C++ 程序设计 I

徐东

xu.dong.sh@outlook.com

信息与计算科学

数学系

上海师范大学

2018 年 8 月 23 日

内容

1. 变量
2. 类型转换
3. 复合运算
4. 标准输入
5. 常量

需要变量的原因

- 算术表达式

$$(9.5 * 4.5 - 2.5 * 3) / (45.5 - 3.5)$$

- 语句

$$(9.5 * 4.5 - 2.5 * 3) / (45.5 - 3.5) ;$$

- *cout* 语句

$$cout << (9.5 * 4.5 - 2.5 * 3) / (45.5 - 3.5) ;$$

需要变量的原因

- 算术表达式 (未被执行的半成品指令)

$$(9.5 * 4.5 - 2.5 * 3) / (45.5 - 3.5)$$

- 语句

$$(9.5 * 4.5 - 2.5 * 3) / (45.5 - 3.5) ;$$

- *cout* 语句

$$cout << (9.5 * 4.5 - 2.5 * 3) / (45.5 - 3.5) ;$$

需要变量的原因

- 算术表达式（未被执行的半成品指令）

$$(9.5 * 4.5 - 2.5 * 3) / (45.5 - 3.5)$$

- 语句（只计算结果的指令）

$$(9.5 * 4.5 - 2.5 * 3) / (45.5 - 3.5) ;$$

- *cout* 语句

$$cout << (9.5 * 4.5 - 2.5 * 3) / (45.5 - 3.5) ;$$

需要变量的原因

- 算术表达式（未被执行的半成品指令）

$$(9.5 * 4.5 - 2.5 * 3) / (45.5 - 3.5)$$

- 语句（只计算结果的指令）

$$(9.5 * 4.5 - 2.5 * 3) / (45.5 - 3.5) ;$$

- *cout* 语句（计算结果并将结果输出到控制台的指令）

$$cout << (9.5 * 4.5 - 2.5 * 3) / (45.5 - 3.5) ;$$

需要变量的原因

```
cout << 3.14 * 2.1 * 2.1 << endl ;

cout << 3.14 * 2.1 * 2.1 * 6.9 << endl;
```

需要变量的原因

```
cout << 3.14 * 2.1 * 2.1 << endl ;

cout << 3.14 * 2.1 * 2.1 * 6.9 << endl;
```

- 这段代码完成什么任务?

需要变量的原因

```
cout << 3.14 * 2.1 * 2.1 << endl ;

cout << 3.14 * 2.1 * 2.1 * 6.9 << endl;
```

- 这段代码完成什么任务?
- 同一个数据值被重复计算两次

需要变量的原因

```
cout << 3.14 * 2.1 * 2.1 << endl ;

cout << 3.14 * 2.1 * 2.1 * 6.9 << endl;
```

- 这段代码完成什么任务?
- 同一个数据值被重复计算两次 (红色标注的部分)

需要变量的原因

```
cout << 3.14 * 2.1 * 2.1 << endl ;

cout << 3.14 * 2.1 * 2.1 * 6.9 << endl;
```

- 这段代码完成什么任务?
- 同一个数据值被重复计算两次 (红色标注的部分)
- 在程序运行过程中，如果某个数据值 (原始值或中间值) 会被重复使用多次，那么应该把它存储起来。

需要变量的原因

```
cout << 3.14 * 2.1 * 2.1 << endl ;

cout << 3.14 * 2.1 * 2.1 * 6.9 << endl;
```

- 这段代码完成什么任务？
- 同一个数据值被重复计算两次（红色标注的部分）
- 在程序运行过程中，如果某个数据值（原始值或中间值）会被重复使用多次，那么应该把它存储起来。
- 保存数据的（内存）区域，称为变量。

变量

- 变量
  - 类似于储物的盒子

变量

- 变量
  - 类似于储物的盒子
- 储物的盒子

变量

- 变量
  - 类似于储物的盒子
- 储物的盒子
  - 必须有个标签

变量

- 变量
  - 类似于储物的盒子
- 储物的盒子
  - 必须有个标签
  - 必须明确能放什么物品，不能放什么物品。

变量

- 变量
  - 类似于储物的盒子
- 储物的盒子
  - 必须有个标签
  - 必须明确能放什么物品，不能放什么物品。
  - 可以往盒子里放相同类型的物品，也可以把物品取出来。

变量

- 变量
  - 类似于储物的盒子
- 储物的盒子
  - 必须有个标签（变量名）
  - 必须明确能放什么物品，不能放什么物品。
  - 可以往盒子里放相同类型的物品，也可以把物品取出来。

变量

- 变量
  - 类似于储物的盒子
- 储物的盒子
  - 必须有个标签（变量名）
  - 必须明确能放什么物品，不能放什么物品。（数据类型）
  - 可以往盒子里放相同类型的物品，也可以把物品取出来。

变量

- 变量
  - 类似于储物的盒子
- 储物的盒子
  - 必须有个标签（变量名）
  - 必须明确能放什么物品，不能放什么物品。（数据类型）
  - 可以往盒子里放相同类型的物品，也可以把物品取出来。（赋值、取值）

变量

- 变量用于表示（存取）特定类型的、可被改变的数据值。

变量

- 变量用于表示（存取）特定类型的、可被改变的数据值。
- $C++$ 是强类型语言

变量

- 变量用于表示（存取）特定类型的、可被改变的数据值。
- $C++$ 是强类型语言
  - 为了使用变量，必须事先告诉编译器：变量的名字和它可以存储的数据类型。

变量

- 变量用于表示（存取）特定类型的、可被改变的数据值。
- $C++$ 是强类型语言
  - 为了使用变量，必须事先告诉编译器：变量的名字和它可以存储的数据类型。从而，确保对数据执行的操作合法，减少错误。

C++ 的数据类型

- 常用基本数据类型

数据类型	C++ 关键字	说明
整数	int	
浮点数	double	双精度，不精确（不是实数）
字符串	string	不是基本数据类型
字符	char	单个字符或转义字符
布尔值	bool	真假
空类型	void	

C++ 的数据类型

- 科学计数法

双精度数	所表示的实数值
2E-3	0.002
105.4E-10	0.00000001054
2.45e17	24,5000,0000,0000,0000.0
304.24E8	304,2400,0000.0

变量名的命名规则

- 标识符的命名规则

变量名的命名规则

- 标识符的命名规则
  - 由（英文）字母、数字和下划线组成；
  - 第一个字符不能是数字；
  - 不能和 $C++$关键字 冲突；
  - 区分大小写；
  - 应该能反映所存储数据的背景知识（具有描述性的名称）。

- 合法的变量名

变量名的命名规则

- 标识符的命名规则
  - 由（英文）字母、数字和下划线组成；
  - 第一个字符不能是数字；
  - 不能和 $C++$关键字 冲突；
  - 区分大小写；
  - 应该能反映所存储数据的背景知识（具有描述性的名称）。

- 合法的变量名
  - a，$\_n$，age，$income$，$customs\_duties$，$crudeBirthRate$

变量声明

- 变量声明语句

数据类型 变量名 = 初始值 ;

变量声明

- 变量声明语句

数据类型 变量名 = 初始值 ;

- `double radius = 2.1;`

变量声明

- 变量声明语句

$$\text{数据类型}\ \ \text{变量名}\ =\ \text{初始值}\ ;$$

- `double radius = 2.1;`
- 变量声明告知编译器根据指定的数据类型为（目标）变量分配合适的内存空间（并赋予初始值）。

变量声明

- 变量声明语句

$$\text{数据类型}\ \ \text{变量名}\ =\ \text{初始值}\ ;$$

- `double radius = 2.1;`
- 变量声明告知编译器根据指定的数据类型为（目标）变量分配合适的内存空间（并赋予初始值）。
- 程序员应该负责管理每个变量的初值。

变量声明

- 变量声明语句

$$\text{数据类型}\ \ \text{变量名}\ =\ \text{初始值}\ ;$$

- `double radius = 2.1;`
- 变量声明告知编译器根据指定的数据类型为（目标）变量分配合适的内存空间（并赋予初始值）。
- 程序员应该负责管理每个变量的初值。
- 变量必须先声明（或定义）再使用

变量的使用

```
cout << 3.14 * 2.1 * 2.1 << endl ;
cout << 3.14 * 2.1 * 2.1 * 6.9 << endl;
```

变量的使用

```
cout << 3.14 * 2.1 * 2.1 << endl ;
cout << 3.14 * 2.1 * 2.1 * 6.9 << endl;

//引入变量之后
```

变量的使用

```
cout << 3.14 * 2.1 * 2.1 << endl ;
cout << 3.14 * 2.1 * 2.1 * 6.9 << endl;

//引入变量之后
double radius = 2.1 ;
```

变量的使用

```
cout << 3.14 * 2.1 * 2.1 << endl ;
cout << 3.14 * 2.1 * 2.1 * 6.9 << endl;

//引入变量之后
double radius = 2.1 ;
cout << 3.14 * radius * radius << endl ;
cout << 3.14 * radius * radius * 6.9 << endl;
```

变量的使用

```
cout << 3.14 * 2.1 * 2.1 << endl ;
cout << 3.14 * 2.1 * 2.1 * 6.9 << endl;

//引入变量之后
double radius = 2.1 ;
cout << 3.14 * radius * radius << endl ;
cout << 3.14 * radius * radius * 6.9 << endl;
```

变量的使用

```
cout << 3.14 * 2.1 * 2.1 << endl ;
cout << 3.14 * 2.1 * 2.1 * 6.9 << endl;

//引入变量之后
double radius = 2.1 ;
cout << 3.14 * radius * radius << endl ;
cout << 3.14 * radius * radius * 6.9 << endl;
```

- 存储在变量中的（当前）数据值参与表达式的计算

变量声明

- 声明相同数据类型的多个变量

  数据类型 var1 = 初值, var2 = 初值, ···, varN = 初值;

变量声明

- 声明相同数据类型的多个变量

  数据类型 var1 = 初值, var2 = 初值, ···, varN = 初值;

单独声明两个同类型的变量

```
double radius = 2.1;
double h = 6.9;
```

变量声明

- 声明相同数据类型的多个变量

  数据类型 var1 = 初值, var2 = 初值, ···, varN = 初值;

单独声明两个同类型的变量

```
double radius = 2.1;
double h = 6.9;
```

一次声明同类型的两个变量

```
double radius = 2.1, h = 6.9;
```

变量声明

- 声明相同数据类型的多个变量

  数据类型 var1 = 初值, var2 = 初值, ···, varN = 初值;

单独声明两个同类型的变量

```
double radius = 2.1;
double h = 6.9;
```

一次声明同类型的两个变量

```
double radius = 2.1, h = 6.9;
```

- 注意，相同的数据类型变量。

变量声明

- 声明相同数据类型的多个变量

  数据类型 var1 = 初值, var2 = 初值, ···, varN = 初值;

单独声明两个同类型的变量

```
double radius = 2.1;
double h = 6.9;
```

一次声明同类型的两个变量

```
double radius = 2.1, h = 6.9;
```

- 注意，相同的数据类型变量。

赋值语句

```
double radius = 2.1, h = 6.9;
cout << 3.14 * radius * radius << endl ;
cout << 3.14 * radius * radius * 6.9 << endl;
```

- 如何保存在程序运行过程中产生的中间数据?

赋值语句

```
double radius = 2.1, h = 6.9;
cout << 3.14 * radius * radius << endl ;
cout << 3.14 * radius * radius * 6.9 << endl;
```

- 如何保存在程序运行过程中产生的中间数据?
  1. 声明变量

赋值语句

```
double radius = 2.1, h = 6.9;
cout << 3.14 * radius * radius << endl ;
cout << 3.14 * radius * radius * 6.9 << endl;
```

- 如何保存在程序运行过程中产生的中间数据?
  1. 声明变量
  2. 保存数据 (赋值语句)

赋值语句

- 赋值语句

变量 = 数据值 ;

赋值语句

- 赋值语句

$$变量 = 数据值 ;$$

- =

赋值语句

- 赋值语句

$$变量 = 数据值 ;$$

- =
  - 赋值运算符

赋值语句

- 赋值语句

$$变量 = 数据值 ;$$

- =
  - 赋值运算符
  - 优先级最低
  - 从右往左执行运算（右结合性）

赋值语句

- 赋值语句

$$变量 = 数据值 ;$$

- =
  - 赋值运算符
  - 优先级最低
  - 从右往左执行运算（右结合性）
- 赋值后，新值覆盖旧值。

赋值语句

- 赋值语句

$$变量 = 数据值;$$

- =
  - 赋值运算符
  - 优先级最低
  - 从右往左执行运算（右结合性）
- 赋值后，新值覆盖旧值。
- 赋值前，变量值保持不变。

赋值语句

- 使用变量保存中间数据

```
double radius = 2.1, h = 6.9;
```

赋值语句

- 使用变量保存中间数据

```
double radius = 2.1, h = 6.9;
double area = 0.0; //保存底面积，初值0。
```

赋值语句

- 使用变量保存中间数据

```
double radius = 2.1, h = 6.9;
double area = 0.0; //保存底面积，初值0。

area = 3.14 * radius *  radius;
```

赋值语句

- 使用变量保存中间数据

```
double radius = 2.1, h = 6.9;
double area = 0.0; //保存底面积，初值0。

area = 3.14 * radius *  radius;

cout << area << endl ;
cout << area * h << endl;
```

赋值语句

- 使用变量保存中间数据

```
double radius = 2.1, h = 6.9;
double area = 0.0; //保存底面积，初值0。

area = 3.14 * radius *  radius;

cout << area << endl ;
cout << area * h << endl;
```

- 变量必须先声明再使用!!!

赋值语句

```
1  double radius = 2.1, h = 6.9;
2
3  //使用表达式的计算结果对变量初值
4  double area = 3.14 * radius * radius;
5
6  cout << area << endl;
7  cout << area * h << endl;
```

错误的代码

```
1       double area = 3.14 * radius * radius;
2
3       double radius = 2.1, h = 6.9;
4
5       cout << area << endl;
6       cout << area * h << endl;
```

错误的代码

```
1    double area = 3.14 * radius * radius;
2
3    double radius = 2.1, h = 6.9;
4
5    cout << area << endl;
6    cout << area * h << endl;
```

- 变量必须先声明再使用

错误的代码

```
1       double radius = 0.0, h = 0.0;
2
3       double area = 3.14 * radius * radius;
4
5       radius = 2.1;
6       h = 6.9;
7       cout << area << endl;
8       cout << area * h << endl;
```

错误的代码

```
1    double radius = 0.0, h = 0.0;
2
3    double area = 3.14 * radius * radius;
4
5    radius = 2.1;
6    h = 6.9;
7    cout << area << endl;
8    cout << area * h << endl;
```

- 顺序结构导致逻辑错误

注意事项

- 变量必须先声明再使用
- 声明变量时应同时赋初值
- 使用变量即使用变量的当前值
- 赋值兼容
- 编译器只接受合法的操作，不对数据的合理性进行判断。

由不同类型数据组成的算术表达式

$$2L + 3 * 4.5$$

由不同类型数据组成的算术表达式

$$2L + 3 * 4.5$$

- 计算结果的数据类型

由不同类型数据组成的算术表达式

$$2L + 3 * 4.5$$

- 计算结果的数据类型
  - double

由不同类型数据组成的算术表达式

$$2L + 3 * 4.5$$

- 计算结果的数据类型
  - double
- 自动类型转换

由不同类型数据组成的算术表达式

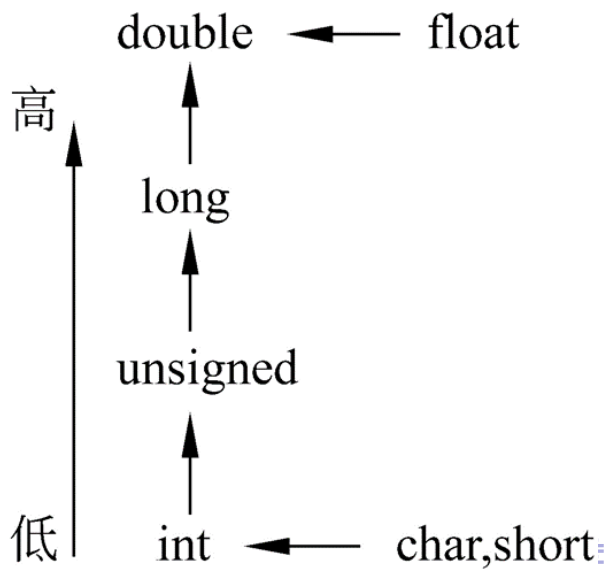
$$2L + 3 * 4.5$$

- 计算结果的数据类型
  - double
- 自动类型转换
  - 小精度自动转变为大精度
  - 确保不损失精度

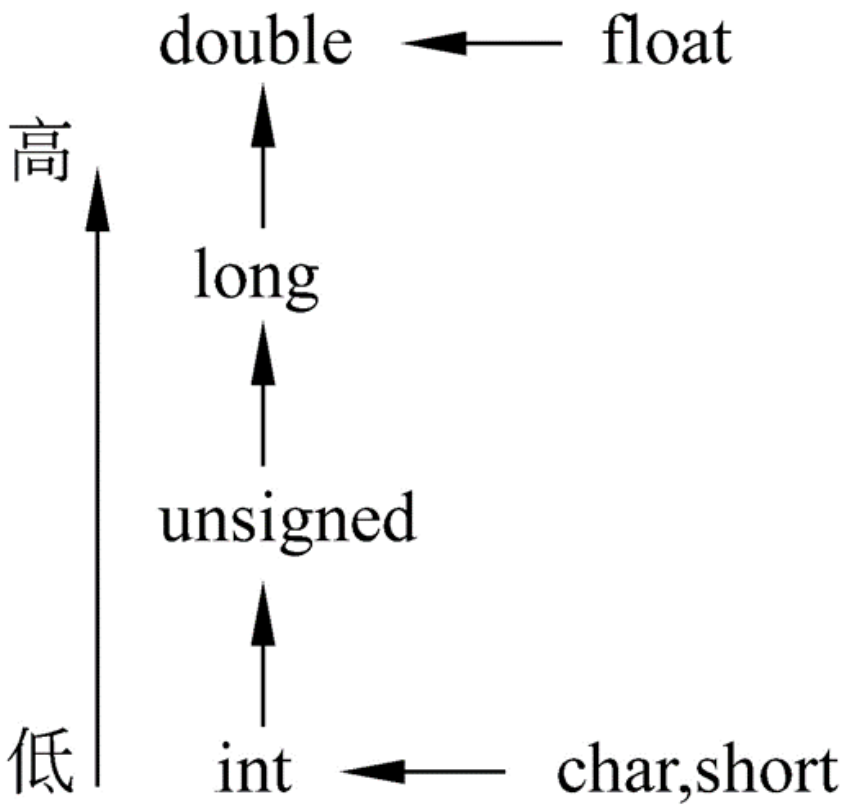
由不同类型数据组成的算术表达式

$$2L + 3 * 4.5$$

- 计算结果的数据类型
  - double
- 自动类型转换
  - 小精度自动转变为大精度
  - 确保不损失精度



类型转换的方向



- 大精度数据类型 $\not\Rightarrow$ 小精度数据类型

强制类型转换

- 形式
  - (目标数据类型)(需要转换的数据)
  - $static\_cast <$ 目标数据类型 $>$ (数据)
  - ...
- 注意赋值兼容

强制类型转换

- 形式
  - (目标数据类型)(需要转换的数据)
  - $static\_cast <$ 目标数据类型 $>$ (数据)
  - ...
- 转换后
  - 损失精度
  - 产生新值 (不影响原始数据)
- 注意赋值兼容

强制类型转换

```
1  (int)(3.14 * 2.1 * 2.1);
2  (int)(3.14); // 等价于(int)3.14;
3
4  static_cast<int>(2 * 3.14 * 2.1);
5  static_cast<int>(3.14);
6
7  double area = 0.0;
8  area = 3.14 * 2.1 * 2.1;
9
10 cout << "转换后的值 = " << (int)(area) << endl;
11 cout << "原始数据未改变 = " << area << endl;
```

强制类型的三个特例

- 赋值语句
- 函数参数
- 函数返回值

强制类型的三个特例

- 赋值语句
- 函数参数
- 函数返回值

强制类型的三个特例

- 赋值语句
- 函数参数
- 函数返回值

```
1    int a = 0, b = 0;
2    a = (int)(3.14 * 2.1 * 2.1);
3    b = 3.14 *2.1 * 2.1;
4    cout << "a␣=␣" << a << endl;
5    cout << "b␣=␣" << b << '\n';
```

强制类型的三个特例

- 赋值语句　　左值（赋值号左边的变量）决定转换方向
- 函数参数
- 函数返回值

```
1    int a = 0, b = 0;
2    a = (int)(3.14 * 2.1 * 2.1);
3    b = 3.14 *2.1 * 2.1;
4    cout << "a␣=␣" << a << endl;
5    cout << "b␣=␣" << b << '\n';
```

不安全类型转换

```
1  int main(){
2      int a = 20000;
3      char c = a;
4      int b = c;
5
6      cout << "a = " << a << endl;
7      cout << "b = " << b << endl;
8      cout << "c = " << c << endl;
9  }
```

不安全类型转换

```
1  int main(){
2      int a = 20000;
3      char c = a;
4      int b = c;
5
6      cout << "a = " << a << endl;
7      cout << "b = " << b << endl;
8      cout << "c = " << c << endl;
9  }
```

- 缩小变换

不安全类型转换

```
1  int main(){
2      int a = 20000;
3      char c = a;
4      int b = c;
5
6      cout << "a = " << a << endl;
7      cout << "b = " << b << endl;
8      cout << "c = " << c << endl;
9  }
```

- 缩小变换
- char 值的范围依赖于计算机（可以方便移植的范围 [0,127]）

安全类型转换

- 等价转换
  - `bool` → `char`
  - `bool` → `int`
  - `bool` → `double`
  - `char` → `int`
  - `char` → `double`
  - `int` → `double`

复合运算符

```
int a = 12;
a = a + 3;
```

复合运算符

```
int a = 12;
a = a + 3;
```

复合运算符

```
int a = 12;
a = a + 3;
```

- 同一个变量出现在赋值号 (=) 的两端

复合运算符

```
int a = 12;
a = a + 3;
```

- 同一个变量出现在赋值号 (=) 的两端
- `a = a + 3;` 等价的简写形式

```
a += 3;
```

复合运算符

```
int a = 12;
a = a + 3;
```

- 同一个变量出现在赋值号 (=) 的两端
- a = a + 3; 等价的简写形式

```
a += 3;
```

- **+=**
  - 复合运算符

复合运算符

复合赋值符	示例	说明
+=	x+=b	x=x+b
−=	x−=b	x=x−b
*=	x*=b	x=x*b
/=	x/=b	x=x/b
% =	x% =b	x=x%b

其中，x 必须是变量。

自增运算

```
int a = 11;
a = a + 1;
```

自增运算

```
int a = 11;
a = a + 1;
```

自增运算

```
int a = 11;
a = a + 1;
```

- `a = a + 1;` 的特定简写形式

自增运算

```
int a = 11;
a = a + 1;
```

- `a = a + 1;` 的特定简写形式

```
a++;
```

自增运算

```
int a = 11;
a = a + 1;
```

- `a = a + 1;` 的特定简写形式

```
a++;
```

- `++`

自增运算

```
int a = 11;
a = a + 1;
```

- `a = a + 1;` 的特定简写形式

```
a++;
```

- `++`
  - 自增运算符

自增表达式的两种形式

- 形式 1

$$var{+}{+}$$

- 形式 2

$${+}{+}var$$

自增表达式的两种形式

- 形式 1

$$var{+}{+}$$

- 形式 2

$${+}{+}var$$

- 两者的差异
  - 自增表达式的副作用

自增表达式的两种形式

- 形式 1

$$var{+}{+}$$

- 形式 2

$${+}{+}var$$

- 两者的差异
  - 自增表达式的副作用
- 自增运算符只能用于变量

自减运算

```
1    int a = 9;
2    a = a - 1;
3
4    a--; // 等价于a = a - 1;
```

自减运算

```
1    int a = 9;
2    a = a - 1;
3
4    a--; // 等价于a = a - 1;
```

- --
  - 自减运算符

自减表达式的两种形式

- 形式 1

$$var\ --$$

- 形式 2

$$--\ var$$

自减表达式的两种形式

- 形式 1

$$var\ --$$

- 形式 2

$$--\ var$$

- 两者的差异
  - 自减表达式的副作用（同自增表达式）

自减表达式的两种形式

- 形式 1

$$var\ --$$

- 形式 2

$$--\ var$$

- 两者的差异
  - 自减表达式的副作用（同自增表达式）
- 自减运算符只能用于变量

cin

- `cin`
  - 标准输入设备 (键盘)
  - `std` 中定义的标识符
- 添加头文件 `iostream`
  - `#`$include < iostream >$

cin

- cin 语句

$$cin \quad >> \quad var \quad ;$$

- >>
  - 流提取运算符
- 按下回车表示输入结束
- 输入值必须与目标变量的数据类型一致或赋值兼容

cin

- cin 语句中只能出现变量
- 可以按（cin 语句中变量的）顺序一次输入全部数据值（注意匹配数据类型）
- $C++$ 采用空白符作为输入数据项之间的分隔符
- >> 操作符自动忽略输入数据项之间的空白符
- 空白符：空格，换行，Tab。

cin

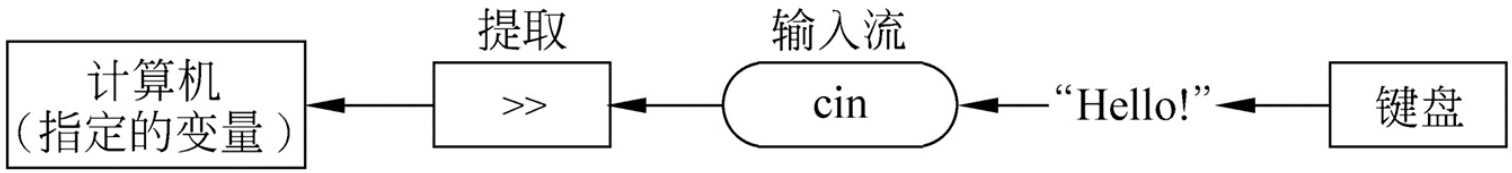
- cin 语句

$$cin >> var1 >> var2 >> \cdots >> varN;$$

- 变量两两之间必须以 >> 分隔
- 输入的数据项之间，以空格或者回车分隔。
- 根据变量的数据类型，系统自动从键盘输入中截取数据并保存至目标变量中。

标准输入流

- 管道（数据流）



任务

- 交换两个（相同数据类型的）变量中的值

常量

- 常量的声明
  - const 关键字
  - 声明时给出常量值
  - 常量值不能被修改

```
1  const double PI = 3.14;
2
3  cout << 2 * PI * 2.1 ;
```

Q & A