C++程序设计

主讲：王老师

尚德机构

学习是一种信仰

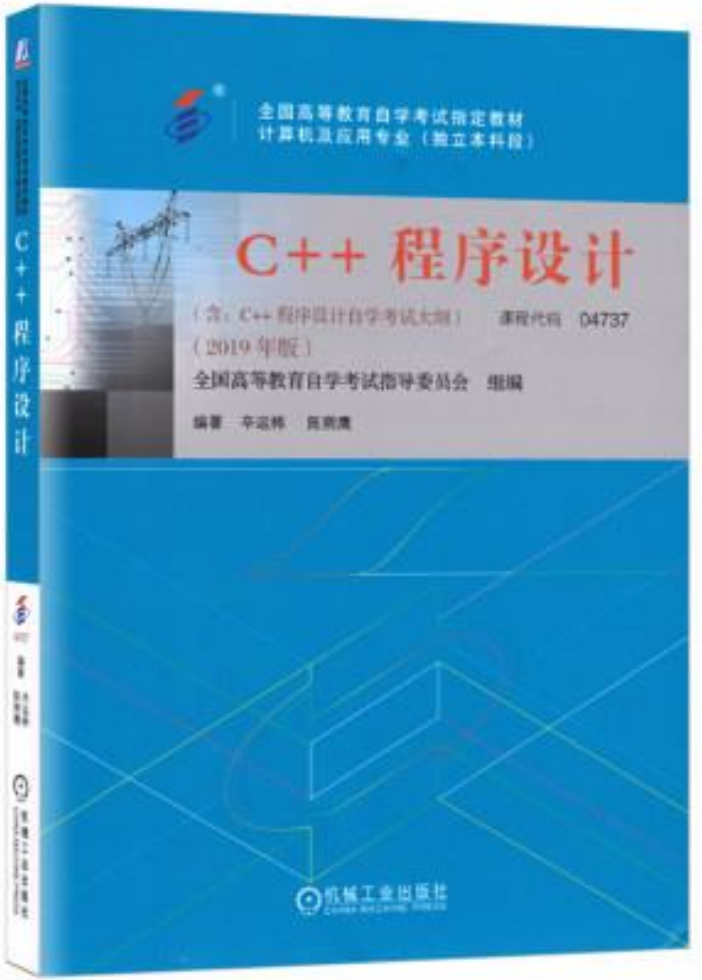
教材介绍

C++程序设计

（2019年版）

编著：辛运帷 陈朔鹰

机械工业出版社



考试题型

单选题　　1分×20题 = 20分

填空题　　1分×15题 = 15分

程序填空题　4分×5题 = 20分

程序分析题　6分×5题 = 30分

程序设计题　　2题 = 15分

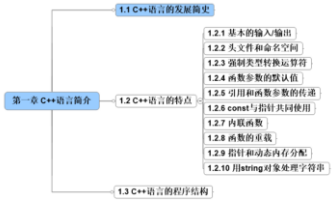
（第一题5分，第二题10分）

第一章 C++语言简介

本章主要内容

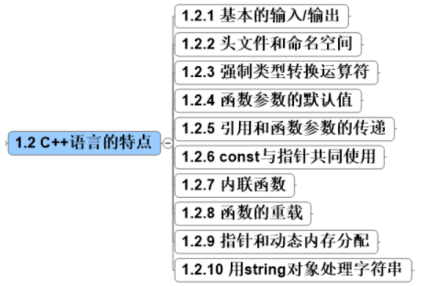
- C++语言的发展简史
- C++语言的特点
- C++语言的程序结构

1.1 C++语言的发展简史



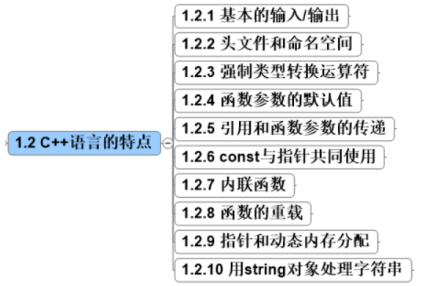
- 将程序设计语言分为低级语言、中级语言和**高级语言**。机器语言和汇编语言属于低级语言一类，因为它们能够直接操纵计算机的寄存器和内存。机器语言是一种依赖于CPU的指令系统，使用机器指令的二进制代码编写程序，能够直接被计算机识别。汇编语言使用能够代表指令的助记符来编写程序，可以看作是符号化了的机器语言。
- 高级语言是面向用户的语言，很多语言在形式上接近于算术语言和自然语言，程序员编写方便。使用高级语言编写的程序易读且通用性强，但大部分不能直接与硬件打交道，也不能直接在计算机上运行，需要系统软件的支持，如需要编译程序及链接程序将高级语言编译链接为机器指令后才能运行。
- C语言是C++语言的前身，在进一步扩充和完善C语言的基础上得到了C++语言。

1.2 C++语言的特点



- 它是C语言的继承，尽量**兼容C语言**，既保持了C语言的简洁和高效，可以像C语言那样进行结构化程序设计，同时也增强了C语言对类型的处理。
- 加入了**面向对象**的特征，可以进行以抽象数据类型为特点的基于对象的程序设计，还可以进行以继承和多态为特点的面向对象的程序设计。

1.2 C++语言的特点



与C语言相比，C++语言的优点：

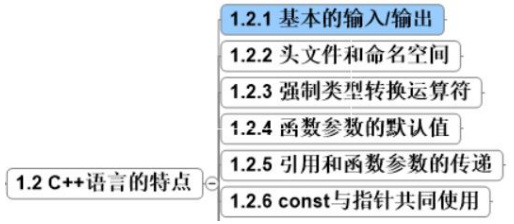
(1)从程序运行的稳定性来说，C++语言比C语言更安全，它支持过程化编程、面向对象编程和泛型编程。因为能够支持面向对象的开发方式，所以C++语言的应用领域更加广泛。

(2)C++语言可运行于多种平台上，如Windows、MAC操作系统及UNIX的多种版本。

(3)C++语言中加入了面向对象的概念，虽然C语言的语法绝大部分都被保留在C++语言中，但C++的程序结构与C语言的程序结构存在很大差别。

C++语言对C语言做了很多改进，C++语言相对于C语言的最根本的变化是引进了类和对象的概念。

1.2.1 基本的输入/输出



功能	C语言中使用函数	C++语言中提供类	C++类中对象	运算符
键盘输入	scanf()	输入流类istream	cin	>>
屏幕输出	printf()	输出流类ostream	cout	<<

- 在C++中，可以使用流提取运算符“>>”从标准输入设备键盘取得数据。例如，语句“**cin**>>x;”从键盘获取输入数据并赋给变量x。使用cin可以获得多个来自键盘的输入值。
- cout是一个标准输出流**对象**，使用流插入运算符“<<”向输出设备屏幕输出信息。

```
#include<iostream>
#include<string>
using namespace std;
int main( )
{
    int oneInt1,oneInt2;
    char strArray[20];
    string str;
    double oneDouble;
    char oneChar='a';
    cout<<"输入两个整型值，一个字符，一个字符串和一个浮点值，";
    cout<<"以空格、Tab键或〈Enter〉键分隔:"<<endl;
    cin>>oneInt1>>oneInt2>>oneChar>>strArray>>oneDouble;
    str=strArray;
    cout<<"输入的数据是:"<<endl;                          //endl的作用是换行
    cout<<"字符串是：\t\t"<<str<<endl                     //\t是制表符Tab
        <<"两个整型值分别是：\t"<<oneInt1<<"和\t"<<oneInt2<<endl
        <<"字符是：\t\t"<<oneChar<<"\n"                   //"\n"是换行符，与endl效果相同
        <<"浮点值是：\t\t"<<oneDouble<<endl;
    return 0;
}
```

假定，程序运行时输入：
10 20 g abc 25.5
屏幕上会输出如下的运行结果：
输入的数据是：
字符串是：　　abc
两个整型值分别是：　10和　20
字符是：　　g
浮点值是：　　25.5

真题演练

已知："int a=5;char c='a';"则输出语句cout<<c+1<<a<<c;的显示结果是（ ）。

A:65a

B:985a

C:98'5'a

D:65'a’

真题演练

已知："int a=5;char c='a';"则输出语句cout<<c+1<<a<<c;的显示结果是（ ）。

A:65a

B:985a

C:98'5'a

D:65'a’

答案：B

真题演练

下面程序的输出结果是（　）

```
#include<iostream>
using namespace std;
int main( )
{int a=1,b=-2,c=3;
if(a<b)
   if(b<0)c=0;
   else c+=1;
cout<<c<<endl;
return 0;
}
```

A:0　B:2　C:3　D:4

真题演练

下面程序的输出结果是（　）

```
#include<iostream>
using namespace std;
int main( )
{int a=1,b=-2,c=3;
if(a<b)
   if(b<0)c=0;
   else c+=1;
cout<<c<<endl;
return 0;
}
```

A:0　B:2　C:3　D:4

真题演练

有以下程序段，其输出结果是（　）

```
#include<iostream>
using namespace std;
void main( )
{  char b[ ]='Hello,you’ ;
   b[5]='!’ ;
   cout<<b<<endl;
}
```

A:Hello,you

B:Hello

C:Hello!you

D:!

真题演练

有以下程序段，其输出结果是（　）

```
#include<iostream>
using namespace std;
void main()
{  char b[ ]='Hello,you’ ;
   b[5]='!’ ;
   cout<<b<<endl;
}
```

A:Hello,you

B:Hello

C:Hello!you

D:!

真题演练

C++中函数中的return指令可以（　）。

A:只能有一条

B:0或多条

C:至少有一条

D:只能主函数调用

真题演练

C++中函数中的return指令可以（　）。

A:只能有一条

B:0或多条

C:至少有一条

D:只能主函数调用

答案：B

真题演练

在一个被调用函数中，关于return语句使用的描述，错误的是（）

A:被调用函数中可以不用return语句

B:被调用函数中可以使用多个return语句

C:被调用函数中，如果有返回值，就一定要有return语句

D:被调用函数中，一个return语句可返回多个值给调用函数

真题演练

在一个被调用函数中，关于return语句使用的描述，错误的是（）

A:被调用函数中可以不用return语句

B:被调用函数中可以使用多个return语句

C:被调用函数中，如果有返回值，就一定要有return语句

D:被调用函数中，一个return语句可返回多个值给调用函数

答案：D

真题演练

下列输出语句中，正确的是（　）。

A:cout<<("%c\n","student")

B:cout<<("%s\n","hello")

C:cout<<("%c\n","c")

D:cout<<("%s\n", &a)

真题演练

下列输出语句中，正确的是（　）。

A:cout<<("%c\n","student")

B:cout<<("%s\n","hello")

C:cout<<("%c\n","c")

D:cout<<("%s\n", &a)

答案：B

真题演练

以下4个选项中，不能看作一条语句的是（　）。

A:if(b==0)m=1;n=2;

B:a=0,b=0,c=0;

C:if(a>0);

D:{;}

真题演练

以下4个选项中，不能看作一条语句的是（　）。

A:if(b==0)m=1;n=2;

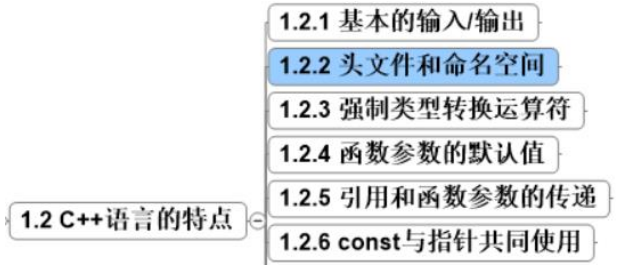
B:a=0,b=0,c=0;

C:if(a>0);

D:{;}

答案：A

1.2.2 头文件和命名空间

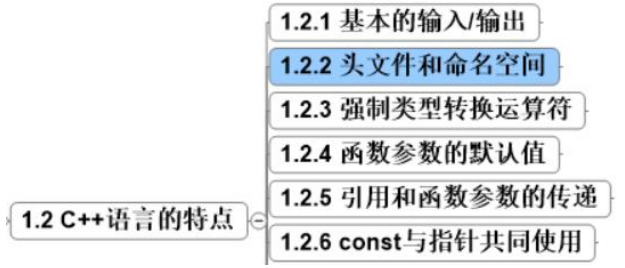


- iostream是C++的标准输入/输出流。当在程序中使用cin或cout时，必须在程序的最前面包含这个流。如果还要使用其他的内容，那么需要包含其他的头文件。每条#include指令仅可以包含一个头文件，如果需要包含多个头文件，则需要使用多条#include指令。【嵌入指令】
- 在C++中，头文件不再以“.h”结尾，以“.h”结尾的头文件是C语言中常用的头文件。

常用的头文件有以下一些。

- 标准输入输出流：**<iostream>**。
- 标准文件流：<fstream>。
- 标准字符串处理函数：<string>。
- 标准数学函数：<cmath>。

1.2.2 头文件和命名空间



当使用尖括号时，C++编译器将首先在C++系统设定的目录中寻找要包含的文件，如果没有找到，再到指令中指定的目录中去查找。采用双引号时，C++编译器在用户当前目录下或指令中指定的目录下寻找要包含的文件。例如，要包含e:\myprog目录下的头文件ex1.h，相应的语句如下：

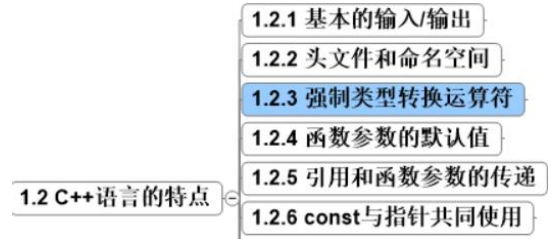
#include"e:\myprog\ex1.h“

C++中为了避免名字定义冲突，特别引入了“命名空间”的定义，即namespace。命名空间的作用是为了消除同名引起的歧义。 using namespace std;

定义一个命名空间的语法格式如下：

```
namespace 命名空间名
{
    命名空间内的各种声明(函数声明、类声明、……)
}
```

*1.2.3 强制类型转换运算符



static_cast用于将一种数据类型转换成另一种数据类型，使用格式如下：

static_cast<类型名>(表达式)

其功能是把表达式转换为类型名所指定的类型。static_cast也可以省略。以程序1-1中声明的变量为例，下面4种写法都是正确的：

```
oneInt2=static_cast<int>(oneDouble); //强制类型转换
oneInt2=int(oneDouble);              //强制类型转换运算符的新形式
oneInt2=(int)oneDouble;              //强制类型转换运算符的旧有形式
oneInt2=oneDouble;                   //自动类型转换
```

真题演练

在下列成对的表达式中，运算结果类型相同的一对是（　）。

A:7 / 2和7.0 / 2.0

B:7 / 2.0和7 / 2

C:7.0 / 2和7 / 2

D:7.0 / 2.0和7.0 / 2

真题演练

在下列成对的表达式中，运算结果类型相同的一对是（　）。

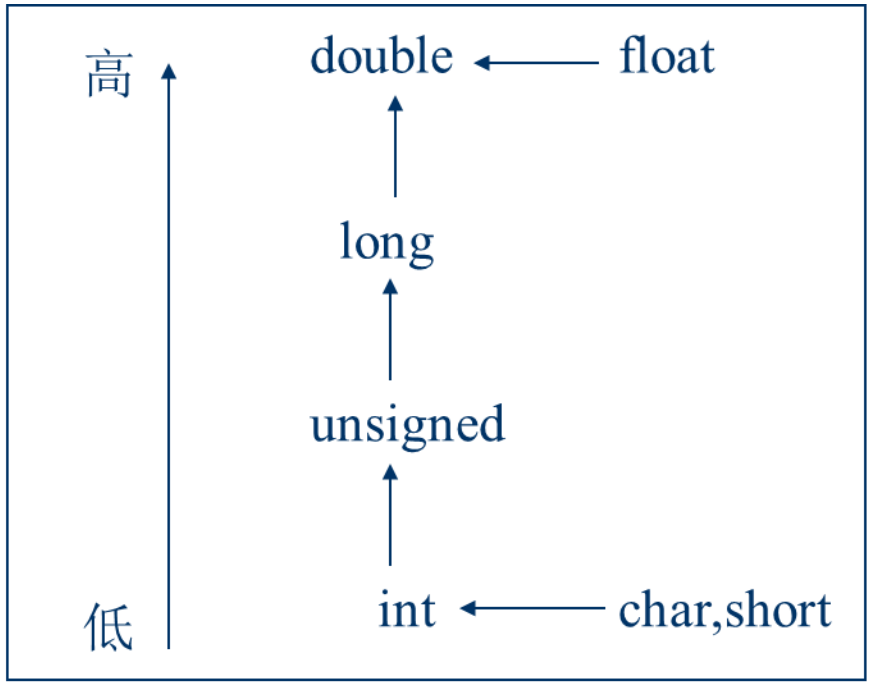
A:7 / 2和7.0 / 2.0

B:7 / 2.0和7 / 2

C:7.0 / 2和7 / 2

D:7.0 / 2.0和7.0 / 2

答案：D



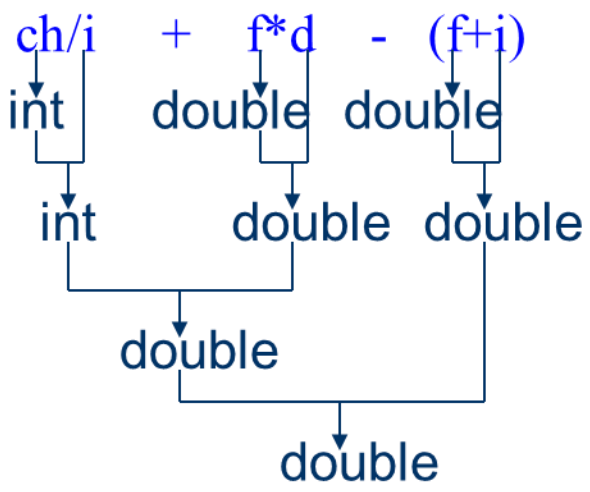
说明：

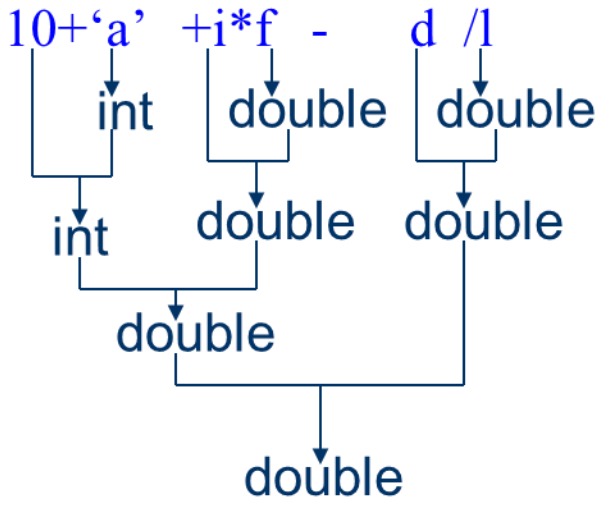
← 必定的转换

↑ 运算对象类型不同时转换

例 int i;
float f;
double d;
long l;

例 char ch;
int i;
float f;
double d;





真题演练

若有以下标志符定义：char c=’c’；int a=4；float f=3.14；double d=1.212；则表达式c+a/（int）d+f的结果类型是（ ）

A:float

B:char

C:int

D:double

真题演练

若有以下标志符定义：char c=’c’；int a=4；float f=3.14；double d=1.212；则表达式c+a/（int）d+f的结果类型是（ ）

A:float

B:char

C:int

D:double

答案：D

真题演练

编写C++程序一般需经过的几个步骤依次是（　）。

A:编辑、调试、编译、连接

B:编译、调试、编辑、连接

C:编译、编辑、连接、运行

D:编辑、编译、连接、运行

真题演练

编写C++程序一般需经过的几个步骤依次是（　）。

A:编辑、调试、编译、连接

B:编译、调试、编辑、连接

C:编译、编辑、连接、运行

D:编辑、编译、连接、运行

答案：D

1.2.4 函数参数的默认值


```
#include<iostream>
using namespace std;
void func(int a=11,int b=22,int c=33)  //为参数a，b，c分别设置了默认值11，22与33
{
    cout<< “a=” <<a<<” ，b="<<b<<"，c="<<c<<endl;
}
int main()
{   func( );              //调用时缺少了3个实参，将使用定义中的3个参数默认值
    func(55);             //调用时缺少了后2个实参，将使用定义中的后2个参数默认值
    func(77,99);          //调用时缺少了最后1个实参，将使用定义中的最后1个参数默认值
    func(8,88,888);       //调用时实参完备，将不使用定义中的任何参数默认值
    return 0;
}
```

```
a=11,b=22,c=33
a=55,b=22,c=33
a=77,b=99,c=33
a=8,b=88,c=888
```

1.2.4 函数参数的默认值

C++语言规定，提供默认值时必须按**从右至左**的顺序提供，即有默认值的形参必须在形参列表的最后。如果有某个形参没有默认值，则它左侧的所有形参都不能有默认值。

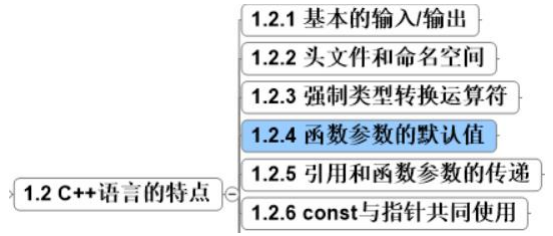
调用函数时，主调函数的实参与被调函数的形参按**从左至右**的顺序进行匹配对应。

以下函数声明，哪些是正确的？对于不正确的函数声明请解释原因。

```
void default value1(int a=2, double b=3.0);
void default value2(int a, double b=3.0);
void default value3(int a=2, double b);

void func1(int a,int b=2,int c=3);
void func2(int a=1,int b,int c=3);
void func3(int a=1,int b=2,int c);
```

1.2.4 函数参数的默认值



例1-2函数调用示例（一）

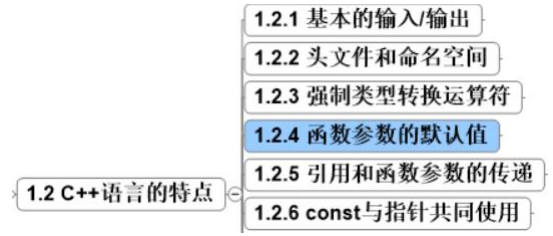
假设给出如下的函数声明：

```
void func(int a,int b=2,int c=3);
```

则下列函数调用中哪些是正确的？请解释原因。

```
func(1,22,333);
func( );
func(10,20);
func(5, ,9);
```

1.2.4 函数参数的默认值



例1-2函数调用示例（一）

假设给出如下的函数声明：

```
void func(int a,int b=2,int c=3);
```

则下列函数调用中哪些是正确的？请解释原因。

```
func(1,22,333);
func( );
func(10,20);
func(5, ,9);
```

【分析】

“func(1,22,333);”是正确的，调用时给出了所有实参，且参数的类型也是匹配的。

“func( );”是错误的，声明中参数a没有默认值，调用时必须给出实参值。

“func(10,20);”是正确的，实参表中的两个值10和20分别对应于函数声明中的形参a和b，参数c使用默认值3。

“func(5, ,9);”是错误的，调用时给出的实参应该是连续排列的。

真题演练

考虑函数原型int fun1(float x,char y='$',int a=9,char b='@')，下面的函数调用中，属于不合法调用的是（　）

A:fun1(3.14)

B:fun1(3.14, '#')

C:fun1(3.14, '$', '@')

D:fun1(3.14, '#',7, '@')

真题演练

考虑函数原型int fun1(float x,char y='$',int a=9,char b='@')，下面的函数调用中，属于不合法调用的是（　）

A:fun1(3.14)

B:fun1(3.14, '#')

C:fun1(3.14, '$', '@')

D:fun1(3.14, '#',7, '@')

答案：C

真题演练

下列函数原型声明语句中，错误的是（　）

A:int f(void);

B:void f(int);

C:int f(a);

D:void f(double a);

真题演练

下列函数原型声明语句中，错误的是（　）

A:int f(void);

B:void f(int);

C:int f(a);

D:void f(double a);

答案：C

真题演练

考虑函数原型void test(int a,int b=7,char ch='*')，下面的函数调用中，属于不合法调用的是（　）

A:test(5)

B:test(5,8)

C:test(6,'#')

D:test(0,0,'*'　)

真题演练

考虑函数原型void test(int a,int b=7,char ch='*')，下面的函数调用中，属于不合法调用的是（　）

A:test(5)

B:test(5,8)

C:test(6,'#')

D:test(0,0,'*'　)

答案：C

真题演练

假设给出如下的函数声明：void func(int a,int b=2,int c=3);，则下列函数调用正确的是（　）

A:func(1,22,333);

B:func();

C:func('#',20);

D:func(5, ,9);

真题演练

假设给出如下的函数声明：void func(int a,int b=2,int c=3);，则下列函数调用正确的是（　）

A:func(1,22,333);

B:func();

C:func('#',20);

D:func(5, ,9);

答案：A

真题演练

设存在函数int max(int,int)返回两参数中较大值，若求22，59，70三者中最大值，下列表达式不正确的是（　）。

A:int m = max(22,max(59,70));

B:int m = max(max(22,59),70);

C:int m = max(22,59,70);

D:int m = max(59,max(22,70));

真题演练

设存在函数int max(int,int)返回两参数中较大值，若求22，59，70三者中最大值，下列表达式不正确的是（　）。

A:int m = max(22,max(59,70));

B:int m = max(max(22,59),70);

C:int m = max(22,59,70);

D:int m = max(59,max(22,70));

答案：C

真题演练

在C++中，函数原型不能标识（　　）。

A:函数的返回类型

B:函数参数的个数

C:函数参数类型

D:函数的功能

真题演练

在C++中，函数原型不能标识（　）。

A:函数的返回类型

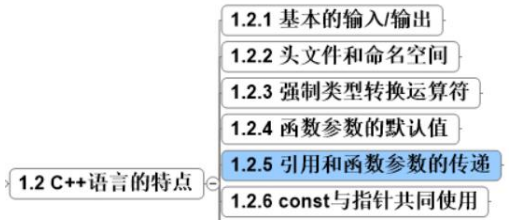
B:函数参数的个数

C:函数参数类型

D:函数的功能

答案：D

1.2.5 引用和函数参数的传递



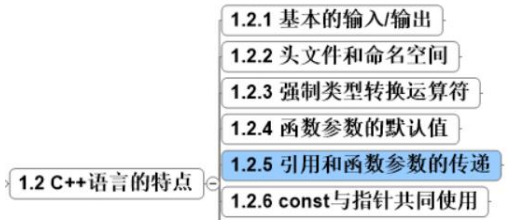
1. 引用的定义

引用相当于给变量起了一个**别名**。变量对应于某个内存地址，**如果给某个变量起了别名（不需要给它另开辟内存单元），相当于变量和这个引用都对应到同一地址**。程序中使用哪个名字都是允许的。在C++中，“引用”的定义格式如下：

类型名 &引用名=同类型的某变量名;

举例如下：

```
int oneInt;
int &aname=oneInt;          //声明引用
```




```
#include<iostream>
using namespace std;
int main()
{
    int oneInt=1;
    int &ref=oneInt;                    //ref是oneInt的引用，ref等价于oneInt
    const int &refc=oneInt;             //定义常引用
    ref=2;                              //修改ref也即修改了oneInt
    cout<<"oneInt="<<oneInt<<" ,"<<"ref="<<ref<<endl;  //输出oneInt=2,ref=2
    cout<<"refc="<<refc<<endl;          //输出refc=2
    oneInt=3;                           //修改oneInt也即修改了ref
    cout<<"ref= "<<ref<<endl;           //输出ref=3
    cout<<"refc= "<<refc<<endl;         //输出refc=3
    int & ref2=ref;                     //ref2和ref都是oneInt的引用
    cout<<"ref2="<<ref2<<endl           //输出ref2=3
    //refc=5;                           //错误，不能使用常引用对所引用的变量进行修改
    return 0;
}
```

1.2.5 引用和函数参数的传递

2. 引用在函数中的使用

- 在程序中不仅能定义变量的引用，还可以将引用用在函数中。引用既可以作为函数的参数使用，也可以作为函数的返回值使用。
- 在C++中，函数调用时参数的传递有两种方式：传值和传引用。
- 传值，实际上是传递对象的值。传引用是传递对象的**首地址值**。如果函数的形参不是引用，那么调用时实参传递给形参通常采用的是传值的方式，即将实参的值拷贝给形参。在函数执行过程中，都是对这个拷贝进行操作的，函数执行完毕返回后，形参的值并不拷贝回实参，也就是说函数内部对形参的改变不会影响到函数外实参的值。
- 如果函数的形参是引用，则调用时实参传递给形参采用的是传引用的方式。函数调用时，实参对象名传递给形参对象名，形参对象名就成为实参对象名的别名，即形参是对应实参的引用，它们是等价的，代表同一个对象，也可以看作是将实参的地址传递给了形参。在函数内部对形参的操作，都是对这个地址的内容进行的，相当于对实参的值进行了操作。所以当函数执行完毕返回后，实参的变化被保留下来。

```
#include<iostream>
using namespace std;

void SwapValue(int a,int b)
{   int tmp;
    tmp=a;
    a=b;
    b=tmp;
    cout<<"在SwapVa1ue( )函数中:\t\ta="<<a<<",b="<<b<<endl;}
void SwapRef(int & a,int & b)                    //a、b值互换
{   int tmp;
    tmp=a;
    a=b;
    b=tmp;
    cout<<"在SwapRef()函数中:\t\ta="<<a<<",b="<<b<<endl;}
int main()
{
    int a=10,b=20;
    cout<<"数据交换前:\t\ta="<<a<<",b="<<b<<endl<<endl;
    SwapValue(a,b);
    cout<<"调用SwapVal()后:\t\ta="<<a<<",b="<<b<<endl<<endl;
    a=10;
    b=20;
    SwapRef(a,b);
    cout<<"调用SwapRef()后:\t\ta="<<a<<",b="<<b<<endl<<endl;
    return 0;
}
```

```
数据交换前:                a=10,b=20

在SwapValue( )函数中:          a=20,b=10
调用SwapVal()后:           a=10,b=20

在SwapRef()函数中:          a=20,b=10
调用SwapRef()后:           a=20,b=10
```

```
#include<iostream>
using namespace std;
int oneX=10;
int oneY=20;
int & refVa1ue(int & x)
{
    return x;
}
int main()
{
    refVa1ue(oneX)=30;                  //返回值是引用，可以作为左值使用
    cout<<"oneX="<<oneX<<endl;          //输出oneX=30
    refVa1ue(oneY)=40;                  //返回值是引用的函数调用表达式，可以作为左值使用
    cout<<"oneY="<<oneY<<endl;          //输出oneY=40
    return 0;
}
```

```
oneX=30
oneY=40
```

真题演练

设int &max(int &,int &)返回两参数中较大者，如有两整型变量int a=10; int b=15; 在执行max(a,b)++后，a，b值分别为（　　）。

A:10，15

B:11，15

C:10，16

D:11，16

真题演练

设int &max(int &,int &)返回两参数中较大者，如有两整型变量int a=10; int b=15; 在执行max(a,b)++后，a，b值分别为（　　）。

A:10，15

B:11，15

C:10，16

D:11，16

答案：C

1.2.6 const与指针共同使用

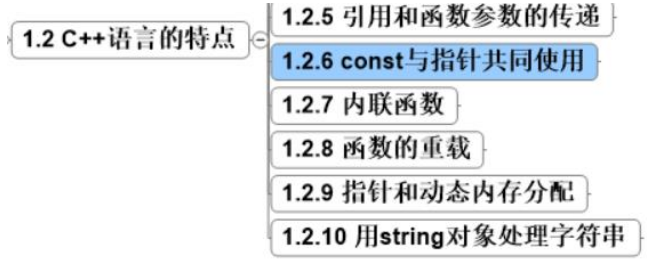
const修饰指针变量时，基本含义如下：

1)如果唯一的const位于符号*的左侧，表示指针所指数据是常量，数据不能通过本指针改变，但可以通过其他方式进行修改；指针本身是变量，可以指向其他的内存单元。

2)如果唯一的const位于符号*的右侧，表示指针本身是常量，不能让该指针指向其他内存地址；指针所指的数据可以通过本指针进行修改。

3)在符号*的左右各有一个const时，表示指针和指针所指数据都是常量，既不能让指针指向其他地址，也不能通过指针修改所指向的内容。

1.2.6 const与指针共同使用



例1-4 单项选择题

设有说明const char * const p="ABCD";，则下列选项中正确的是【 】

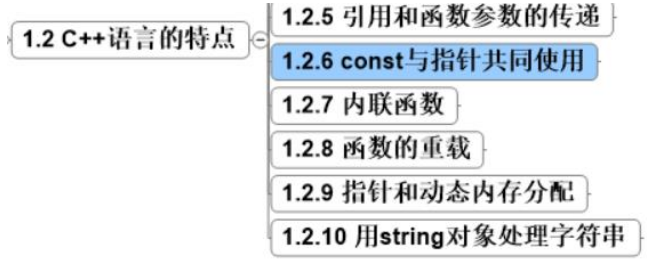
A.允许修改指针p本身，且允许通过p修改所指向的数据

B.允许修改指针p本身，且禁止通过p修改所指向的数据

C.禁止修改指针p本身，且允许通过p修改所指向的数据

D.禁止修改指针p本身，且禁止通过p修改所指向的数据

1.2.6 const与指针共同使用



例1-4 单项选择题

设有说明const char * const p="ABCD";，则下列选项中正确的是【 】

A.允许修改指针p本身，且允许通过p修改所指向的数据

B.允许修改指针p本身，且禁止通过p修改所指向的数据

C.禁止修改指针p本身，且允许通过p修改所指向的数据

D.禁止修改指针p本身，且禁止通过p修改所指向的数据

答案：D。

【分析】const既出现在*的左侧，也出现在*的右侧，表示指针本身及指针所指的数据都是不可修改的。

1.2.6 const与指针共同使用


```
#include<iostream>
using namespace std;
int main( )
{
    int a1=3;                          //普通变量，a1=5是正确的
    const int a2=a1;                   //数据是常量的，a2=5是错误的
    int * a3=&a1;                      //普通指针指向普通变量，*a3=6是正确的
    const int * a4=&a1;                //数据是常量的，普通指针*a4=5是错误的
    int * const a5=&a1;                //指针是常量的，不能修改指针，但*a5=10是正确的
    int const * const a6=&a1;          //数据是常量的，指针也是常量的
    const int * const a7=&a1;          //数据是常量的，指针也是常量的
    return 0;
}
```

可以简单地记住const的修饰规则：**const修饰其左侧的内容；如果const是本行的第一个标识符，则它修饰其右侧的内容。**

1.2.7 内联函数

- 为了避免这种频繁的函数调用与返回，C++语言引入了内联函数的概念。使用内联函数，编译器在编译时并不生成函数调用，而是将程序中出现的每一个内联函数的调用表达式直接用该内联函数的函数体进行替换，就像整个函数体在调用处被重写了一遍一样。很显然，使用内联函数会使最终可执行程序的体积增大。这是以空间消耗节省时间开销。
- 内联函数应该定义在前，调用在后，定义时只需在函数头返回值类型的前面加上**关键字inline**。
- 内联函数主要**应用于代码量少的函数，频繁调用；**
- 如果函数体中有循环语句和switch语句则通常不定义为内联函数。

真题演练

内联函数的特点是（　）。

A:减少代码量，加快访问速度

B:减少代码量，减缓访问速度

C:增加代码量，减缓访问速度

D:增加代码量，加快访问速度

真题演练

内联函数的特点是（　）。

A:减少代码量，加快访问速度

B:减少代码量，减缓访问速度

C:增加代码量，减缓访问速度

D:增加代码量，加快访问速度

答案：D

真题演练

包含哪种语句的函数不能声明为内联函数（　　）。

A:循环

B:变量自增自减

C:if...else...

D:变量声明

真题演练

包含哪种语句的函数不能声明为内联函数（　　）。

A:循环

B:变量自增自减

C:if...else...

D:变量声明

答案：A

1.2.8 函数的重载

所谓函数重载，是指在程序的同一范围内声明几个功能类似的同名函数。

实现函数的重载必须满足下列条件之一：

- 参数表中对应的参数类型不同。
- 参数表中参数个数不同。

如果函数参数表中不同类型参数的次序不同，也符合上面所说的条件。要注意的是，返回值类型不能用来区分函数，也就是说，如果两个函数的名字和参数表都是一样的，**仅仅是返回值类型不同，则这两个函数不是重载的**，编译器认为它们是重复定义，编译时会报错。

课堂练习

下列有关重载函数的说法中正确的是（　）。

A:重载函数必须具有不同的返回值类型

B:重载函数参数个数必须相同

C:重载函数必须有不同的形参列表

D:重载函数名可以不同

课堂练习

下列有关重载函数的说法中正确的是（　）。

A:重载函数必须具有不同的返回值类型

B:重载函数参数个数必须相同

C:重载函数必须有不同的形参列表

D:重载函数名可以不同

答案：C

```
#include<iostream>
using namespace std;

int biggerInt(int x,int y)                    //返回两个int型数中的较大者
{
    if(x>y) return x;
    else return y;
}
float biggerFloat(float x,float y)            //返回两个float型数中的较大者
{
    if(x>y) return x;
    else return y;
}
double biggerDouble(double x,double y)        //返回两个double型数中的较大者
{
    if(x>y) return x;
    else return y;
}

void main()
{
     int a=5,b=8;
     cout<<biggerInt(a,b)<<endl;
}
```

所谓函数重载，是指在程序的同一范围内声明几个功能类似的同名函数。例如，在同一个类中声明3个求两者中较大值的同名函数，如例1-6所示。使用同一个函数名作为功能一样的函数的函数名，这也符合人们的习惯。针对同名的函数，分别为其编写函数体，即可实现各自的功能。

```
#include<iostream>
using namespace std;

int bigger(int x,int y)              //返回两个int型数中的较大者
{
    if(x>y) return x;
    else return y;
}
float bigger(float x,float y)        //返回两个float型数中的较大者
{
    if(x>y) return x;
    else return y;
}
double bigger(double x,double y)     //返回两个double型数中的较大者
{
    if(x>y) return x;
    else return y;
}

void main()
{
    int a=5,b=8;
    cout<<bigger(a,b)<<endl;
}
```

所谓函数重载，是指在程序的同一范围内声明几个功能类似的同名函数。例如，在同一个类中声明3个求两者中较大值的同名函数，如例1-6所示。使用同一个函数名作为功能一样的函数的函数名，这也符合人们的习惯。针对同名的函数，分别为其编写函数体，即可实现各自的功能。

```
#include<iostream>
using namespace std;

int abs(int n)
{
    return(n<0?-n:n);
}
float abs(float f)
{
    if(f<0) f=-f;
    return f;
}
double abs(double d)
{
    if(d<0) return -d;
    return d;
}

void main()
{
     int a=-6;
     cout<<abs(a)<<endl;
}
```

例1-8 错误的重载函数

```
float add(int, float);          //将整数和浮点数相加，返回浮点数
int   add(int, float);          //将整数和浮点数相加，返回整数，错误!
```

编译阶段，程序还没有执行，所以并不知道返回值是什么，更加确定不了它的类型，所以编译器并不能根据返回值确定该调用哪个函数。

另外，**采用引用参数也不能区分函数**，见例1-9。

例1-9错误的重载函数

```
void print(double);
void print(double&);            //错误!
```

例1-10 调用重载函数

```
int xI=10,yI=20;
float xF=30,yF=40;
double xD=50,yD=60;
cout<<bigger(xI,yI)<<endl;
cout<<bigger(xF,yF)<<endl;
cout<<bigger(xD,yD)<<endl;
```

【分析】根据函数重载的规则，bigger(xI,yI)调用的是int bigger(int,int)函数，bigger(x,yF)调用的是float bigger(float,float)函数，bigger(xD,yD)调用的是double bigger(double,double)函数。

例1-11 调用函数时进行必要的类型提升

```
double bigger(double x,double y)
{
    if(x>y) return x;
    else return y;
}
int xI=10,yI=20;
float xF=30,yF=40;
double xD=50,yD=60;
cout<<bigger(xI,yF)<<endl;
```

例1-12 调用重载函数时的二义性

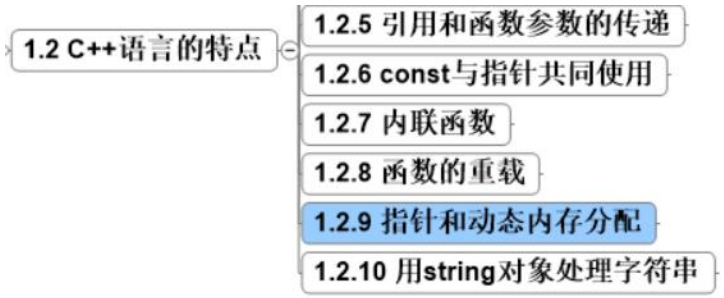
若定义了重载函数Sum( )，如下所示：

```
int Sum(int a,int b,int c=0);
int Sum(int a,int b);
```

则函数调用语句：

```
Sum(1,2);
```

将导致编译错误。因为编译器不知道是应该以(1，2，0)作为参数调用第一个Sum( )函数，还是以(1,2)作为参数调用第二个Sum( )函数，即产生二义性。在设计程序时，应该避免这种情况的发生。



1.2.9 指针和动态内存分配

指针变量中保存的是一个地址，有时也称指针指向一个地址。

例1-13单项选择题

若有以下的定义：

int a=100,*p=&a;

则下列选项中，表述错误的是【 】。

A.声明变量p，其中*表示p是一个指针变量

B.变量p经初始化，获得变量a的地址

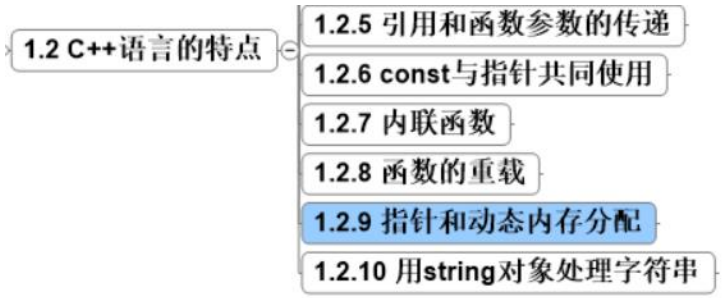
C.变量p只可以指向一个整型变量

D.变量p的值为100

答案：D。

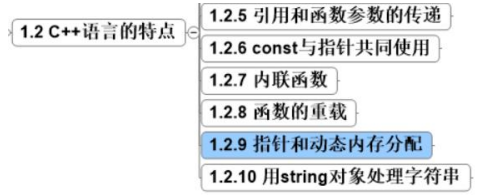
【分析】指针p指向整型变量a，p中保存a的地址，而不是值100。p指向的地址中的值是100。地址与地址中的值不要混淆。

1.2.9 指针和动态内存分配



数组的长度是声明数组时指定的，在整个程序运行过程中通常是不变化的。C++语言不允许定义元素个数不确定的数组。例如：

```
int n;
int a[n];                    //在有些编译环境下，这种定义是不允许的，因为n未知
```



1.2.9 指针和动态内存分配

在C++语言中，使用new运算符实现动态内存分配。例如，可以写如下的语句：

p=new T;

其中，T是任意类型名，p是类型为T*的指针。这样的语句会动态分配出一片大小为sizeof(T)字节的内存空间，并且将该内存空间的起始地址赋值给指针p。例如：

int * p;

p=new int;

*p=5;

第2行语句动态分配了有4个字节大小的内存空间，p指向这个空间的首地址，之后通过指针p可以读写该内存空间。第3行语句是向这个空间中存入数值5。

使用new运算符还可以动态分配一个任意大小的数组：

p=new T[N];

其中，T是任意类型名，p是类型为T*的指针，N代表数组“元素个数”，可以是任何的值为正整数的表达式。数组中元素的类型是T类型。这条语句动态分配了N*sizeof(T)个字节的内存空间，指针p指向这段空间的首地址。

例1-14 动态分配整型数组

```
int * pArray;                    //指向数组的指针
int i = 5;
pArray = new int [i*20];         //分配了100个元素的整型数组
pArray[0] = 20;                  //数组的第一个值
pArray [99] = 30;                //数组的最后一个值
```

数组的下标从0开始，含n个元素的数组的下标范围是从0～n-1。超出这个范围的值都会导致数组下标越界。例如，例1-14中定义的数组pArray，当使用pArray[-1]或是pArray[100]时，下标会越界。不过在编译时，对于数组越界的错误并不会提示，而显示编译正确。运行时，会得到意料之外的结果。

使用new运算符动态申请的内存空间，需要在使用完毕释放。C++提供了**delete**运算符，用来释放动态分配的内存空间。delete运算符的基本用法如下：

delete指针;

delete运算符后面的指针必须是指向动态分配的内存空间的，否则运行时很可能会出错。看例1-15中的语句。

例1-15释放指针时出错

```
int oneInt=6;
int * p=&ontInt;
cout<<*p<<endl;
delete  p;                          //出错，p是引用，不是动态分配的
int * q=new int;
*q=8;
cout<<*q<<endl;
delete q;                           //正确，q指向动态分配的空间
```

【程序说明】例1-15中，指针p和q都指向一个整型变量，但指针p指向的空间并不是使用new运算符分配的，所以不能使用delete释放。而指针q指向的空间是使用new分配的，所以使用完毕，通过delete进行释放。

如果是使用new运算符动态分配了一个数组，那么释放该数组时，语句如下：

delete [ ] 指针;

例如，释放例1-14中分配的数组空间的语句是“delete [ ] pArray;”。

课堂练习

如果有int型变量a，则定义指向变量a的指针p正确的写法是（　）。

A:int p=&a

B:int *p=&a

C:int &p=*a

D:int * p=a

课堂练习

如果有int型变量a，则定义指向变量a的指针p正确的写法是（　）。

A:int p=&a

B:int *p=&a

C:int &p=*a

D:int * p=a

答案：B

课堂练习

假定指针变量p定义为"int * p=new int[100];"，要释放p所指向的动态内存，应使用语句（　）。

A:delete p;

B:delete * p;

C:delete &p;

D:delete [ ]p;

课堂练习

假定指针变量p定义为"int * p=new int[100];"，要释放p所指向的动态内存，应使用语句（ ）。

A:delete p;

B:delete * p;

C:delete &p;

D:delete [ ]p;

答案：D

课堂练习

对类中声明的变量，用new运算符创建一维数组的正确形式是（　）。

A:int*p=new a[10];

B:float*p=new float[10];

C:int*p=new float[10];

D:int*p= new int[5]={1,2,3,4,5,6};

课堂练习

对类中声明的变量，用new运算符创建一维数组的正确形式是（　）。

A:int*p=new a[10];

B:float*p=new float[10];

C:int*p=new float[10];

D:int*p= new int[5]={1,2,3,4,5,6};

答案：B

课堂练习

若有说明：int n=2,*p=&n,*q=p;，则以下非法的赋值语句是（　）。

A:n=*q

B:p=n

C:p=q

D:*q=*p

课堂练习

若有说明：int n=2,*p=&n,*q=p;，则以下非法的赋值语句是（　）。

A:n=*q

B:p=n

C:p=q

D:*q=*p

答案：B

课堂练习

若有定义int *p=new int（0），则下列说法正确的是（　）。

A:系统用指针变量p来表示所指整型变量

B:声明一个指针变量p，指向名为new的存储单元

C:系统为指针变量p分配一个整型数据的存储空间

D:通过运算符new，分配一个整型数据的存储空间，并将其内存地址赋予指针变量

课堂练习

若有定义int *p=new int（0），则下列说法正确的是（　）。

A:系统用指针变量p来表示所指整型变量

B:声明一个指针变量p，指向名为new的存储单元

C:系统为指针变量p分配一个整型数据的存储空间

D:通过运算符new，分配一个整型数据的存储空间，并将其内存地址赋予指针变量

答案：D

课堂练习

若有以下的定义：int a=100,*p=&a;，则下列选项中，表述错误的是（　）。

A:声明变量p，其中*表示p是一个指针变量

B:变量p经初始化，获得变量a的地址

C:变量p只可以指向一个整型变量

D:变量p的值为100

课堂练习

若有以下的定义：int a=100,*p=&a;，则下列选项中，表述错误的是（　）。

A:声明变量p，其中*表示p是一个指针变量

B:变量p经初始化，获得变量a的地址

C:变量p只可以指向一个整型变量

D:变量p的值为100

答案：D

课堂练习

```
#include <iostream>
void main()
{    char *p[10]={”  abc”  ,"aabdfg", "dcdbe","abbd","cd"};
     cout<<p[3]<<endl;
}
```

A:dcdbe

B:abbd

C:abc

D:abb

课堂练习

```
#include <iostream>
void main()
{    char *p[10]={”  abc”  ,"aabdfg", "dcdbe","abbd","cd"};
     cout<<p[3]<<endl;
}
```

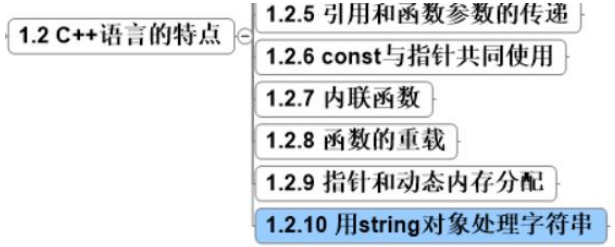
A:dcdbe

B:abbd

C:abc

D:abb

答案：B



1.2.10 用string对象处理字符串

C++标准模板库中提供了string数据类型，专门用于处理字符串。string是一个类，这个类型的变量称为“string对象”。

要在程序中使用string对象，必须在程序中包含头文件string，即在程序的最前面，要加上如下语句：

```
#include<string>
```

声明一个string对象，与声明普通变量是类似的，格式如下：

```
string 变量名;
```

```
string str1;                    //声明string对象str1，值为空
string city="Beijing";          //声明string对象city，并使用字符串常量进行初始化
string str2=city;               //声明string对象str2，并使用字符串变量进行初始化
cout<<"str1="<<str1<<"."<<endl;
cout<<city<<","<<str2<<endl;
```

还可以使用字符数组对string变量进行初始化。例如:

```
char name[ ]="C++程序";
string s1=name;
```

还可以声明一个string对象数组，即数组中每个元素都是字符串。例如:

```
string citys[ ]={"Beijing","Shanghai","Tianjin","Chongqing"};
cout<<citys[1]<<endl;                                   //输出Shanghai，数组下标从0开始
cout<<sizeof(citys)/sizeof(string)<<endl;               //输出数组元素个数
```

最后一行语句将输出数组元素个数。citys是string对象数组，sizeof(citys)是整个数组占用的空间大小，sizeof(string)是每个string对象的大小，所以sizeof(citys)/sizeof(string)表示的是数组元素个数。

2.string对象的操作

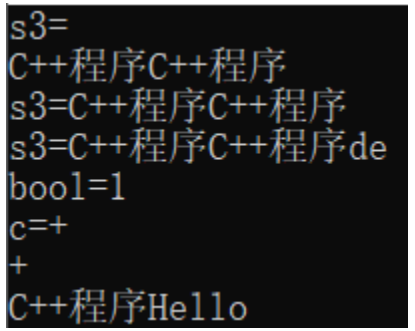
string对象可以使用cin和cout进行输入和输出，实际上，到目前为止，已经使用了很多这样的示例。例如：

```
string s1,s2;
cin>>s1>>s2;
cout<<s1<<","<<s2<<endl;
```

string对象之间可以互相赋值，也可以用字符串常量和字符数组的名字对string对象进行赋值。赋值时不需要考虑被赋值的对象是否有足够的空间来存储字符串。例如：

```
string s1,s2="OK";
s1="China";
s2=s1;                  //s1和s2表示的字符串不等长，赋值后s2的内容和s1相同
cout<<"s1="<<s1<<"s2="<<s2<<endl;
```

```
#include<iostream>
#include<string>
using namespace std;
void main()
{
    string s1, s2;
    s1="C++程序";
    s2=s1;
    string s3;
    cout<<"s3= "<<s3<<endl;          //输出s3=
    s3=s1+s2;
    cout<<s1+s2<<endl;               //输出C++程序C++程序
    cout<<"s3="<<s3<<endl;           //输出s3=C++程序C++程序
    s3+="de";
    cout<<"s3="<<s3<<endl;           //输出s3=C++程序C++程序de
    bool b=s1<s3;                    //b 为true
    cout<<"bool="<<b<<endl;          //输出bool=1
    char c=s1[2];                    //c为'+'，下标从0开始计算
    cout<<"c="<<c<<endl;             //输出c=+
    cout<<s1[2]<<endl;               //输出+
    char arrstr[]="Hello";
    s3=s1+arrstr;
    cout<<s3<<endl;                  //输出C++程序Hello
}
```



```
#include<iostream>
#include<string>
using namespace std;

int main()
{
    string str;                         //未初始化，空串
    if(str.empty())
        cout<<"str is NULL."<<",length="<<str.length()<<endl;
    else
        cout<<"str is not NULL."<<endl;
    str=str.append("abcdefg");
    cout<<"str is "<<str<<",  size="<<str.size()<<endl;
    const char *p=str.c_str();
    cout<<"p="<<p<<endl;
    cout<<"find:"<<str.find("de",0)<<endl;          //查找成功，3
    cout<<"find:"<<str.find("de",4)<<endl;          //查找失败
    string str1=str.insert(4,"123");
    cout<<str1<<endl;
    return 0;
}
```

```
str is NULL.,length=0
str is abcdefg, size=7
p=abcdefg
find:3
find:4294967295
abcd123efg
```

课堂练习

下面不能够判断字符串S是空串的是（　）。

A:if(S[0]==0)

B:if(strlen(S)==0)

C:if(strcmp(S,"")==0)

D:if(S=='\0’　)

课堂练习

下面不能够判断字符串S是空串的是（　）。

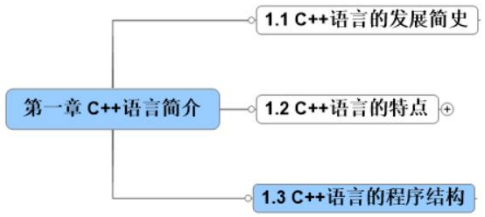
A:if(S[0]==0)

B:if(strlen(S)==0)

C:if(strcmp(S,"")==0)

D:if(S=='\0’　)

答案：D



1.3 C++语言的程序结构

C++程序以.cpp作为文件扩展名，文件中包含若干个类和若干个函数。程序中必须**有且仅有一个主函数main( )**，这是程序执行的总入口。主函数也称为主程序。**程序从主函数main( )的开始处执行，主函数可以在任何地方出现**，按照其控制结构，一直执行到结束。

程序的结束通常是遇到了以下两种情形之一。

1)在主函数中遇到return语句。

2)执行到主函数最后面的括号}。

主函数中可以调用程序中定义的其他函数，但其他函数不能调用主函数。主函数仅是系统为执行程序时所调用的。

C++程序中，仍沿用C语言的注释风格，即注释有以下两种形式：

1)从/*开始，到*/结束，这之间的所有内容都视作注释。

2)从//直到行尾，都是注释。

课堂练习

对指针动态分配空间用的关键字是（　　）。

A:define

B:int

C:new

D:float

课堂练习

对指针动态分配空间用的关键字是（　　）。

A:define

B:int

C:new

D:float

答案：C

课堂练习

对使用关键字new所开辟的动态存储空间，释放时必须使用(　　)。

A:free

B:create

C:delete

D:realse

课堂练习

对使用关键字new所开辟的动态存储空间，释放时必须使用(　　)。

A:free

B:create

C:delete

D:realse

答案：C

本章总结


第一章 C++语言简介
1.1 C++语言的发展简史
1.2 C++语言的特点
1.2.1 基本的输入/输出
1.2.2 头文件和命名空间
1.2.3 强制类型转换运算符
1.2.4 函数参数的默认值
1.2.5 引用和函数参数的传递
1.2.6 const与指针共同使用
1.2.7 内联函数
1.2.8 函数的重载
1.2.9 指针和动态内存分配
1.2.10 用string对象处理字符串
1.3 C++语言的程序结构

祝大家顺利通过考试！