<<a[j++]<<endl;
11 }
```

前者输出 1、2、3、4 后者输出 0、1、2、3。

4.6 除法的类型转换

除法的类型转换完全符合前文中的类型转换。两个整数需要浮点数结果需要先将其中一个数转换为浮点数。

4.7 测试代码

```
1 #include<iostream>
2 using namespace std;
3
4 void testfunc(){
5     int a[5] = {0,1,2,3,4};
6     int i = 0;
7     while(true){
8         if(i>=4) break;
9         cout<<a[i++]<<endl;
10    }
11    int j = 0;
12    while(true){
13        if(j>=4) break;
```



```

14         cout<<a[j++]<<endl;
15     }
16 }
17
18 int main(){
19     testfunc();
20 }

```

5 流程控制语句

5.1 if-else 控制语句

5.1.1 if

功能介绍略。这里介绍一种避免将关系语句误写为赋值语句的方法：使用 `if(3==i)` 替换 `if(i==3)`。

条件运算符和错误防范

许多程序员将更直观的表达式 `variable == value` 反转为 `value == variable`，以此来捕获将相等运算符误写为赋值运算符的错误。例如，下述条件有效，可以正常工作：

```
if (3 == myNumber)
```

但如果错误地使用下面的条件，编译器将生成错误消息，因为它以为程序员试图将一个值赋给一个字面值（3总是等于3，而不能将另一个值赋给它）：

```
if (3 = myNumber)
```

假设犯了类似的错误，但使用的是前一种表示方法：

```
if (myNumber = 3)
```

编译器将只是把3赋给myNumber，而if中的语句块将包含非常常见的、而又非常难以发现的错误（然而，很多编译器会发出警告，因此注意警告是明智的）。一般来说，编写让编译器能够发现错误的代码，比找出导致难以理解的错误的原因要容易得多。

5.1.2 else

略。

5.1.3 else if

略。

5.2 逻辑运算

5.2.1 &&、||、!

略。

5.2.2 细节

1. 使用 `a>3 && a<5` 而不是 `3<a<5`。后者会被视为 `(3<a)<5`。2. 符号! 的优先级高于关系运算符，因此取反要括起来。比如 `!x>1` 表示判断!x 和 1 的大小关系。3. C++ 支持使用 `and`、`or`、`not` 替换 `&&`、`||`、`!`。例如：

```
1 int x = 3;
2 int y = 2;
3 if(3==x and y>1 and y<3) cout<<"If test PASS!"<<endl;
```

5.3 ctype 字符函数库

提供判断字符 (char) 类型的很多函数，比用 if-else 更方便。如可以使用 `isalpha()` 判断是否为字母。以下是一个例子：

```
1 string a_str = "Hello,world!";
2 for(int i = 0;i<a_str.size();i++){
3     if(isalpha(a_str[i]))cout<<a_str[i];
4 }
5 cout<<endl;
```

程序的输出结果为 Helloworld。ctype 其他函数如下：

表 6.4 ctype 中的字符函数

函数名称	返回值
<code>isalnum()</code>	如果参数是字母数字，即字母或数字，该函数返回 true
<code>isalpha()</code>	如果参数是字母，该函数返回 true
<code>isctrl()</code>	如果参数是控制字符，该函数返回 true
<code>isdigit()</code>	如果参数是数字 (0~9)，该函数返回 true
<code>isgraph()</code>	如果参数是除空格之外的打印字符，该函数返回 true
<code>islower()</code>	如果参数是小写字母，该函数返回 true
<code>isprint()</code>	如果参数是打印字符 (包括空格)，该函数返回 true
<code>ispunct()</code>	如果参数是标点符号，该函数返回 true
<code>isspace()</code>	如果参数是标准空白字符，如空格、进纸、换行符、回车、水平制表符或者垂直制表符，该函数返回 true
<code>isupper()</code>	如果参数是大写字母，该函数返回 true
<code>isxdigit()</code>	如果参数是十六进制数字，即 0~9、a~f 或 A~F，该函数返回 true
<code>tolower()</code>	如果参数是大写字符，则返回其小写，否则返回该参数
<code>toupper()</code>	如果参数是小写字符，则返回其大写，否则返回该参数

5.4 ?:

?: 是 C++ 唯一的一个三目运算符。注意嵌套问题，当判断条件过多时推荐使用 if-else 语句。

5.5 switch-case 语句

switch 处填入一个表达式 (值必须在整数集合内)，case 语句填入一个整数值。根据表达式的值，程序跳转到对应的语句**依次**执行代码。要避免依次执行请使用 break 语句。如以下例子：

```
1  int z = 0;
2  while (1)
3  {
4      if(z>=4) break;
5      switch (z)
6      {
7          case 0:
8              cout<<"z is 0."<<endl;
9              break;
10         case 1:
11             cout<<"z is 1."<<endl;
12             break;
13         case 2:
14             cout<<"z is 2."<<endl;
15             break;
16         default:
17             cout<<"Error:z has a wrong value."<<endl;
18             break;
19     }
20     z++;
21 }
```

输出结果：

```
z is 0.
z is 1.
z is 2.
Error:z has a wrong value.
```

5.6 break 和 continue

略。

5.7 测试代码

```
1  #include<iostream>
2  #include<string>
3  #include<cctype>
4  using namespace std;
5
6  void testfunc(){
7      int x = 3;
8      int y = 2;
9      if(3==x and y>1 and y<3) cout<<"If test PASS!"<<endl;
10     string a_str = "Hello,world!";
11     for(int i = 0;i<a_str.size();i++){
12         if(isalpha(a_str[i])) cout<<a_str[i];
13     }
14     cout<<endl;
15     int z = 0;
16     while (1)
17     {
18         if(z>=4) break;
19         switch (z)
20         {
21             case 0:
22                 cout<<"z is 0."<<endl;
23                 break;
24             case 1:
25                 cout<<"z is 1."<<endl;
26                 break;
27             case 2:
28                 cout<<"z is 2."<<endl;
29                 break;
30             default:
31                 cout<<"Error:z has a wrong value."<<endl;
32                 break;
33     }
```

```

34         z++;
35     }
36 }
37
38 int main(){
39     testfunc();
40 }

```

6 循环

6.1 for

6.1.1 基本结构

for 循环基本结构:

```

for (初始化; 判断条件; 更新) {
    循环体;
}

```

任何一个部分都可以省略 (; 不能省略), 省略判断条件默认为真。

for 循环是入口条件循环, 每轮循环开始前都将计算循环条件, 因此当循环条件较为复杂时会降低效率 (视编译器而定)。

6.1.2 和逗号运算符使用的技巧

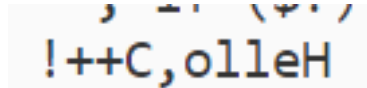
for 循环的另一个技巧是和逗号运算符的使用, 参见以下反转字符串的方法:

```

1 string s = "Hello,C++!";
2 int i,j;//必须提前声明, int j = s.size()-1不是一个表达式语句。
3 for(i = 0,j = s.size()-1;i<j;i++,j--){
4     char c = s[i];
5     s[i] = s[j];
6     s[j] = c;
7 }
8 cout<<s<<endl;

```

输出:



6.1.3 基于范围的循环

C++ 引入的新特性，类似于 Python 的范围遍历，**通常和 auto 结合使用**。一个简单的例子如下，用于输出数组全部元素。

```
1 float f[4] = {0.1,0.2,0.3,1.1};
2 for(auto x:f){
3     cout<<x<<endl;
4 }
```

6.2 while

功能略。永远循环的两种等价写法:

1. for(;;)
2. while(1)

6.3 do-while

先执行语句再判断条件，一般不常用。

6.4 测试代码

```
1 #include<iostream>
2 #include<string>
3 using namespace std;
4
5 void testfunc(){
6     string s = "Hello,C++!";
7     int i,j;//必须提前声明, int j = s.size()-1不是一个表达式语
8     句。
9     for(i = 0,j = s.size()-1;i<j;i++,j--){
10         char c = s[i];
```

```
10     s[i] = s[j];
11     s[j] = c;
12 }
13 cout<<s<<endl;
14 float f[4] = {0.1,0.2,0.3,1.1};
15 for(auto x:f){
16     cout<<x<<endl;
17 }
18 }
19
20 int main(){
21     testfunc();
22 }
```

7 C++ 高级数据类型

7.1 一维数组

7.1.1 基本概述

只能存访相同类型，声明长度必须是常量（被 `const` 修饰）。

7.1.2 数组越界

会触发段错误或者修改意外之外的值。

7.1.3 数组初始化

直接阅读以下代码和注释即可：

```
1 //初始化数组
2 int arr1[4];
3 int arr2[3] = {1,2,3};
4 int arr3[7] = {1,2,3}; //不满默认补零。
5 int arr4[10] = {0}; //快速初始化一个全为0的数组
6 int arr5[] = {12,23,445,464,1225}; //自动推测数组长度
```

打印数组的结果如下，可用于检验数组生成结果：

```
[0,0,0,0]
[1,2,3]
[1,2,3,0,0,0,0]
[0,0,0,0,0,0,0,0,0,0]
[12,23,445,464,1225]
```

注意: 在程序的非静态区声明的数组取值不一定为默认值 (0)。

7.1.4 C++11 的数组初始化

可以省略“=”号, 若“{}”中无内容将会视为全 0。如以下代码:

```
1 int arr6[] {1,3,5,7,9};
2 int arr7[5] {};
```

7.2 多维数组

多维数组实质上在计算机内仍是按一维数组存储, 不过其下标进行了对应的换算。数组存储的方法有很多种, 如二维数组有按行优先和按列优先两种存储方法。实际中常常尽量选择局部性良好的方法。

另外, 多维数组应尽量使其存储位置连续, 这样可以利用 Cache 加速实现性能提升。C++ 中 vector 速度慢于自带的多维数组的原因便是 vector 行间数据存储不连续。详情可见[C++ 多维 vector 为什么比多维数组慢](#)。

7.3 字符串

7.3.1 C 风格字符串

一个包含 ‘\0’ 的 char 数组, 且 ‘\0’ 视为字符串的结束, 该字符以后的字符被忽略。

1. 初始化

可以使用数组或字符串的形式初始化, 注意数组的空间要给够。以下是例子:

```
1 // 字符串
2 char str1[15] = {'H','e','l','l','o',' ',' ','w','o','r','l','d',' ','\0'}; // 末尾 '\0' 不可省略。
```



```
3 char str2[15] = "Hello,world!"; // 多余空间均被赋值为 '\0'。  
4 char str3[] = "Hello,world!"; // 自动推测
```

2. 拼接字符串常量

C++ 允许以下操作:

```
1 cout<<"Hello"  
2 ",world!"<<endl;
```

当字符串很长时此技巧很有用。

3. cstring 库

cstring 库 (或 string.h) 提供了一些与 C 风格相关的函数, 如 strlen() 和 strcmp 函数。

```
1 printf("The size of str1 is %d.\n",strlen(str1));  
2 if(strcmp(str1,str2)) printf("%s","str1 = str2.\n");  
3 else printf("%s","str1!=str2.\n");
```

注意判断字符串相等不能使用“==”, 这里“str1==str2”返回的是两者地址是否相同, 如下:

```
1 printf("The size of str1 is %d.\n",strlen(str1));  
2 if(str1==str2) printf("%s","The address of str1 is equal to  
   str2.\n");  
3 else printf("%s","The address of str1 is not equal to str2.\n")  
   ;
```

7.3.2 C++ 风格字符串

使用的是 string 库, 具体用法参照官方文档。

7.3.3 字符串 I/O: 读取一行

对于 C 风格字符串, 有 cin.get(str,length) 和 cin.getline(str,lenth) 两个函数。这里 length 表示读取的字符串最大长度。当读取到换行符或者读取字符超过最大长度限制将会停止读取。对于 C++ 风格字符串, 有 getline(cin,str) 一个函数。示例如下:

```
1 char str5[50];  
2 char str6[50];
```

```

3 char test_c;
4 string str7;
5 cout<<"Readline begin."<<endl;
6 cin.getline(str5,20);
7 cin.get(str6,20);
8 //get会保留换行符，因此c必为换行符；对于cin可以忽略。
9 scanf("%c",&test_c);
10 cout<<int(test_c)<<endl;
11 getline(cin,str7);
12 cout<<"Readline end."<<endl;

```

输出可见 test_c 的 ASCII 编号为 10，正是代表换行符。

```

Readline begin.
test1
test2
10
test3
Readline end.

```

7.3.4 其他形式的字符串

如下表:

类型名称	别称	适用版本	占用比特数	字面量前缀	备注
wchar_t	宽字符类型	C++	16	L	用于国际化
char16_t	-	C++11	16	u	-
char32_t	-	C++11	32	U	-

另外 C++11 还可以使用前缀 “u8” 指定编码格式，“R” 表示原始字符类型。

7.4 结构体

C 语言不能省略 struct.

7.4.1 结构体初始化

如下代码，解析见注释：

```
1 //结构体声明
2 struct test
3 {
4     int x;
5     int y;
6     double d;
7     string s;
8 };
9 //采用数组的方式初始化
10 test t1 = {1,2,1.2,"test"}; //不省略等号
11 test t2 {1,2,1.3,"test2"}; //省略等号(C++11)
12 cout<<t1.d<<endl;
13 cout<<t2.s<<endl;
14 //声明结构体的同时初始化变量
15 struct position
16 {
17     float x;
18     float y;
19 } p1,p2;
20 cout<< p1.x-p2.x <<endl;
21
22 //省略结构体名,以后无法使用结构体名新建变量
23 struct {
24     double x;
25     double y;
26     string name;
27 } shop = {1.0,1.0,"Cake"};
28 cout<<shop.name<<endl;
```

7.4.2 结构数组

略。

7.4.3 结构中的位字段

使用“:”指定占据特定位数的结构成员，常用于低级编程。此处不展开。

7.5 共用体

只能同时存储多种数据类型中的一种，当一个变量拥有多种不同的数据类型时可以使用。常用于嵌入式编程节省内存，普通系统不常用，此处不展开。

7.6 枚举

7.6.1 声明和赋值

一种创建符号常量的方法，声明与 struct 类似。默认将枚举赋值为 0,1,2,...，枚举类型能隐式转换为整数，但是整数不可以隐式转换为枚举类型。枚举类型只定义了赋值语句，其他操作未定义。当整数超出了枚举类型范围也是未定义行为，在某些编译器中将会报错。例子如下：

```
1 enum color {red,green,white,black};
2 color c = red;
3 cout<<c<<endl;
4 //color c2 = 3; Error!
5 color c2 = (color)3;//OK
```

7.6.2 指定枚举的值

枚举类型允许有相同的值，枚举有效范围为枚举值中最大的值。C++ 允许枚举被赋值为 long long 类型，不允许非整数类型。例子如下：

```
1 enum bits {a = 2,one = 2,two = 4,three = 8,four = 16,super_long
    = 2147483649L};//可声明为 long long.
2 bits b = (bits)6;//Valid, In range.
3 cout<<bits::super_long<<endl;//访问枚举值使用::符号
```

7.7 测试代码

```
1 #include<iostream>
2 #include<cstring>
3 #include<string>
4 using namespace std;
5
```

```
6 //打印数组
7 void printfArray(int* arr_begin, int length){
8     cout<<"[";
9     for(int i = 0;i<length;i++){
10         if(i==0) printf("%d",arr_begin[i]);
11         else printf(",%d",arr_begin[i]);
12     }
13     cout<<""]"<<endl;
14 }
15
16 void testfunc(){
17     //初始化数组
18     int arr1[4]; //在非静态区声明不一定全0。
19     int arr2[3] = {1,2,3};
20     int arr3[7] = {1,2,3}; //不满默认补零。
21     int arr4[10] = {0}; //快速初始化一个全为0的数组
22     int arr5[] = {12,23,445,464,1225}; //自动推测数组长度
23     printfArray(arr1,4);
24     printfArray(arr2,3);
25     printfArray(arr3,7);
26     printfArray(arr4,10);
27     printfArray(arr5,5);
28     //C++11 数组初始化
29     int arr6[] {1,3,5,7,9};
30     int arr7[5] {};
31     printfArray(arr6,5);
32     printfArray(arr7,5);
33
34     //字符串
35     char str1[20] = {'H','e','l','l','o',' ',' ','w','o','r','l','d',
36         ' ','!','\0','a','b'}; //末尾'\0'不可省略。
37     char str2[20] = "Hello,world!"; //多余空间均被赋值为'/0'。
38     char str3[] = "Hello,world!"; //自动推测
39     printf("str1:%s\n",str1);
40     printf("str2:%s\n",str2);
41     printf("str3:%s\n",str3);
42
43     cout<<"Hello"
44     ",world!"<<endl;
```

```
44
45     printf("The size of str1 is %d.\n",strlen(str1));
46     if(strcmp(str1,str2)) printf("%s","str1 = str2.\n");
47     else printf("%s","str1!=str2.\n");
48
49     printf("The size of str1 is %d.\n",strlen(str1));
50     if(str1==str2) printf("%s","The address of str1 is equal to
51         str2.\n");
52     else printf("%s","The address of str1 is not equal to str2
53         .\n");
54
55     char str5[50];
56     char str6[50];
57     char test_c;
58     string str7;
59     cout<<"Readline begin."<<endl;
60     cin.getline(str5,20);
61     cin.get(str6,20);
62     //get会保留换行符，因此c必为换行符；对于cin可以忽略。
63     scanf("%c",&test_c);
64     cout<<int(test_c)<<endl;
65     getline(cin,str7);
66     cout<<"Readline end."<<endl;
67
68     //结构体声明
69     struct test
70     {
71         int x;
72         int y;
73         double d;
74         string s;
75     };
76     //采用数组的方式初始化
77     test t1 = {1,2,1.2,"test"}; //不省略等号
78     test t2 {1,2,1.3,"test2"}; //省略等号(C++11)
79     cout<<t1.d<<endl;
80     cout<<t2.s<<endl;
81     //声明结构体的同时初始化变量
82     struct position
```

```
81     {
82         float x;
83         float y;
84     } p1,p2;
85     cout<< p1.x-p2.x <<endl;
86
87     //省略结构体名,以后无法使用结构体名新建变量
88     struct {
89         double x;
90         double y;
91         string name;
92     } shop = {1.0,1.0,"Cake"};
93     cout<<shop.name<<endl;
94
95     enum color {red,green,white,black};
96     color c = red;
97     cout<<c<<endl;
98     //color c2 = 3; Error!
99     color c2 = (color)3;//OK
100
101     enum bits {a = 2,one = 2,two = 4,three = 8,four = 16,
102               super_long = 2147483649L};//可声明为long long.
103     bits b = (bits)6;//Valid, In range.
104     cout<<bits::super_long<<endl;//访问枚举值使用::符号
105 }
106
107 int main(){
108     testfunc();
109 }
```

参考资料

- [1]. [OI-wiki](#).
- [2]. C++ Primerplus 中文版 (第六版).
- [3]. [菜鸟教程](#)
- [4]. [CSDN](#).