

# C++程序设计(拾叁)

---

徐东/计算数学

# 内容

- 引用
- 指针
- 指针运算
- 指针与数组
- 动态数组
- 常量指针（指向常量的指针）
- 指针常量
- 函数指针
- 其他

# 参数传递：值传递

```
int add(int a,int b);
int main(){
    int x = 0, y = 0;
    x = 9;
    y = 86;
    cout << add(x,y) << endl;
    return 0;
}
int add(int a,int b){
    a++;
    ++b;
    return a+b;
}
```

# 参数传递：值传递

```
int add(int a,int b);  
int main(){  
    int x = 0, y = 0;  
    x = 9;  
    y = 86;  
    cout << add(x,y) << endl;  
    return 0;  
}  
int add(int a,int b){  
    a++;  
    ++b;  
    return a+b;  
}
```

main

x

0

y

0

# 参数传递：值传递

```
int add(int a,int b);
int main(){
    int x = 0, y = 0;
    x = 9;
    y = 86;
    cout << add(x,y) << endl;
    return 0;
}
int add(int a,int b){
    a++;
    ++b;
    return a+b;
}
```

main

x

0

y

0

# 参数传递：值传递

```
int add(int a,int b);  
int main(){  
    int x = 0, y = 0;  
    x = 9;  
    y = 86;  
    cout << add(x,y) << endl;  
    return 0;  
}  
int add(int a,int b){  
    a++;  
    ++b;  
    return a+b;  
}
```

main

x

9

y

0

# 参数传递：值传递

```
int add(int a,int b);
int main(){
    int x = 0, y = 0;
    x = 9;
    y = 86;
    cout << add(x,y) << endl;
    return 0;
}
int add(int a,int b){
    a++;
    ++b;
    return a+b;
}
```

main

x

9

y

0

# 参数传递：值传递

```
int add(int a,int b);
int main(){
    int x = 0, y = 0;
    x = 9;
    y = 86;
    cout << add(x,y) << endl;
    return 0;
}
int add(int a,int b){
    a++;
    ++b;
    return a+b;
}
```

main

x

9

y

86



# 参数传递：值传递

```
int add(int a,int b);  
int main(){  
    int x = 0, y = 0;  
    x = 9;  
    y = 86;  
    cout << add(x,y) << endl;  
    return 0;  
}  
int add(int a,int b){  
    a++;  
    ++b;  
    return a+b;  
}
```

main

x

9

y

86

- **调用函数** add()

# 参数传递：值传递

```
int add(int a,int b);  
int main(){  
    int x = 0, y = 0;  
    x = 9;  
    y = 86;  
    cout << add(x,y) << endl;  
    return 0;  
}  
int add(int a,int b){  
    a++;  
    ++b;  
    return a+b;  
}
```

- **调用函数** add()

main

x

9

y

86

add

a

b

# 参数传递：值传递

```
int add(int a,int b);  
int main(){  
    int x = 0, y = 0;  
    x = 9;  
    y = 86;  
    cout << add(x,y) << endl;  
    return 0;  
}  
int add(int a,int b){  
    a++;  
    ++b;  
    return a+b;  
}
```

main

x

9

y

86

add

a

9

b

86

- 实参与形参：值传递

# 参数传递：值传递

```
int add(int a,int b);  
int main(){  
    int x = 0, y = 0;  
    x = 9;  
    y = 86;  
    cout << add(x,y) << endl;  
    return 0;  
}  
int add(int a,int b){  
    a++;  
    ++b;  
    return a+b;  
}
```

main

x

9

y

86

add

a

9

b

86

- 值传递：形式参数可视为实际参数的副本（复制品）

# 参数传递：值传递

```
int add(int a,int b);  
int main(){  
    int x = 0, y = 0;  
    x = 9;  
    y = 86;  
    cout << add(x,y) << endl;  
    return 0;  
}  
int add(int a,int b){  
    a++;  
    ++b;  
    return a+b;  
}
```

main

x

9

y

86

add

a

9

b

86

- 值传递：形式参数的改变不影响实际参数

# 参数传递：值传递

```
int add(int a,int b);  
int main(){  
    int x = 0, y = 0;  
    x = 9;  
    y = 86;  
    cout << add(x,y) << endl;  
    return 0;  
}  
int add(int a,int b){  
    a++;  
    ++b;  
    return a+b;  
}
```

main

x

9

y

86

add

a

10

b

86

- 值传递：形式参数的改变不影响实际参数

# 参数传递：值传递

```
int add(int a,int b);  
int main(){  
    int x = 0, y = 0;  
    x = 9;  
    y = 86;  
    cout << add(x,y) << endl;  
    return 0;  
}  
int add(int a,int b){  
    a++;  
    ++b;  
    return a+b;  
}
```

main

x

9

y

86

add

a

10

b

86

- 值传递：形式参数的改变不影响实际参数

# 参数传递：值传递

```
int add(int a,int b);  
int main(){  
    int x = 0, y = 0;  
    x = 9;  
    y = 86;  
    cout << add(x,y) << endl;  
    return 0;  
}  
int add(int a,int b){  
    a++;  
    ++b;  
    return a+b;  
}
```

main

x

9

y

86

add

a

10

b

87

- 值传递：形式参数的改变不影响实际参数



# 参数传递：值传递

```
int add(int a,int b);
int main(){
    int x = 0, y = 0;
    x = 9;
    y = 86;
    cout << add(x,y) << endl;
    return 0;
}
int add(int a,int b){
    a++;
    ++b;
    return a+b;
}
```

main

x

9

y

86

add

a

10

b

87

- 返回计算结果 97（函数调用结束）

# 参数传递：值传递

```
int add(int a,int b);  
int main(){  
    int x = 0, y = 0;  
    x = 9;  
    y = 86;  
    cout << add(x,y) << endl;  
    return 0;  
}  
int add(int a,int b){  
    a++;  
    ++b;  
    return a+b;  
}
```

main

x

9

y

86

add

a

10

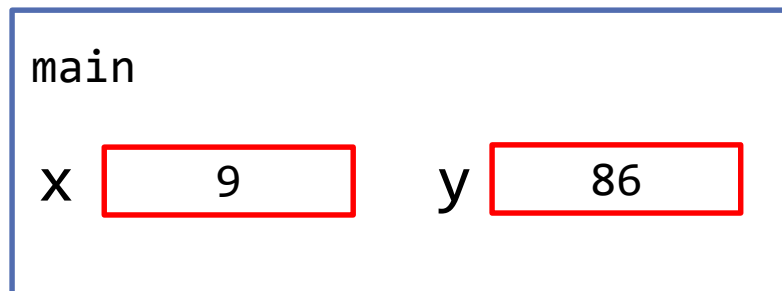
b

87

- 函数add调用结束

# 参数传递：值传递

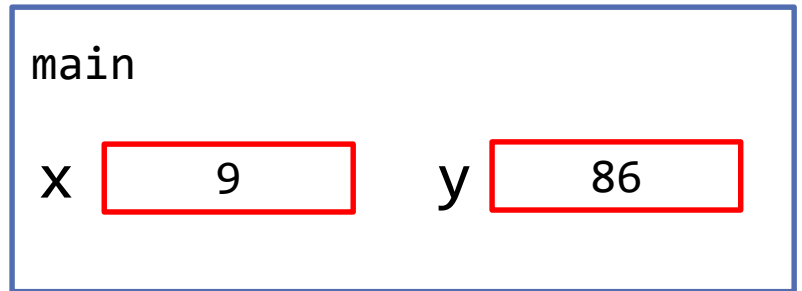
```
int add(int a,int b);  
int main(){  
    int x = 0, y = 0;  
    x = 9;  
    y = 86;  
    cout << add(x,y) << endl;  
    return 0;  
}  
int add(int a,int b){  
    a++;  
    ++b;  
    return a+b;  
}
```



- **函数add调用结束（系统自动删除相关变量）**

# 参数传递：值传递

```
int add(int a,int b);
int main(){
    int x = 0, y = 0;
    x = 9;
    y = 86;
    cout << add(x,y) << endl;
    return 0;
}
int add(int a,int b){
    a++;
    ++b;
    return a+b;
}
```



- **main函数运行结束（系统自动删除主函数中的相关变量）**

# 值传递的优点

```
int add(int a,int b);
int main(){
    int x = 0, y = 0;
    x = 9;
    y = 86;
    cout << add(x,y) << endl;
    return 0;
}
int add(int a,int b){
    a++;
    ++b;
    return a+b;
}
```

main

x

9

y

86

add

a

10

b

87

- 形式参数与实际参数之间互不影响

# 值传递的优点

```
int add(int a,int b);  
int main(){  
    int x = 0, y = 0;  
    x = 9;  
    y = 86;  
    cout << add(x,y) << endl;  
    return 0;  
}  
int add(int a,int b){  
    a++;  
    ++b;  
    return a+b;  
}
```

main

x

9

y

86

add

a

10

b

87

- 形式参数可视为实际参数的副本

# 值传递的缺点

```
int add(int a,int b);
int main(){
    int x = 0, y = 0;
    x = 9;
    y = 86;
    cout << add(x,y) << endl;
    return 0;
}
int add(int a,int b){
    a++;
    ++b;
    return a+b;
}
```

main

x

9

y

86

add

a

10

b

87

- 形式参数可视为实际参数的副本

# 值传递的缺点

```
int add(int a,int b);  
int main(){  
    int x = 0, y = 0;  
    x = 9;  
    y = 86;  
    cout << add(x,y) << endl;  
    return 0;  
}  
int add(int a,int b){  
    a++;  
    ++b;  
    return a+b;  
}
```

main

x

9

y

86

add

a

10

b

87

- 形式参数可视为实际参数的副本（复制需要时间）



# 值传递的缺点

```
int add(int a,int b);  
int main(){  
    int x = 0, y = 0;  
    x = 9;  
    y = 86;  
    cout << add(x,y) << endl;  
    return 0;  
}  
int add(int a,int b){  
    a++;  
    ++b;  
    return a+b;  
}
```

main

x

9

y

86

add

a

10

b

87

- 不适合大型数据值之间的传递

# 解决值传递的缺点的方式

```
int add(int a,int b);  
int main(){  
    int x = 0, y = 0;  
    x = 9;  
    y = 86;  
    cout << add(x,y) << endl;  
    return 0;  
}  
int add(int a,int b){  
    a++;  
    ++b;  
    return a+b;  
}
```

main

x

9

y

86

add

a

10

b

87

- 引用 &

# 引用

```
int add(int &a,int &b);  
int main(){  
    int x = 0, y = 0;  
    x = 9;  
    y = 86;  
    cout << add(x,y) << endl;  
    return 0;  
}  
int add(int &a,int &b){  
    a++;  
    ++b;  
    return a+b;  
}
```

- 把add函数的形式参数声明为“引用”

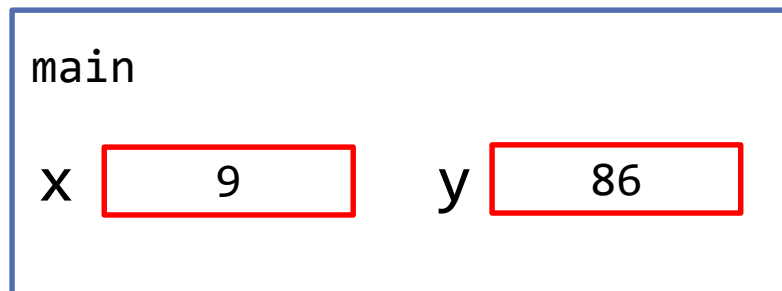
# 引用

```
int add(int &a,int &b);
int main(){
    int x = 0, y = 0;
    x = 9;
    y = 86;
    cout << add(x,y) << endl;
    return 0;
}
int add(int &a,int &b){
    a++;
    ++b;
    return a+b;
}
```

- **通过 & 运算符，把add函数的形式参数声明为“引用”。**

# 引用

```
int add(int &a,int &b);
int main(){
    int x = 0, y = 0;
    x = 9;
    y = 86;
    cout << add(x,y) << endl;
    return 0;
}
int add(int &a,int &b){
    a++;
    ++b;
    return a+b;
}
```



- “引用” 可视为 “变量的别名”

# 引用

```
int add(int &a,int &b);  
int main(){  
    int x = 0, y = 0;  
    x = 9;  
    y = 86;  
    cout << add(x,y) << endl;  
    return 0;  
}  
int add(int &a,int &b){  
    a++;  
    ++b;  
    return a+b;  
}
```

main

x

9

y

86

add

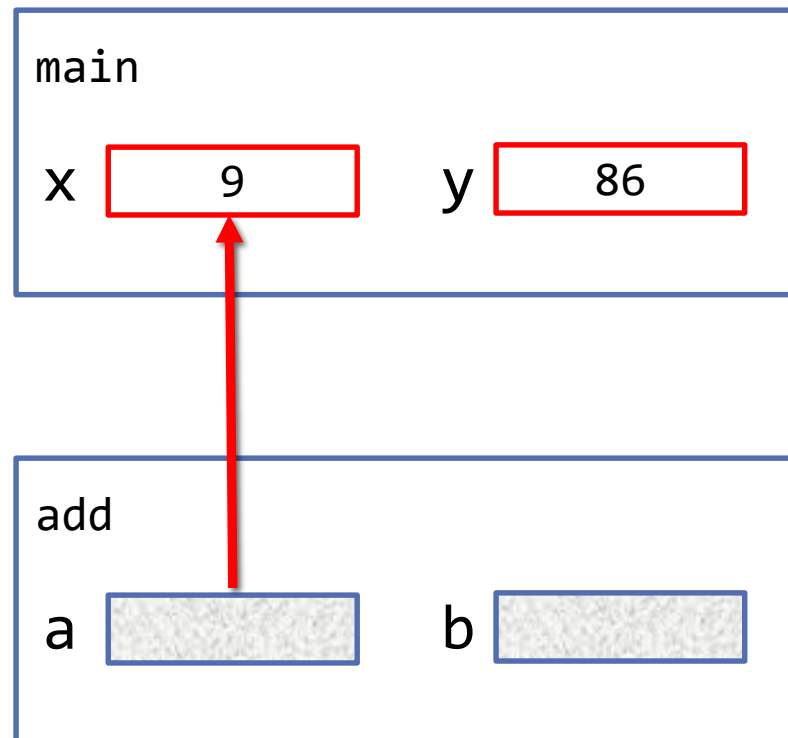
a

b

- “引用” 可视为 “变量的别名”

# 引用

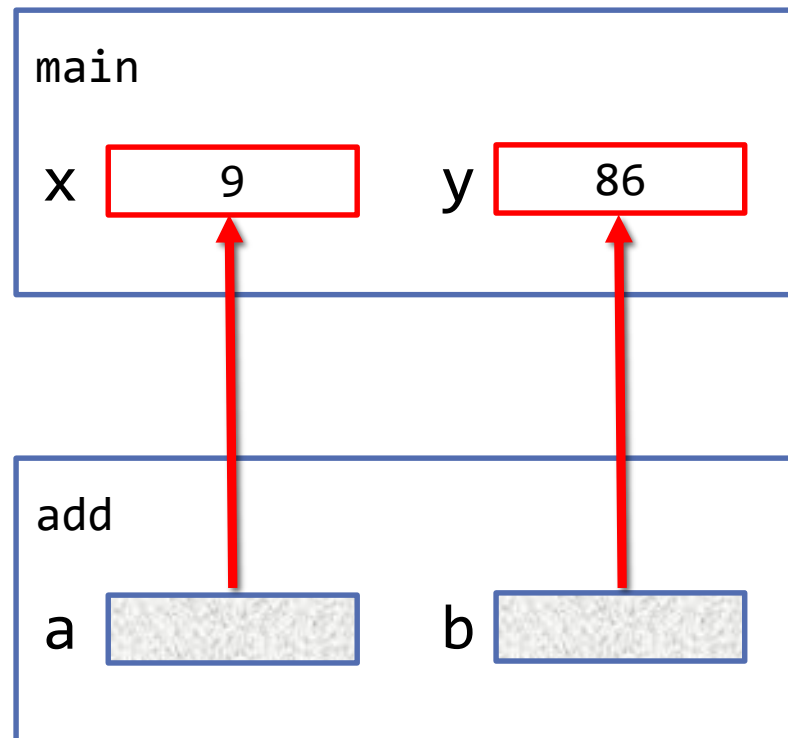
```
int add(int &a,int &b);
int main(){
    int x = 0, y = 0;
    x = 9;
    y = 86;
    cout << add(x,y) << endl;
    return 0;
}
int add(int &a,int &b){
    a++;
    ++b;
    return a+b;
}
```



- “引用” 可视为 “变量的别名” （参数 `a` 就是 变量 `x`）

# 引用

```
int add(int &a,int &b);  
int main(){  
    int x = 0, y = 0;  
    x = 9;  
    y = 86;  
    cout << add(x,y) << endl;  
    return 0;  
}  
int add(int &a,int &b){  
    a++;  
    ++b;  
    return a+b;  
}
```

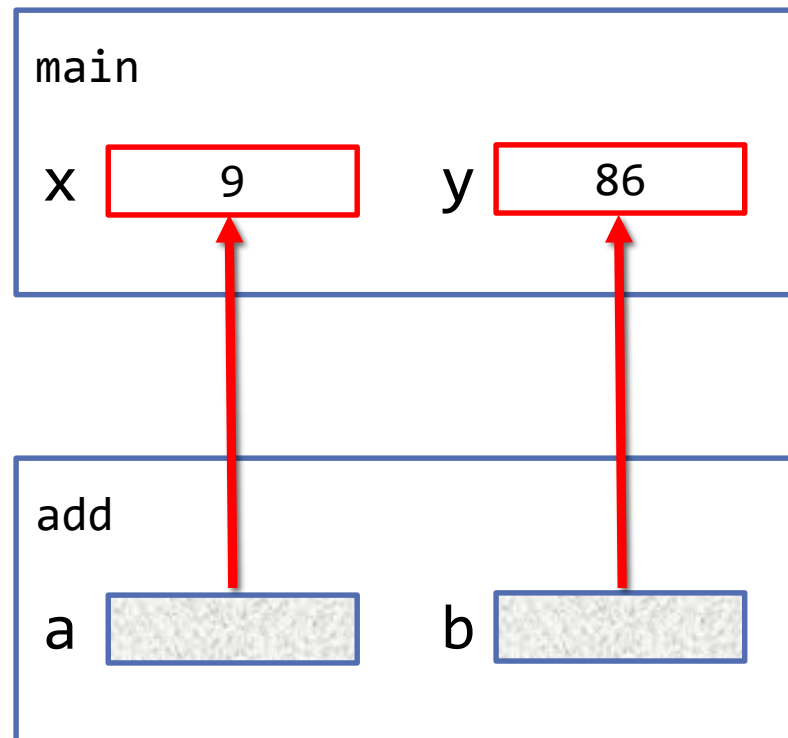


- “引用” 可视为 “变量的别名” （参数 `b` 就是 变量 `y`）



# 引用的优点

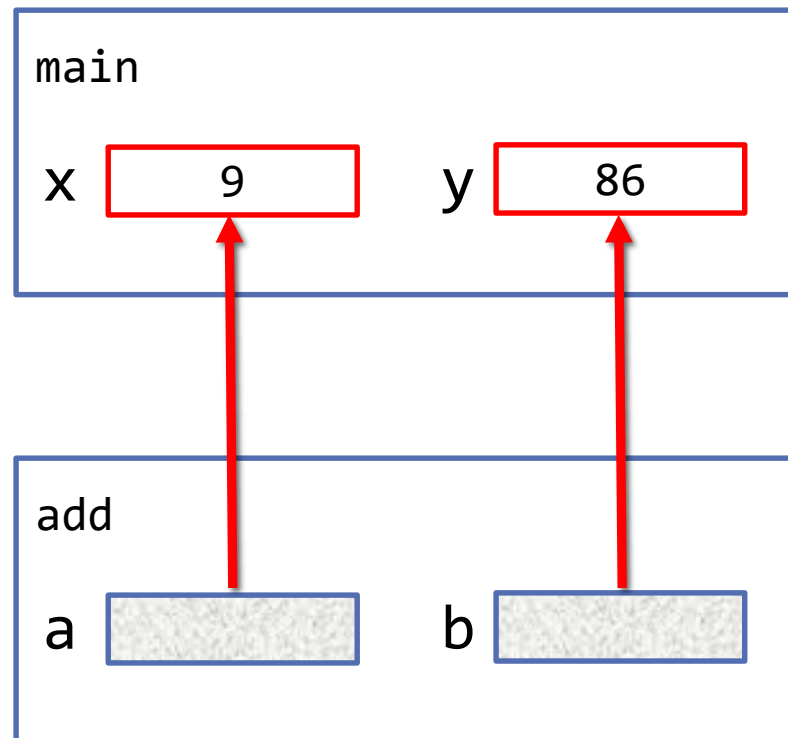
```
int add(int &a,int &b);
int main(){
    int x = 0, y = 0;
    x = 9;
    y = 86;
    cout << add(x,y) << endl;
    return 0;
}
int add(int &a,int &b){
    a++;
    ++b;
    return a+b;
}
```



- “引用” 可视为 “变量的别名”

# 引用的优点

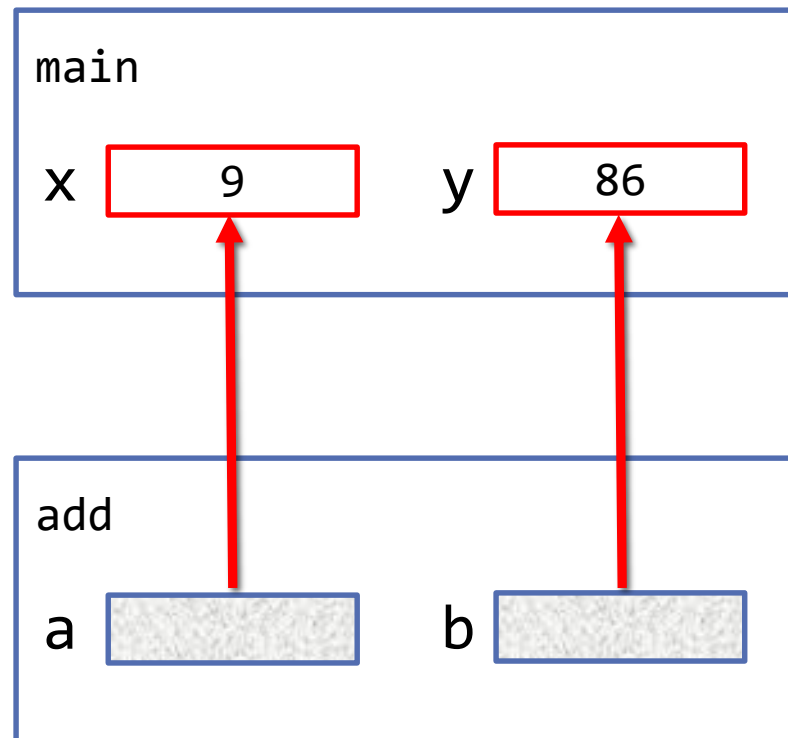
```
int add(int &a,int &b);
int main(){
    int x = 0, y = 0;
    x = 9;
    y = 86;
    cout << add(x,y) << endl;
    return 0;
}
int add(int &a,int &b){
    a++;
    ++b;
    return a+b;
}
```



- “引用” 可视为 “变量的别名” （速度快）

# 引用的缺点

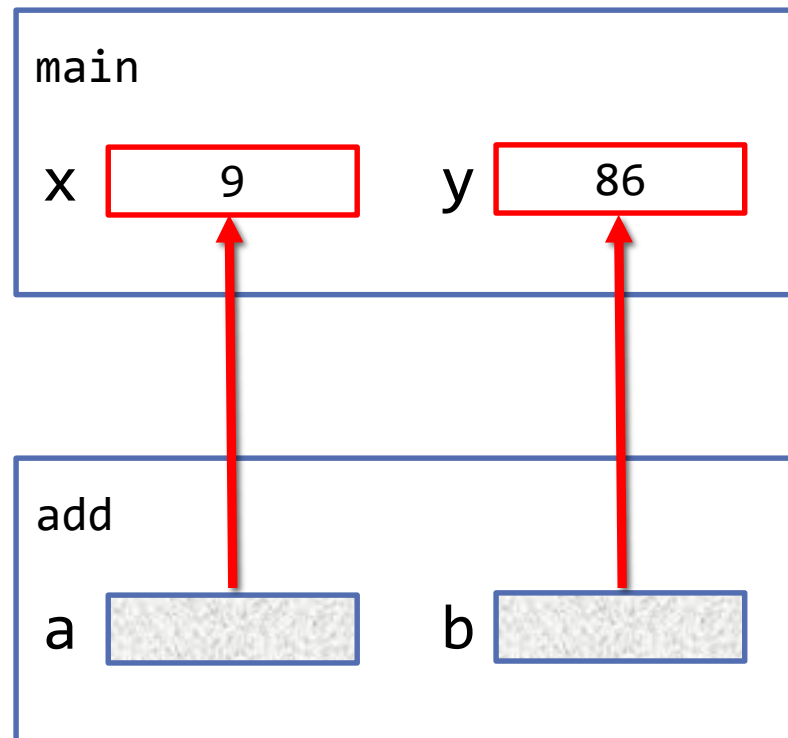
```
int add(int &a,int &b);
int main(){
    int x = 0, y = 0;
    x = 9;
    y = 86;
    cout << add(x,y) << endl;
    return 0;
}
int add(int &a,int &b){
    a++;
    ++b;
    return a+b;
}
```



- “引用” 可视为 “变量的别名”

# 引用的缺点

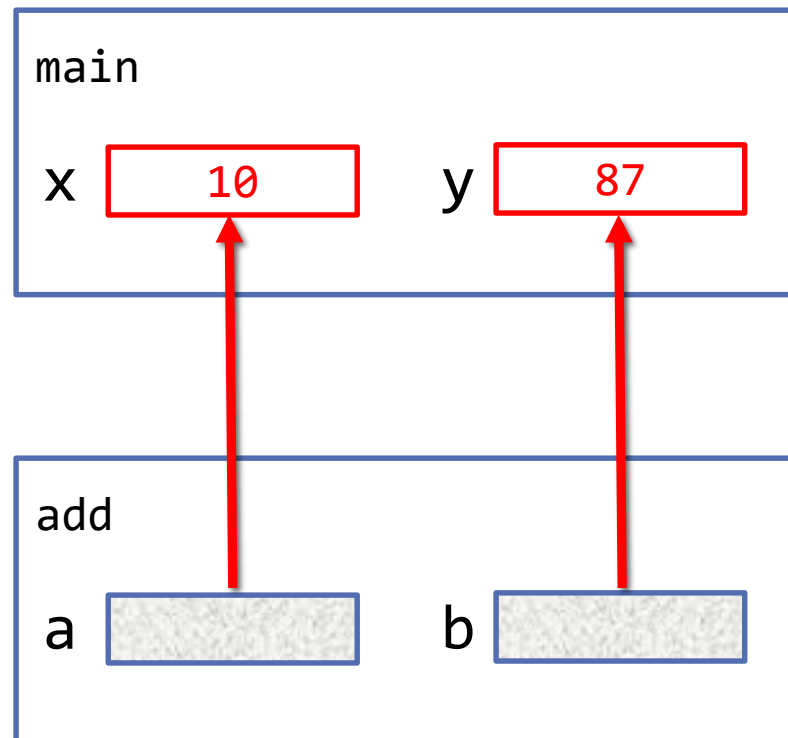
```
int add(int &a,int &b);
int main(){
    int x = 0, y = 0;
    x = 9;
    y = 86;
    cout << add(x,y) << endl;
    return 0;
}
int add(int &a,int &b){
    a++;
    ++b;
    return a+b;
}
```



- “引用” 可视为 “变量的别名” （可通过形参改变实参）

# 引用的缺点

```
int add(int &a,int &b);  
int main(){  
    int x = 0, y = 0;  
    x = 9;  
    y = 86;  
    cout << add(x,y) << endl;  
    return 0;  
}  
int add(int &a,int &b){  
    a++;  
    ++b;  
    return a+b;  
}
```



- “引用” 可视为 “变量的别名” （可通过形参改变实参）

# 避免“引用”的副作用

```
int add(const int &a, const int &b);  
int main(){  
    int x = 0, y = 0;  
    x = 9;  
    y = 86;  
    cout << add(x,y) << endl;  
    return 0;  
}
```

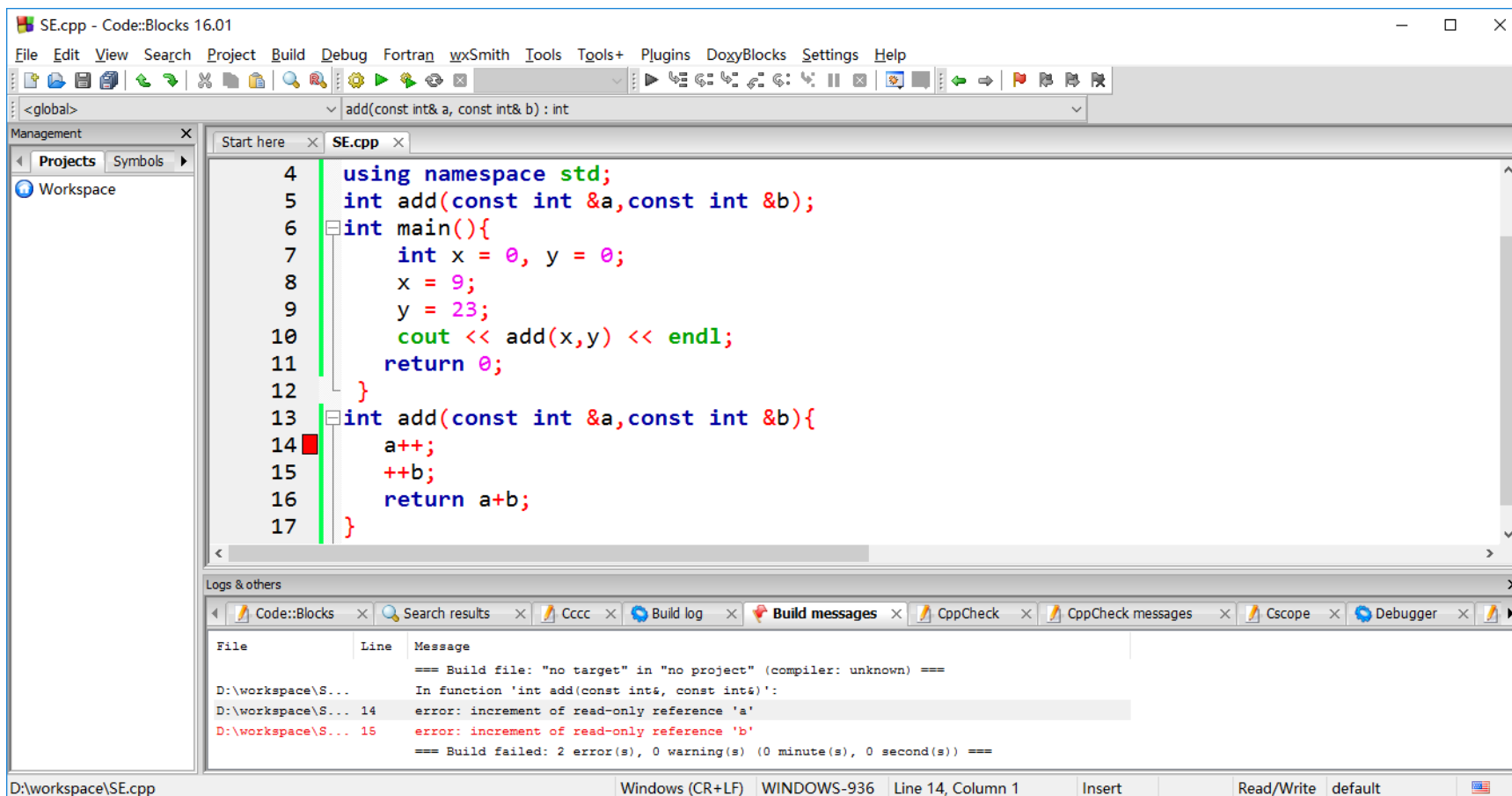
- 将参数声明为 `const &`（常量引用），可避免利用形参修改实参的副作用。

# 避免“引用”的副作用

```
int add(const int &a, const int &b);  
int main(){  
    int x = 0, y = 0;  
    x = 9;  
    y = 86;  
    cout << add(x,y) << endl;  
    return 0;  
}
```

- 将参数声明为 `const &`（常量引用），可避免利用形参修改实参的副作用。
- 通过“常量引用参数”修改实际参数，会引起语法错误。

# 避免“引用”的副作用



The screenshot shows the Code::Blocks IDE with a C++ file named SE.cpp. The code defines a function `add` that takes two constant integer references and increments them before returning their sum. The `main` function calls `add` with values 9 and 23. The compiler errors in the 'Build messages' window indicate that the increments on lines 14 and 15 are invalid because the references are constant.

```
4 using namespace std;
5 int add(const int &a, const int &b);
6 int main(){
7     int x = 0, y = 0;
8     x = 9;
9     y = 23;
10    cout << add(x, y) << endl;
11    return 0;
12 }
13 int add(const int &a, const int &b){
14     a++;
15     ++b;
16     return a+b;
17 }
```

Build messages:

```
=== Build file: "no target" in "no project" (compiler: unknown) ===
In function 'int add(const int&, const int&)':
D:\workspace\S... 14 error: increment of read-only reference 'a'
D:\workspace\S... 15 error: increment of read-only reference 'b'
=== Build failed: 2 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===
```

- 使用“常量引用参数”修改实际参数引起的语法错误



# 参数传递

```
int add(const int &a,const int &b);  
int main(){  
    int x = 0, y = 0;  
    x = 9;  
    y = 23;  
    cout << add(x,y) << endl;  
    return 0;  
}  
int add(const int &a,const int &b){  
    return a+b;  
}
```

- 地址传递

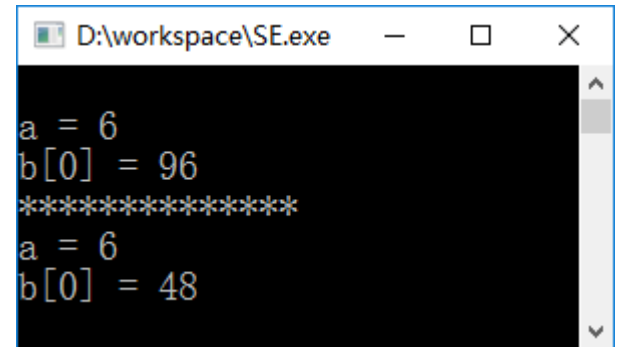
# 实参与形参间数据的两种传递方式

- **值传递**
  - **形参的变化不会影响实参**
- **地址传递**
  - **形参的变化会导致实参的改变**

# 实参与形参间数据的两种传递方式

```
void f(int x, int y[]);
int main(){
    int a = 6;
    int b[6] = {96,1,0,2,4,48};
    cout << "a = " << a << endl;
    cout << "b[0] = " << b[0] << endl;
    cout << "*****" << endl;
    f(a, b);
    cout << "a = " << a << endl;
    cout << "b[0] = " << b[0] << endl;
    return 0;
}

void f(int x,int y[]){
    x = 10;
    y[0] = 48;
}
```



The screenshot shows a window titled "D:\workspace\SE.exe" with a black background and white text. The output of the program is as follows:

```
a = 6
b[0] = 96
*****
a = 6
b[0] = 48
```

# 地址

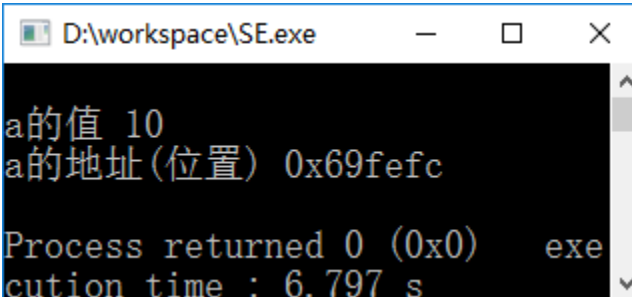
- 地址
  - 变量在内存中的位置
- 获取变量地址的方法
  - &变量名
- &
  - 取地址运算符
  - 获得目标变量的地址

# 取地址操作

```
int main(){
    int a = 10;

    cout << "a 的值 " << a << endl;
    cout << "a 的地址 (位置) " << &a << endl;

    return 0;
}
```

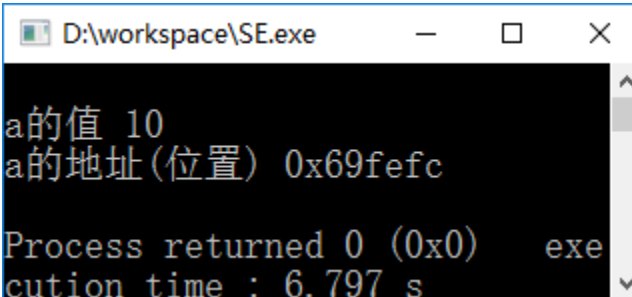


```
D:\workspace\SE.exe
a的值 10
a的地址(位置) 0x69fefc
Process returned 0 (0x0)
Execution time : 6.797 s
```

- &a
  - 获得变量 a 的地址

# 取地址操作

```
int main(){  
    int a = 10;  
  
    cout << "a 的值 " << a << endl;  
    cout << "a 的地址 (位置) " << &a << endl;  
  
    return 0;  
}
```

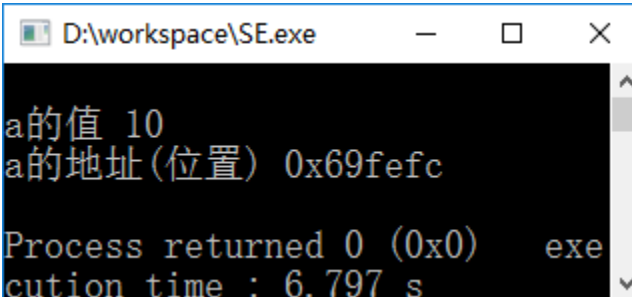


```
D:\workspace\SE.exe  
a的值 10  
a的地址(位置) 0x69fefc  
Process returned 0 (0x0)   exe  
Execution time : 6.797 s
```

- **变量 a 的地址**
  - **特殊的16进制数**

# 取地址操作

```
int main(){  
    int a = 10;  
  
    cout << "a 的值 " << a << endl;  
    cout << "a 的地址 (位置) " << &a << endl;  
  
    return 0;  
}
```



```
D:\workspace\SE.exe  
a的值 10  
a的地址(位置) 0x69fefc  
Process returned 0 (0x0)   exe  
Execution time : 6.797 s
```

- **变量 a 的地址**
  - 特殊的16进制数（不是整数）
  - 需要“特殊的变量”对“地址”进行处理

# 地址与指针

- 普通变量的地址
  - 特殊的16进制数
  - 不是整数
- 处理地址的“特殊变量”
  - 指针(变量)



# 指针变量的声明

- 语法

- 数据类型 \*指针变量名 [ = 初始值 ] ;

- \*

- 指针变量声明符
- 运算符重载

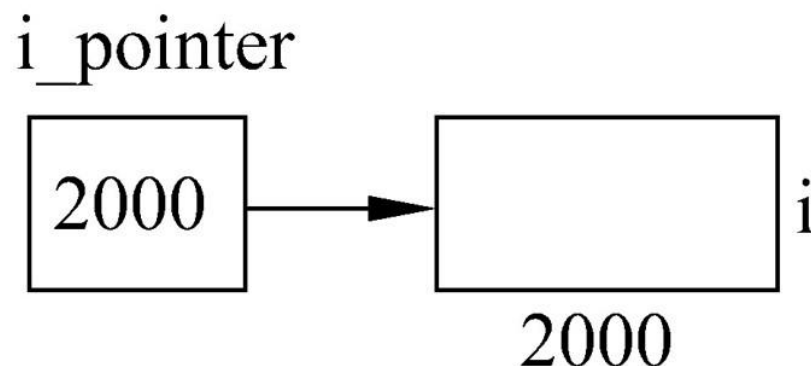
- 初始值

- 某个同类型普通变量的地址

# 指针定义

```
int i = 3 ;  
int *i_pointer = &i ;
```

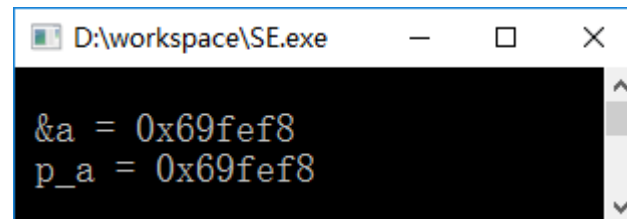
```
int i = 3 ;  
int *i_pointer ;  
i_pointer = &i ;
```



- 指针变量 `i_pointer` 保存了 普通变量 `i` 的地址
- 指针变量 `i_pointer` 指向了 普通变量 `i`
- 2000 是 变量 `i` 的地址

# 指针定义

```
int main() {  
    int a = 10;  
    int *p_a = &a; //声明指向变量a的指针变量p_a  
  
    //输出变量a的地址  
    cout << " &a = " << &a <<endl;  
    cout << " p_a = " << p_a <<endl;  
  
    return 0;  
}
```



```
D:\workspace\SE.exe  
&a = 0x69fef8  
p_a = 0x69fef8
```

- **整型指针 p\_a 指向了 整型变量 a**
- **数据类型必须一致**

# 指针

- 不能使用数值数据对指针变量赋(初)值
- 声明指针变量时必须指定其数据类型
- 指针变量只能保存具有相同数据类型的普通变量的地址
  - 不同类型的指针之间不存在自发的类型转换或者允许强制类型转换的发生

# 指针定义

The screenshot shows the Code::Blocks 16.01 IDE with a C++ file named SE.cpp. The code defines a variable 'a' of type 'int' and a pointer 'p\_a' of type 'double' pointing to 'a'. The compiler error message at the bottom states: "error: cannot convert 'int\*' to 'double\*' in initialization".

```
1  #include "iostream"
2  using namespace std;
3
4  int main()
5  {
6      int a = 10;
7      double *p_a = &a;
8      cout << " &a = " << &a << endl;
9      cout << " p_a = " << p_a << endl;
10
11     return 0;
12 }
13
14
```

Logs & others

File	Line	Message
=== Build file: "no target" in "no project" (compiler: unknown) ===		
In function 'int main()':		
D:\workspace\S...	7	error: cannot convert 'int*' to 'double*' in initialization
=== Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===		

- 指针只能保存同类型（普通）变量的地址

# 指针定义

The screenshot shows the Code::Blocks 16.01 IDE with a C++ file named SE.cpp. The code defines a main function that declares an integer variable 'a' and a double pointer 'p\_a'. It attempts to assign the address of 'a' to 'p\_a' and then prints the values. The compiler has generated an error message in the 'Build messages' window.

```
1  #include "iostream"
2  using namespace std;
3
4  int main()
5  {
6      int a = 10;
7      double *p_a = &a;
8      cout << " &a = " << &a << endl;
9      cout << " p_a = " << p_a << endl;
10
11     return 0;
12 }
13
14
```

Build messages:

File	Line	Message
=== Build file: "no target" in "no project" (compiler: unknown) ===		
In function 'int main()':		
D:\workspace\S...	7	error: cannot convert 'int*' to 'double*' in initialization
=== Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===		

- 双精度型指针 `p_a` 不能指向整型变量 `a`

# 指针的使用

- C++访问变量的两种方式

1. 直接访问

- 通过变量名直接操作变量

2. 间接访问

- 通过指向变量的指针操作变量
- 间接运算符 (\*)
  - 运算符重载

```
int a = 10;  
cout << a << endl;  
a = a + 1;  
cout << a << endl;
```

```
int *p = &a;  
cout << *p << endl;  
*p = *p + 1;  
cout << *p << endl;
```

# 通过指针间接访问变量

- 语法
  - \* 指针变量
- 间接引用指针
  - 可获得由该指针指向的变量内容（数据）
- 星号（\*）的三个用途
  - 运算符重载



# 通过指针间接访问变量

- 间接引用操作符
  - \* 放在可执行语句中的指针之前
- 指针定义符
  - \* 放在指针声明语句中的指针之前
- 乘法运算符
- 非指针变量不能使用间接引用操作符

# 通过指针间接访问变量

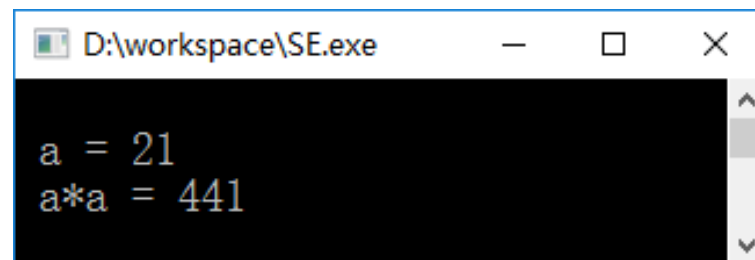
```
int main(){
    int a = 101;
    int *p_a = &a;

    a = 18; //直接访问

    //改变 *p_a 就是改变 a （间接访问）
    *p_a = 21;

    cout << " a = " << a << endl;
    cout << " a*a = " << a * a << endl;

    return 0;
}
```



```
D:\workspace\SE.exe
a = 21
a*a = 441
```

# 通过指针间接访问变量

```
int main(){  
    int a = 101;  
    int *p_a = &a;
```

a = 18; //直接访问

//改变 \*p\_a 就是改变 a (间接访问)

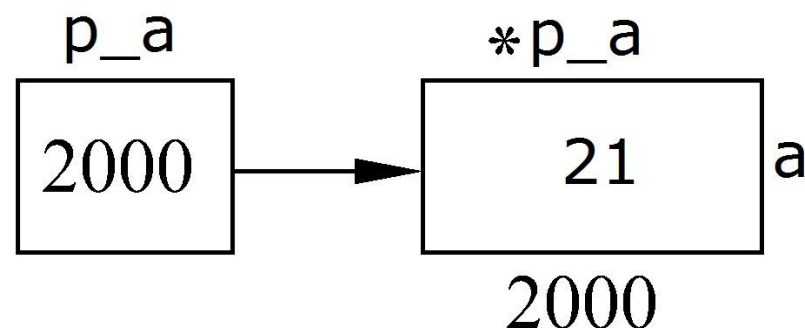
```
*p_a = 21;
```

```
cout << " a = " << a << endl;
```

```
cout << " a*a = " << a * a << endl;
```

```
return 0;
```

```
}
```



- 指针 p\_a 指向了 变量 a

# 通过指针间接访问变量

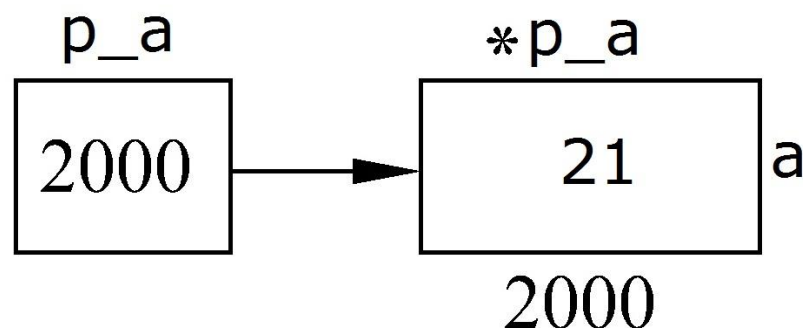
```
int main(){
    int a = 101;
    int *p_a = &a;

    a = 18; //直接访问

    //改变 *p_a 就是改变 a （间接访问）
    *p_a = 21;

    cout << " a = " << a << endl;
    cout << " a*a = " << a * a << endl;

    return 0;
}
```



- 声明指针变量

# 通过指针间接访问变量

```
int main(){  
    int a = 101;  
    int *p_a = &a;  
  
    a = 18; //直接访问
```

//改变 \*p\_a 就是改变 a （间接访问）

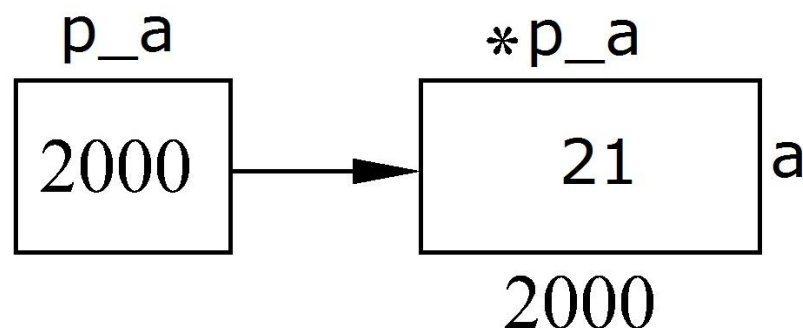
```
*p_a = 21;
```

```
cout << " a = " << a << endl;
```

```
cout << " a*a = " << a * a << endl;
```

```
return 0;
```

```
}
```



- 间接访问

# 通过指针间接访问变量

```
int main(){  
    int a = 101;  
    int *p_a = &a;  
  
    a = 18; //直接访问
```

//改变 \*p\_a 就是改变 a （间接访问）

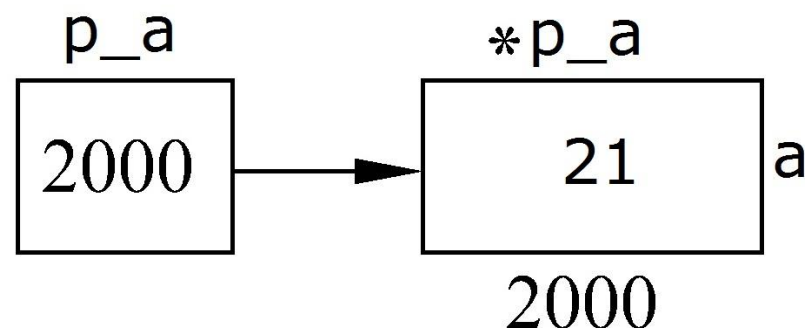
```
*p_a = 21;
```

```
cout << " a = " << a << endl;
```

```
cout << " a*a = " << a * a << endl;
```

```
return 0;
```

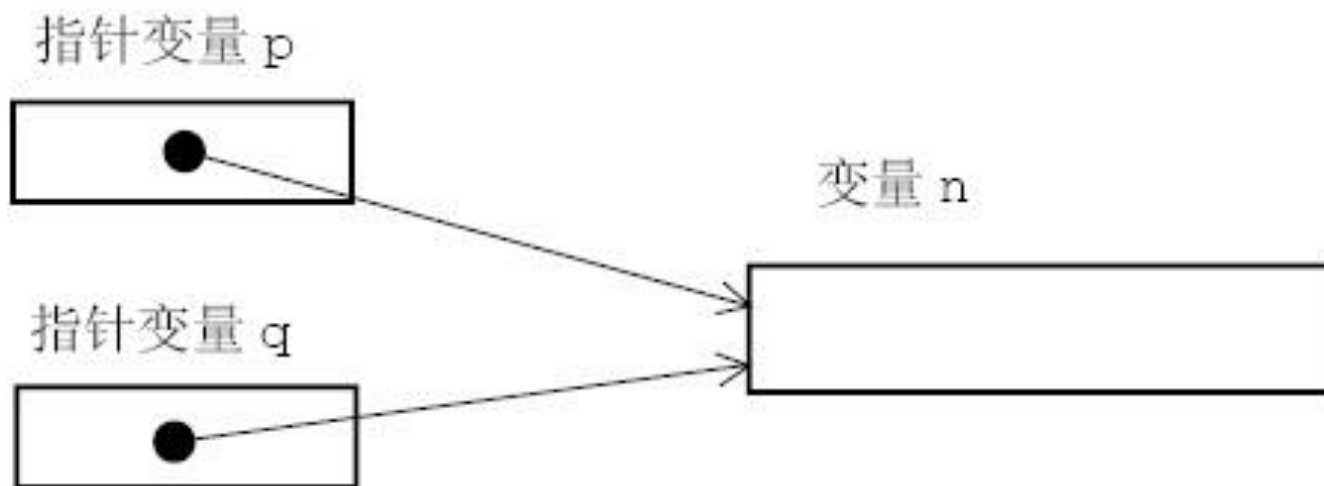
```
}
```



## • 乘法运算

# 无意义的指针比较

- 判断两个指针是否相等
  - 判断两个指针是否指向同一个变量

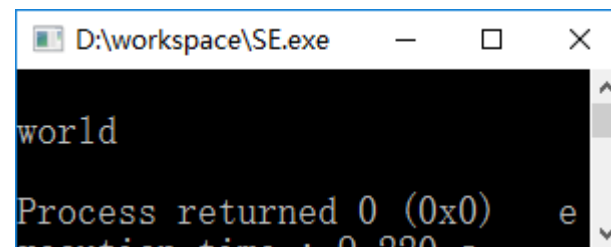


# 指针

```
int main(){
    int a=0, b=0;
    int *p1 = &a, *p2;
    p2 = &b;

    if(p1 == p2){
        cout<<"hello"<<endl;
    }

    if(*p1 == *p2){
        cout<<"world"<<endl;
    }
    return 0;
}
```



```
D:\workspace\SE.exe
world
Process returned 0 (0x0)
Execution time: 0.220 s
```

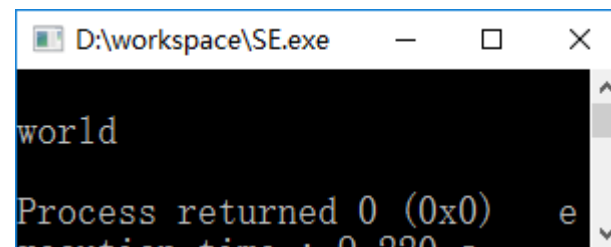


# 指针

```
int main(){
    int a=0, b=0;
    int *p1 = &a, *p2;
    p2 = &b;

    if(p1 == p2){
        cout<<"hello"<<endl;
    }

    if(*p1 == *p2){
        cout<<"world"<<endl;
    }
    return 0;
}
```



```
D:\workspace\SE.exe
world
Process returned 0 (0x0)
Execution time: 0.020 s
```

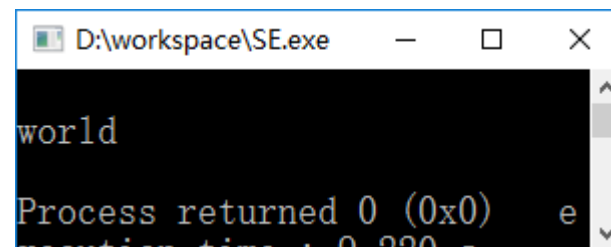
判断p1和p2是否  
指向同一个变量

# 指针

```
int main(){
    int a=0, b=0;
    int *p1 = &a, *p2;
    p2 = &b;

    if(p1 == p2){
        cout<<"hello"<<endl;
    }

    if(*p1 == *p2){
        cout<<"world"<<endl;
    }
    return 0;
}
```



```
D:\workspace\SE.exe
world
Process returned 0 (0x0)
```

判断 a 和 b 的  
变量值是否相同

# 指针运算符

- **&**
  - **取地址运算符**
- **\***
  - **指针声明符**
  - **间接访问符**
- **优先级**      **与一元运算符相同**
- **结合性**      **右结合性(自右向左结合)**

# 指针运算

## 代码

- `int *p1 , a;`
- `a = 101 ;`
- `p1 = &a ;`
- `cout<< *&p1 <<endl ;`
- `cout<< *&a <<endl ;`

## 等价形式

- `int *p1 , a;`
- `a = 101 ;`
- `p1 = &a ;`
- `cout<< &(*p1) <<endl;`
- `cout<< *(&a) <<endl;`

# nullptr

- **空指针**
  - nullptr
  - 可用于对指针变量进行初始化
  - C++ 11
- **Before**
  - NULL

# 指针运算

- 指针运算
  - 加法运算
  - 减法运算
- 控制指针在内存中的移动
- 只有加法和减法可用于指针运算

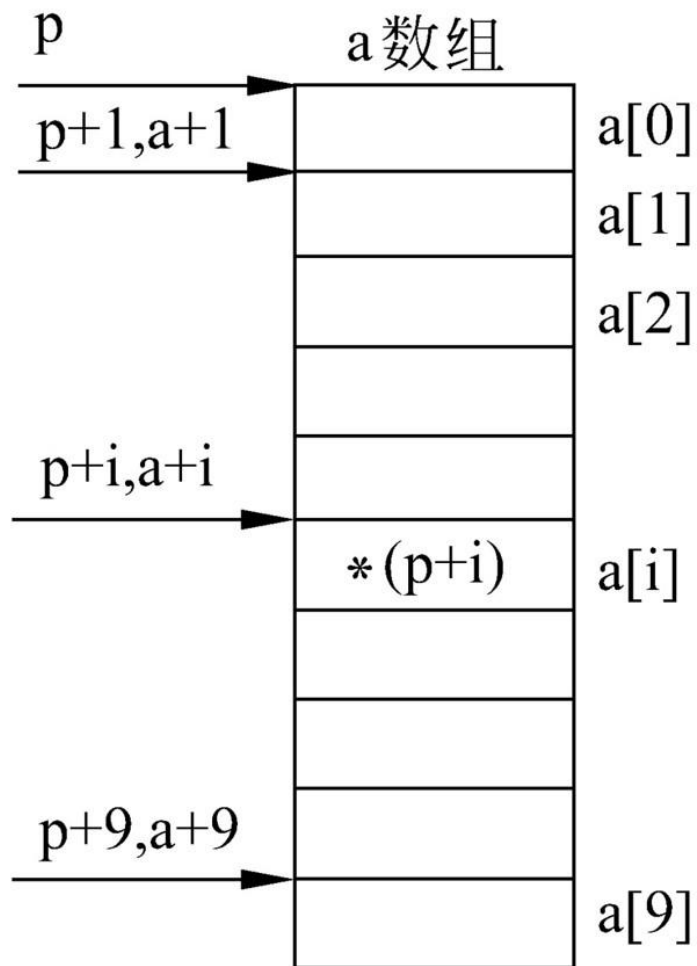
# 指针运算

- 考虑特定类型  $T$  的指针变量  $p$

$$p \pm i = p \pm i \times d$$

- $i$  整型
  - $d$  相应数据类型  $T$  所占的字节数
- 
- 适用于数组元素的移动
    - 数组元素首尾相连

# 指针运算与数组



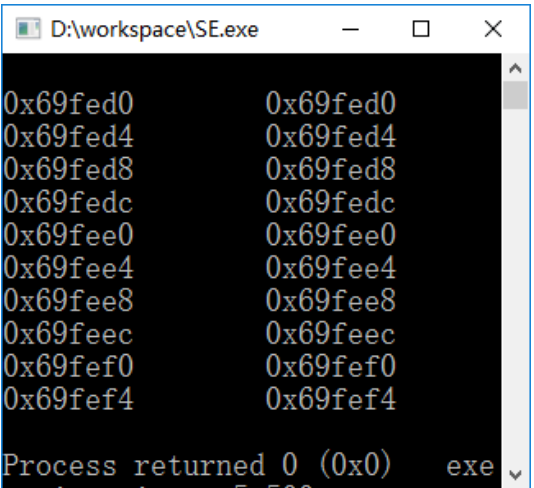
- 数组元素首尾相连
- 每个数组元素具有相同的数据类型
  - 所占的内存空间大小一致
- 通过指针运算实现数组元素的遍历



# 指针运算与数组

```
int main(){
    int a[10]={1,2,3,4,5,6,7,8,9,10};
    int *p = &a[0];
    for(int i = 0; i < 10; ++i){
        cout << &a[i] << "\t" << p << endl;
        p = p + 1;
    }

    return 0;
}
```

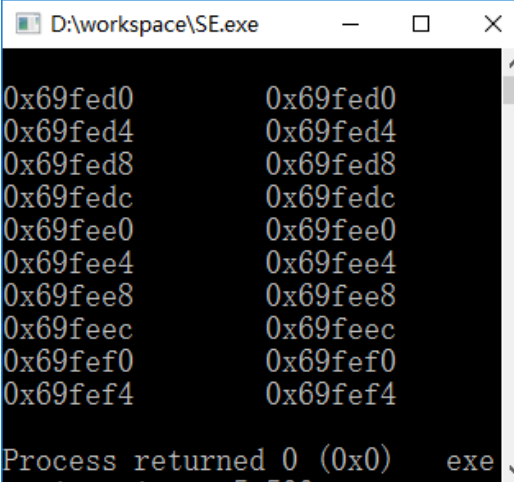


```
D:\workspace\SE.exe
0x69fed0      0x69fed0
0x69fed4      0x69fed4
0x69fed8      0x69fed8
0x69fedc      0x69fedc
0x69fee0      0x69fee0
0x69fee4      0x69fee4
0x69fee8      0x69fee8
0x69feec      0x69feec
0x69fef0      0x69fef0
0x69fef4      0x69fef4
Process returned 0 (0x0)   exe
```

# 指针运算与数组

```
int main(){
    int a[10]={1,2,3,4,5,6,7,8,9,10};
    int *p = &a[0];
    for(int i = 0; i < 10; ++i){
        cout << &a[i] << "\t" << p << endl;
        p = p + 1;
    }

    return 0;
}
```



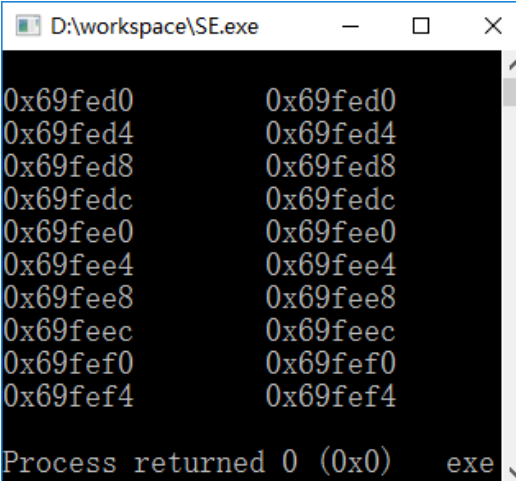
```
D:\workspace\SE.exe
0x69fed0      0x69fed0
0x69fed4      0x69fed4
0x69fed8      0x69fed8
0x69fedc      0x69fedc
0x69fee0      0x69fee0
0x69fee4      0x69fee4
0x69fee8      0x69fee8
0x69feec      0x69feec
0x69fef0      0x69fef0
0x69fef4      0x69fef4
Process returned 0 (0x0)   exe
```

- **整型指针 p 指向整型数组 a 的首元素 (a[0])**

# 指针运算与数组

```
int main(){
    int a[10]={1,2,3,4,5,6,7,8,9,10};
    int *p = &a[0];
    for(int i = 0; i < 10; ++i){
        cout << &a[i] << "\t" << p << endl;
        p = p + 1;
    }

    return 0;
}
```



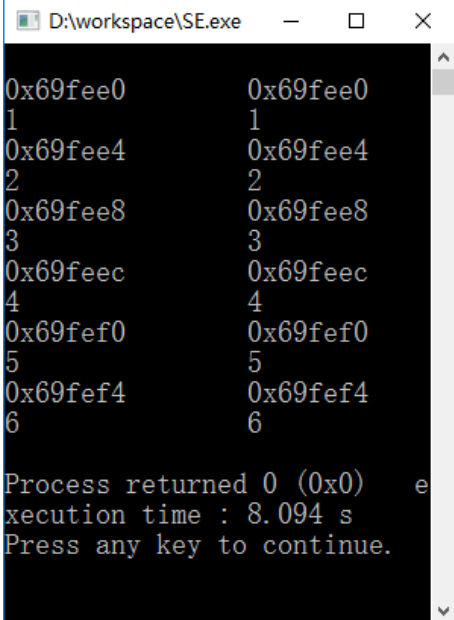
```
D:\workspace\SE.exe
0x69fed0      0x69fed0
0x69fed4      0x69fed4
0x69fed8      0x69fed8
0x69fedc      0x69fedc
0x69fee0      0x69fee0
0x69fee4      0x69fee4
0x69fee8      0x69fee8
0x69feec      0x69feec
0x69fef0      0x69fef0
0x69fef4      0x69fef4
Process returned 0 (0x0) exe
```

- **整型指针 p 指向整型数组 a 的下一个元素**

# 指针运算与数组

```
int main(){
    int a[6]={1,2,3,4,5,6};
    int *p = &a[0];

    for(int i=0; i<6; ++i){
        cout << &a[i] << "\t" << p << endl;
        cout << a[i] << "\t\t" << *p << endl;
        p = p + 1;
    }
    return 0;
}
```



```
D:\workspace\SE.exe
0x69fee0      0x69fee0
1             1
0x69fee4      0x69fee4
2             2
0x69fee8      0x69fee8
3             3
0x69feec      0x69feec
4             4
0x69fef0      0x69fef0
5             5
0x69fef4      0x69fef4
6             6
Process returned 0 (0x0)
Execution time : 8.094 s
Press any key to continue.
```

# 指针运算与数组

```
int main(){
    int a[6]={1,2,3,4,5,6};
    int *p = &a[0];

    for(int i=0; i<6; ++i){
        cout << &a[i] << "\t" << p << endl;
        cout << a[i] << "\t\t" << *p << endl;
        p = p + 1;
    }
    return 0;
}
```

- **整型指针 p 指向整型数组 a 的首元素 (a[0])**

# 指针运算与数组

```
int main(){
    int a[6]={1,2,3,4,5,6};
    int *p = &a[0];

    for(int i=0; i<6; ++i){
        cout << &a[i] << "\t" << p << endl;
        cout << a[i] << "\t\t" << *p << endl;
        p = p + 1;
    }
    return 0;
}
```

- 采用两种方式输出数组 a 当前元素 (a[i]) 的地址

# 指针运算与数组

```
int main(){
    int a[6]={1,2,3,4,5,6};
    int *p = &a[0];

    for(int i=0; i<6; ++i){
        cout << &a[i] << "\t" << p << endl;
        cout << a[i] << "\t\t" << *p << endl;
        p = p + 1;
    }
    return 0;
}
```

- 采用两种方式输出数组 a 当前元素 (a[i]) 的值

# 指针运算与数组

```
int main(){
    int a[6]={1,2,3,4,5,6};
    int *p = &a[0];

    for(int i=0; i<6; ++i){
        cout << &a[i] << "\t" << p << endl;
        cout << a[i] << "\t\t" << *p << endl;
        p = p + 1;
    }
    return 0;
}
```

- **整型指针 p 指向整型数组 a 的下一个元素**



# 指针与数组

- `int a[10] = {1,2,3,4,5,6,7,8,9,10};`
- `int *p = &a[0];`
- `int *q = a;`
- **p 和 q 同时指向数组 a 的首元素**
- **数组名代表数组中第一个元素(首元素)的地址**
  - **不能被修改**
  - **指针常量 (指针本身是常量, 而非指向的对象是常量)**

# 指针与数组

- **数组名**
  - **代表数组的起始地址（第一个元素的地址）**
  - **可以把数组起始地址赋给一个指针（初始化指针）**
  - **通过移动指针（加减指针）来对数组元素进行操作**
- **数组名是指针常量**
  - **不能被修改**

# 指针与数组

- 引用数组元素的两种方式

- 下标法

- 数组名[索引值]

- a[2]


- 指针法

- \*(起始地址 + 偏离量)

- \*(a + 2)

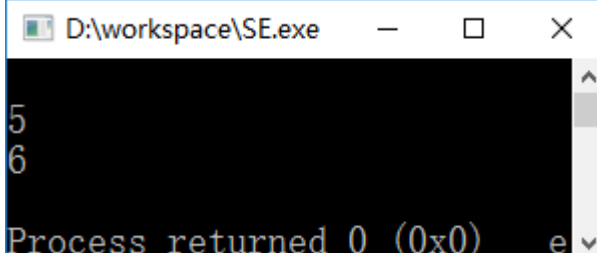
```
int main(){  
  
    int a[6]={1,2,3,4,5,6};  
  
    cout << a[2] << endl;  
    cout << *(a + 2) << endl;  
  
    return 0;  
}
```

# 指针与数组




```
int main(){
    int a[10] = { 1,2,3,6,7,8,9,10 } ;
    int *p = nullptr ;
    p = &a[2];
    *p = *p + 2 ;
    p = p + 1 ;

    cout << a[2] << endl;
    cout << *p << endl;
    return 0;
}
```



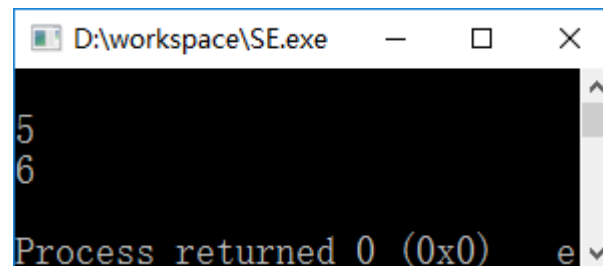
```
D:\workspace\SE.exe
5
6
Process returned 0 (0x0) e
```

# 指针与数组



```
int main(){
    int a[10] = { 1,2,3,6,7,8,9,10 } ;
    int *p = nullptr ;
    p = &a[2];
    *p = *p + 2 ;
    p = p + 1 ;


    cout << a[2] << endl;
    cout << *p << endl;
    return 0;
}
```



```
D:\workspace\SE.exe
5
6
Process returned 0 (0x0) e
```

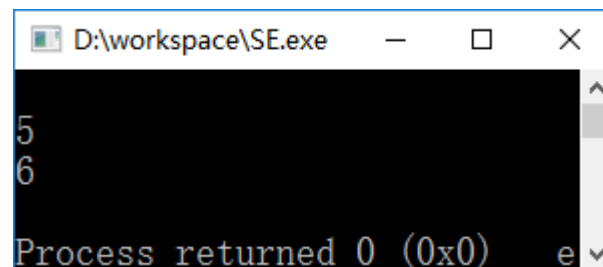
- 修改 a[2] 的值

# 指针与数组



```
int main(){
    int a[10] = { 1,2,3,6,7,8,9,10 } ;
    int *p = nullptr ;
    p = &a[2];
    *p = *p + 2 ;
    p = p + 1 ;


    cout << a[2] << endl;
    cout << *p << endl;
    return 0;
}
```



```
D:\workspace\SE.exe
5
6
Process returned 0 (0x0)
```

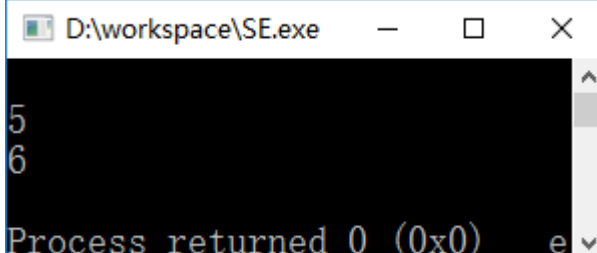
- **p + 1 下一个数组元素的地址**

# 指针与数组



```
int main(){
    int a[10] = { 1,2,3,6,7,8,9,10 } ;
    int *p = nullptr ;
    p = &a[2];
    *p = *p + 2 ;
    p = p + 1 ;

    cout << a[2] << endl;
    cout << *p << endl;
    return 0;
}
```



- **p + 1 下一个数组元素的地址**

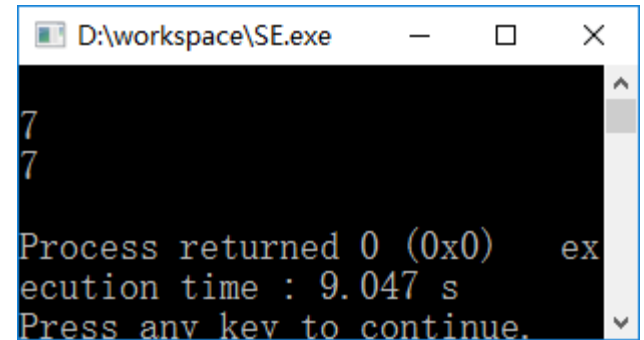
# 指针与数组

```
int main(){
    int a[10] = { 1,2,3,6,8,7,9,10 } ;
    int *p = nullptr ;
    p = &a[2];
    *p = *p + 3 ;
    *p = *(p + 3) ;

    cout << a[2] << endl;
    cout << *p << endl;

    return 0;
}
```

- **区别:**



```
D:\workspace\SE.exe
7
7
Process returned 0 (0x0)
ecution time : 9.047 s
Press any key to continue.
```



# 指针与数组

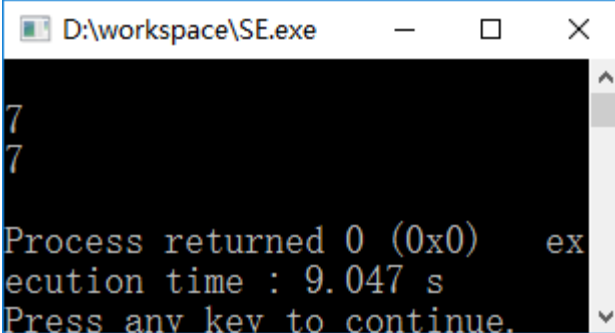
```
int main(){
    int a[10] = { 1,2,3,6,8,7,9,10 } ;
    int *p = nullptr ;
    p = &a[2];

    *p = *p + 3 ;
    *p = *(p + 3) ;

    cout << a[2] << endl;
    cout << *p << endl;


    return 0;
}
```

- **区别：小括号改变运算符的优先级**



```
D:\workspace\SE.exe
7
7
Process returned 0 (0x0)
ecution time : 9.047 s
Press any key to continue.
```

# 指针与数组



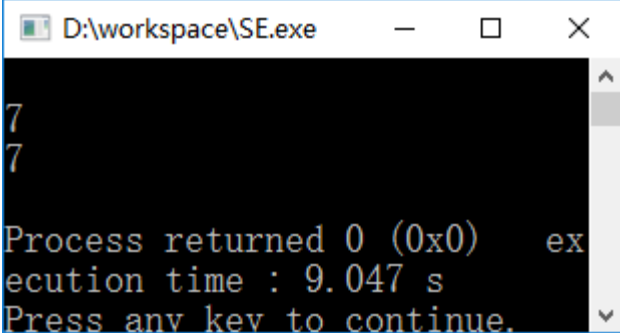
```
int main(){
    int a[10] = { 1,2,3,6,8,7,9,10 } ;
    int *p = nullptr ;
    p = &a[2];

    *p = *p + 3 ;

    *p = *(p + 3) ;

    cout << a[2] << endl;
    cout << *p << endl;


    return 0;
}
```



```
D:\workspace\SE.exe
7
7
Process returned 0 (0x0)
Execution time : 9.047 s
Press any key to continue.
```

- **p 指向数组 a 的第三个元素 (a[2])**

# 指针与数组



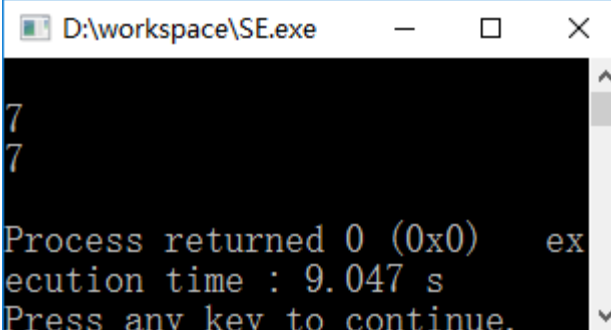
```
int main(){
    int a[10] = { 1,2,3,6,8,7,9,10 } ;
    int *p = nullptr ;
    p = &a[2];

    *p = *p + 3 ;

    *p = *(p + 3) ;

    cout << a[2] << endl;
    cout << *p << endl;

    return 0;
}
```



```
D:\workspace\SE.exe
7
7
Process returned 0 (0x0)
Execution time : 9.047 s
Press any key to continue.
```

- **p + 3 后续第三个数组元素 (a[2 + 3]) 的地址**

# 指针与数组

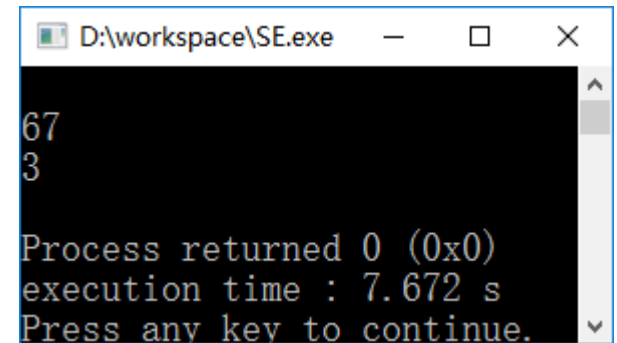
```
int main(){  
    int a[10] = { 1,2,3,6,8,7,9,10 } ;  
    int *p = nullptr ;  
    p = &a[2];  
    *p = *p + 3 ;  
    *p = *(p + 3) ;  
  
    cout << a[2] << endl;  
    cout << *p << endl;  
  
    return 0;  
}
```



- **p + 3 后续第三个数组元素 (a[5]) 的地址**

# 指针与数组

```
int main(){
    int a[10] = { 1,2,3,4,5,6,7,8} ;
    int *p = nullptr;
    p = &a[3];
    p = p + 2 ;
    cout<< *p << *(p + 1) <<endl;
    p = p - 3 ;
    cout<< *p <<endl;
    return 0;
}
```

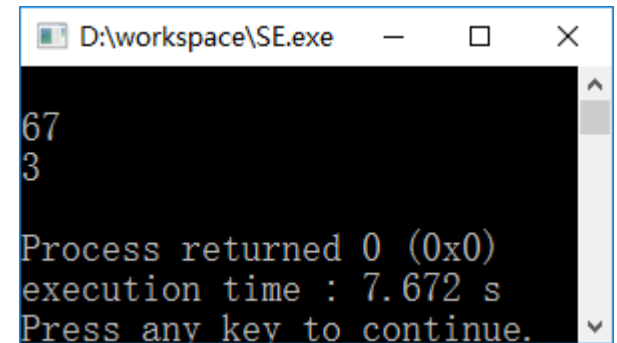


```
D:\workspace\SE.exe
67
3
Process returned 0 (0x0)
execution time : 7.672 s
Press any key to continue.
```

- **指针加法：向数组尾部移动**

# 指针与数组

```
int main(){
    int a[10] = { 1,2,3,4,5,6,7,8} ;
    int *p = nullptr;
    p = &a[3];
    p = p + 2 ;
    cout<< *p << *(p + 1) <<endl;
    p = p - 3 ;
    cout<< *p <<endl;
    return 0;
}
```



```
D:\workspace\SE.exe
67
3
Process returned 0 (0x0)
execution time : 7.672 s
Press any key to continue.
```

- **指针减法：向数组头部移动**

# 指针与数组

## 代码

- `int array[10] = { . . . };`
- `int sum = 0;`
- `int *iptr = array;`
- `for(int i=0; i<10; ++i){`
- `sum += *iptr;`
- `iptr++;`
- `}`

## 等价语句

- `sum += *(iptr++);`

## 更常见的等价形式

- `int array[10] = { . . . };`
- `int sum = 0;`
- `int *iptr = array;`
- `for(int i=0; i<10; ++i){`
- `sum += *iptr++;`
- `}`

## 可省略小括号

- `++` 与 `*` 优先级相同
- 都是右结合

# 任务肆拾肆

- 编写程序，计算平均成绩。
  - 人数不确定
    - 在程序运行时，由用户确定。
- 静态一维数组不适用
  - 无法事先规定数组的大小（整型常量）
- 解决方法
  - 动态一维数组



# 动态管理内存：动态一维数组

- 动态分配内存
  - new 运算符
- 释放内存
  - delete 运算符
- 动态分配内存之后，必须编码显式释放内存。

# 管理动态数组

- 动态分配内存
  - `Type *指针变量 = new Type[数组大小];`
- 释放内存
  - `delete[] 指针变量;`
- 只要求数组大小是整型值即可
- 指针变量视作数组名
- 与静态数组使用方式相同

# 创建动态一维数组

```
int abc[12] ;

int size = 0 ;
cin>>size;

int *cd = new int[size];
for(int i=0; i<size; ++i){
    cin >> cd[i] ;
}

delete[] cd;
```

- **静态一维整型数组**                      abc
- **动态一维整型数组**                      cd

# 动态管理内存

- 动态分配内存

- `Type *指针变量 = new Type(初值);`

- 释放内存

- `delete 指针变量;`

- 单变量

# 动态管理内存

- 动态分配内存

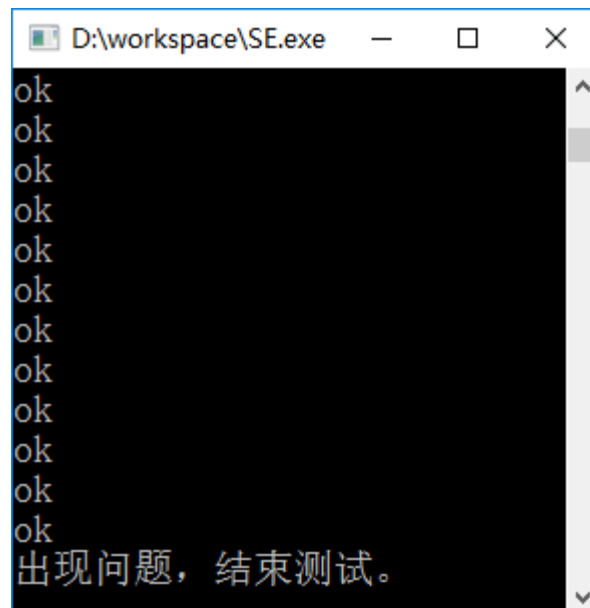
- `Type *指针变量 = new Type(初值);`
- `Type *指针变量 = new Type[数组大小];`

- 无法正常分配内存时

- Before `new` 返回 `NULL`
- Now (C++11) 抛出异常

# 动态管理内存

```
int main(){
    while(true){
        try{
            int *p = new int[1000000];
        }
        catch(...){
            cout << "出现问题，结束测试。" << endl;
            break;
        }
        cout << "ok" << endl;
    }
    return 0;
}
```



```
D:\workspace\SE.exe
ok
ok
ok
ok
ok
ok
ok
ok
ok
ok
ok
ok
ok
ok
出现问题，结束测试。
```

- **错误原因：未释放动态内存（缺少delete[]语句）。**

# 异常处理

- 语法

```
try{
```

在运行过程中，可能出现问题（产生异常）的代码；

```
}
```

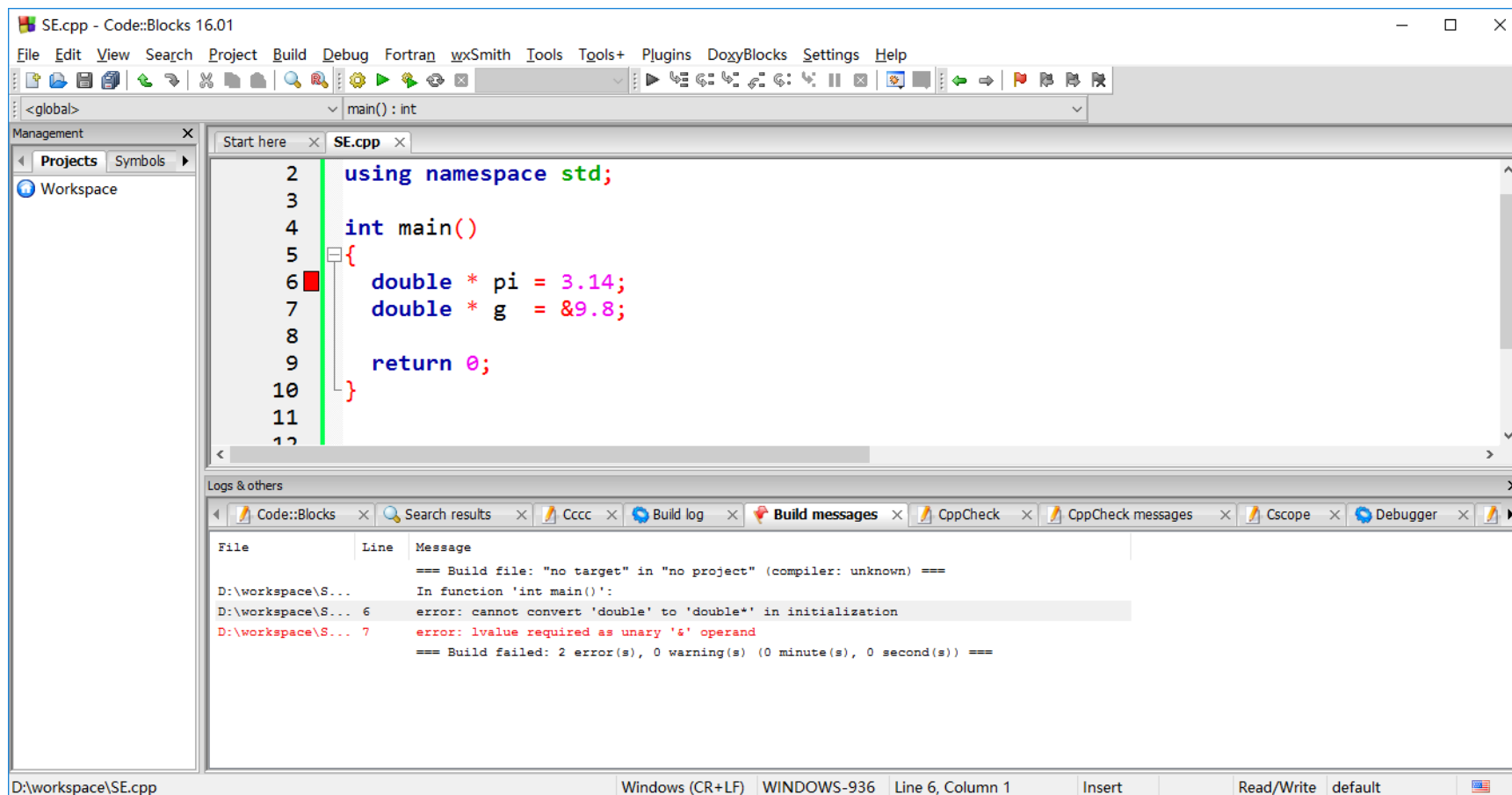
```
catch(...){
```

当特定问题发生的时候，解决该问题的代码；

```
}
```

- ... 代表所有问题（全体异常）

# 指针定义



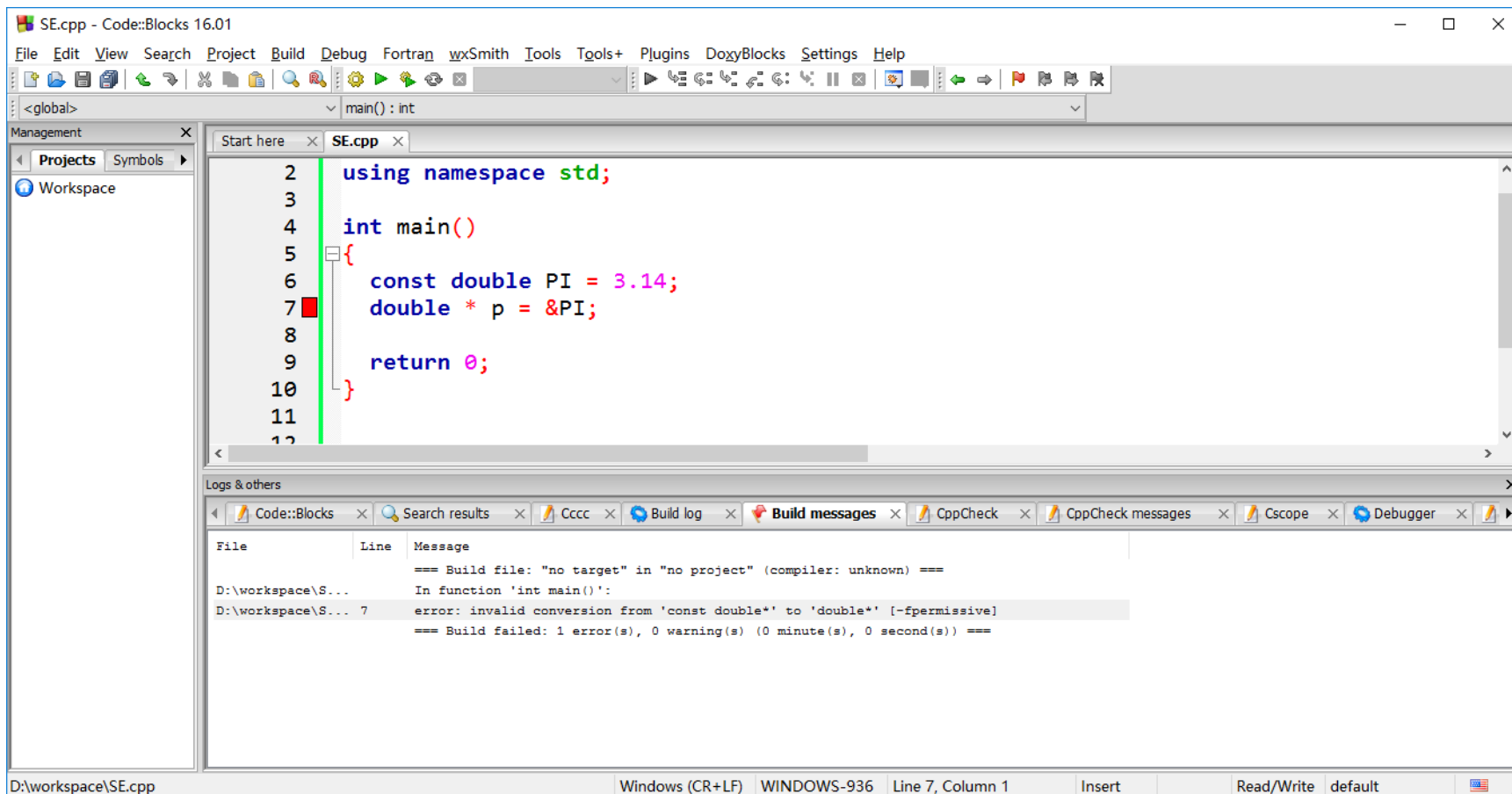
- 指针不能指向字面常量



# 指针定义

- 指针不能指向字面常量
- 指针不能指向 `const` 常量
  - 可以通过指针修改所指向变量的数据
  - `const`常量的值不能被修改

# 指针定义

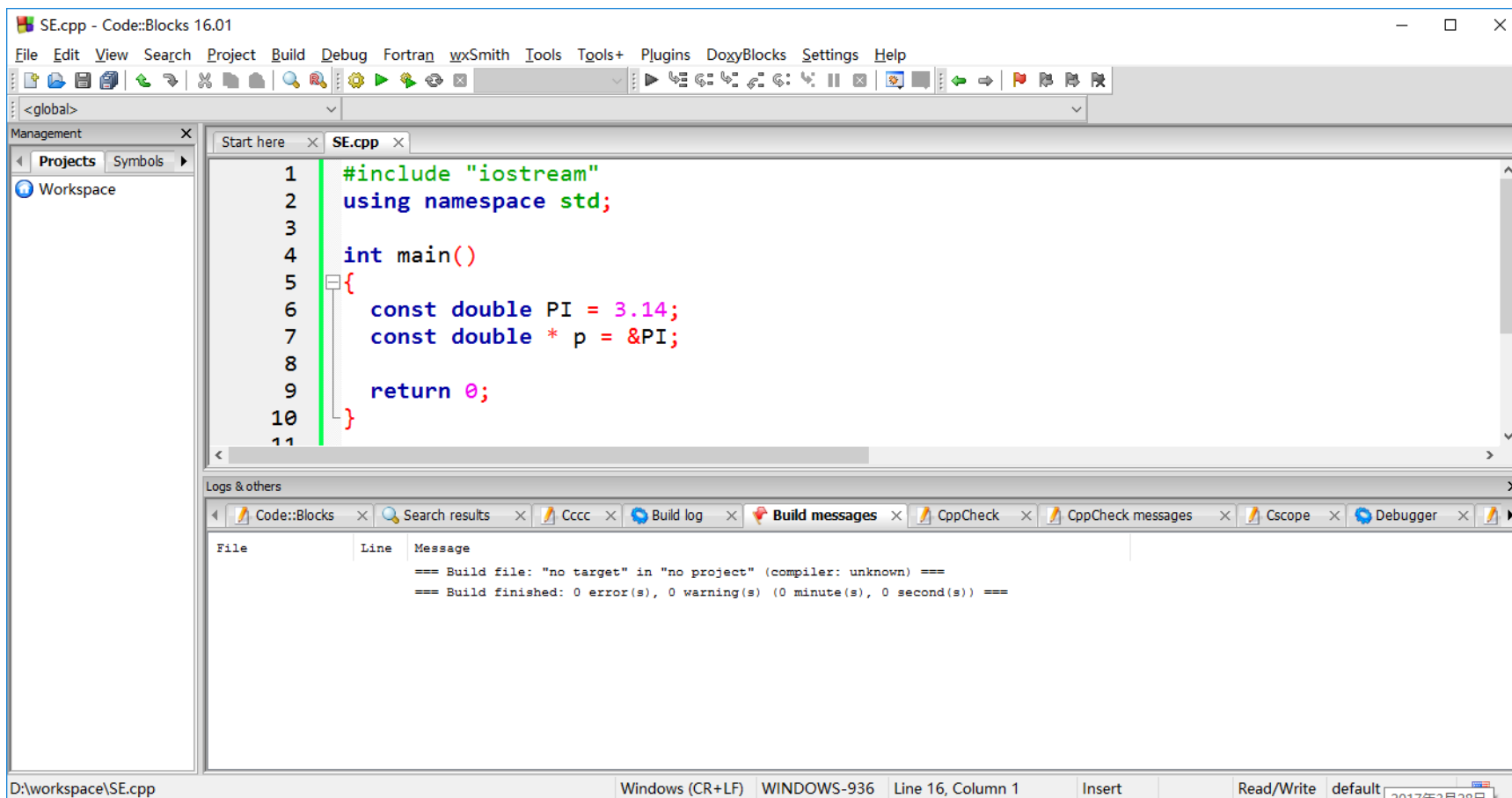


- 指针指向const常量（语法错误）

# 指针定义

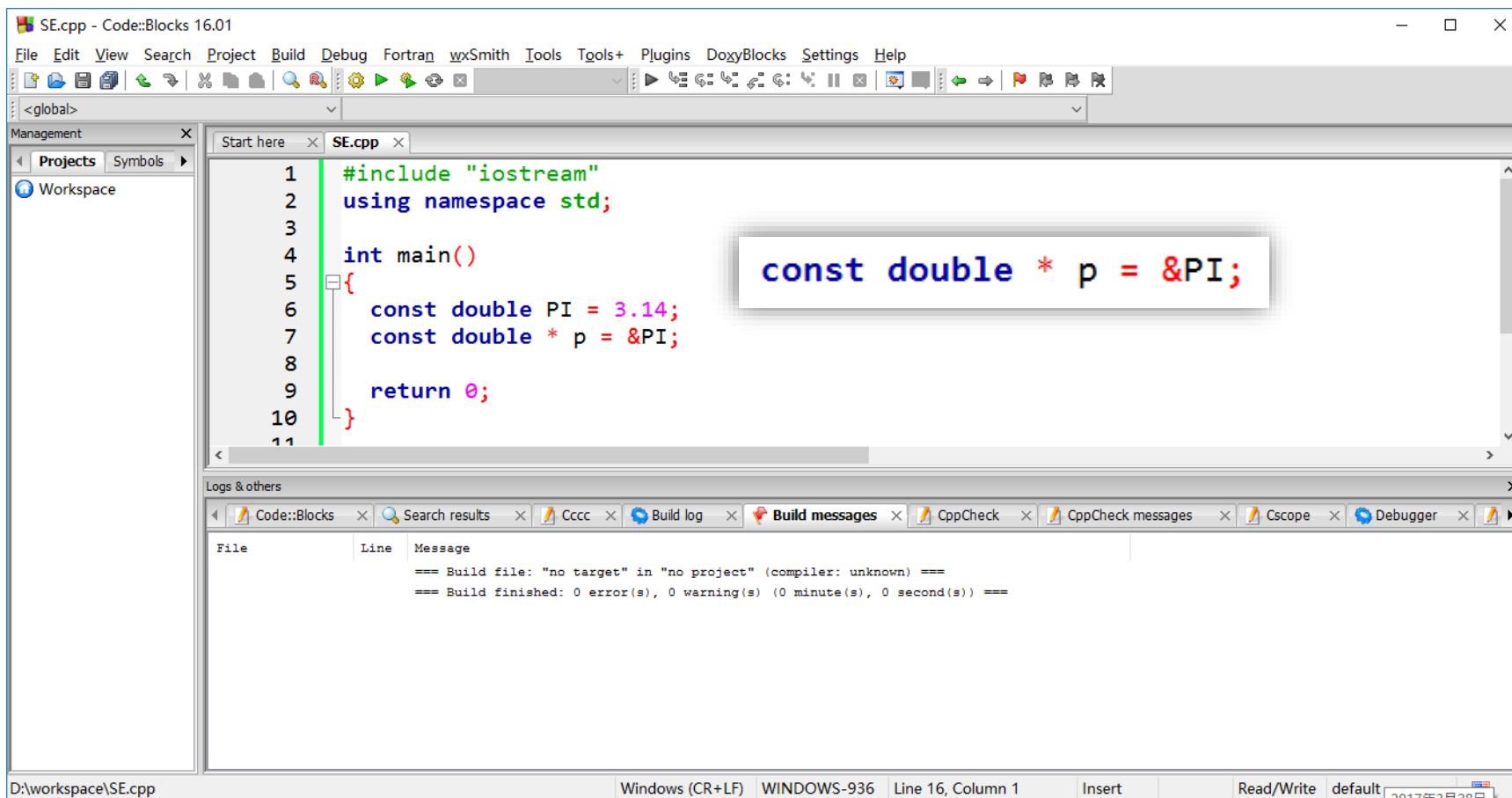
- 指针不能指向字面常量
- 指针不能指向 `const` 常量
  - 可以通过指针修改所指向变量的数据
  - `const`常量的值不能被修改
- 指向常量的指针
  - `const int * icp;`
    - 不能通过指针修改所指向变量的数据

# 指向常量的指针



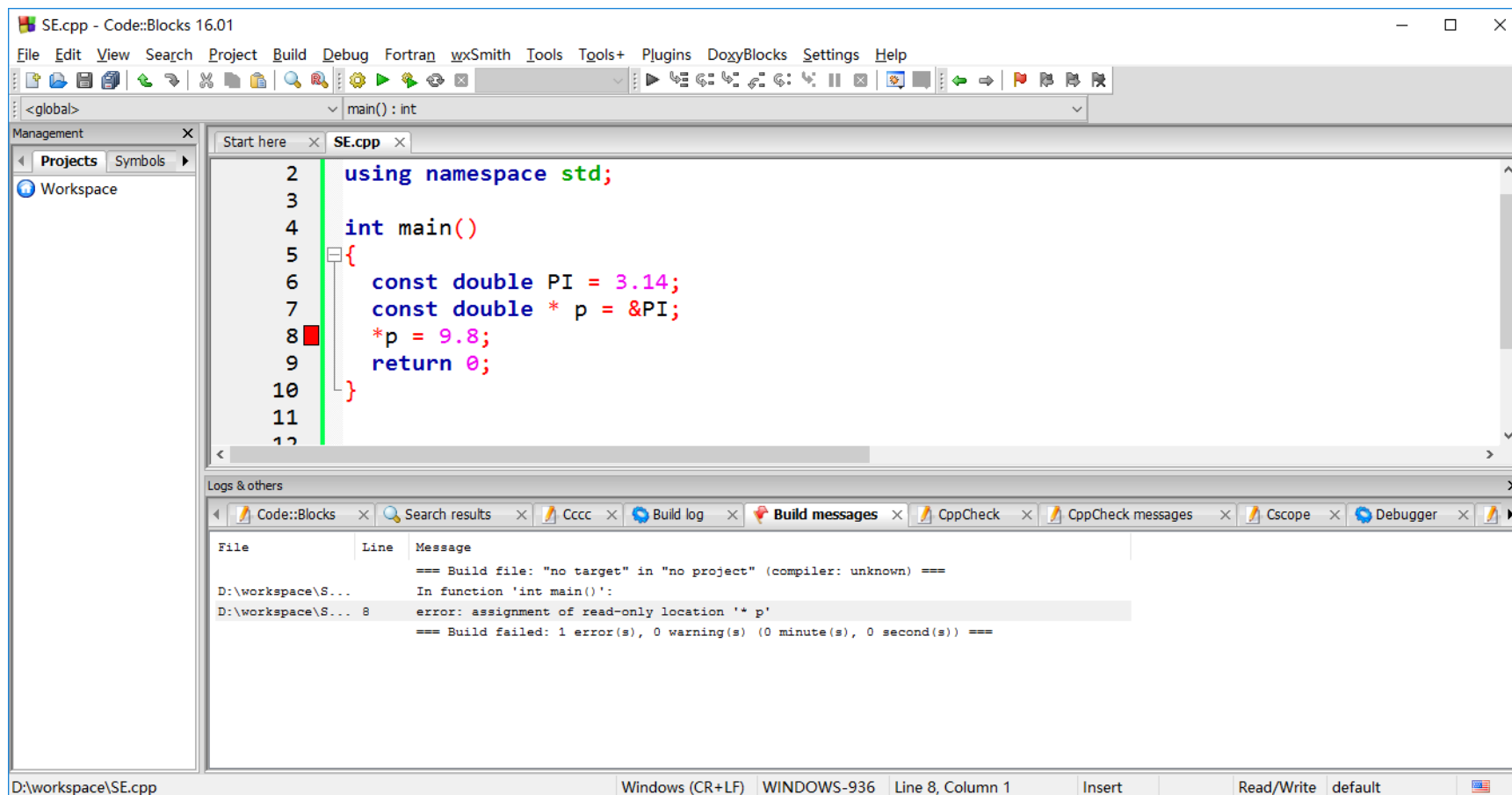
- 声明指向“整型常量”的指针

# 指向常量的指针



- 声明指向“整型常量”的指针（注意 `const` 的位置）

# 指向常量的指针

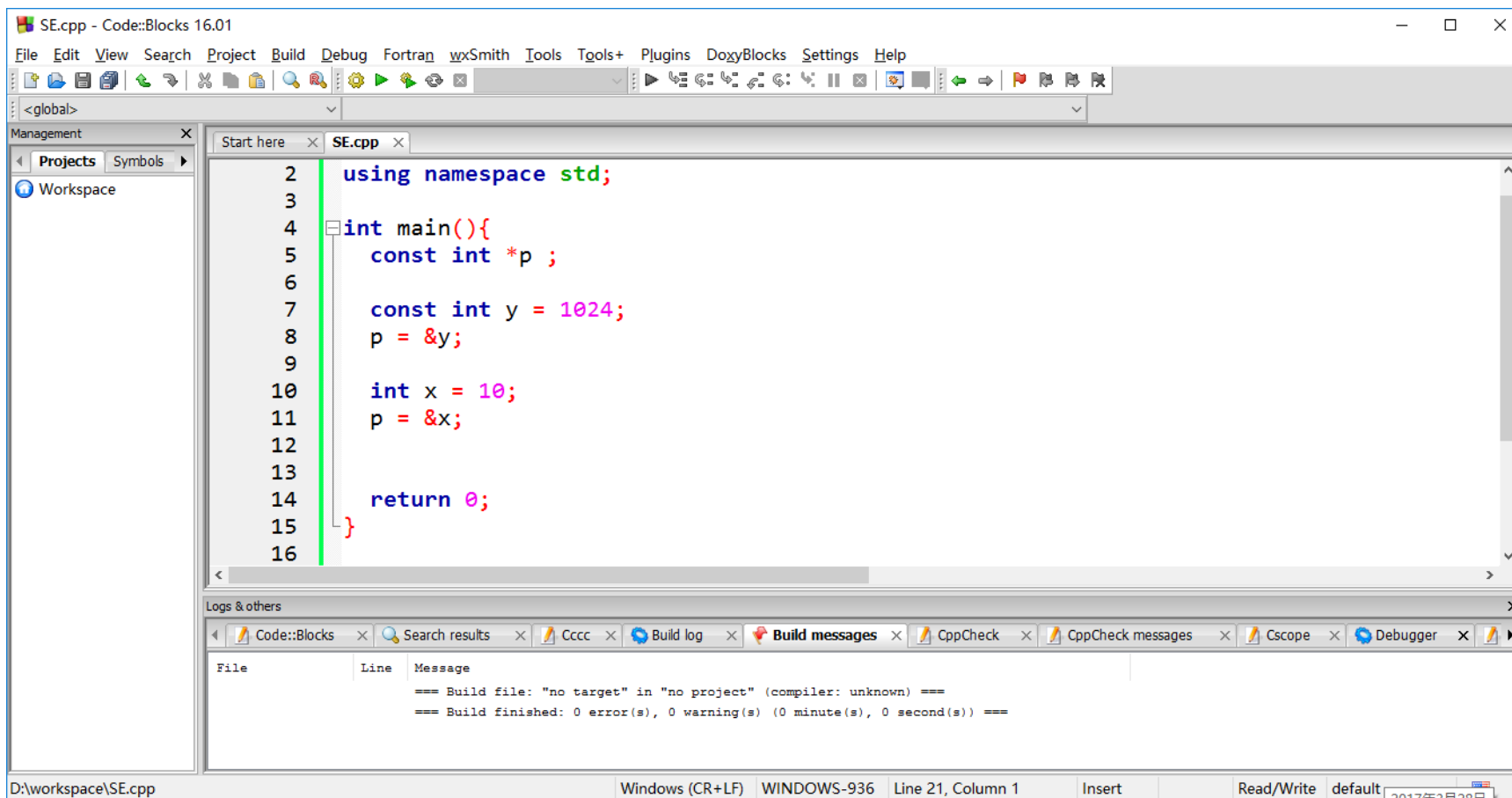


- 不能通过“指向常量的指针”修改常量值

# 指针定义

- `const int *p ;`
  - `const int y = 1024;`
  - `p = &y;`
  - `int x = 10;`
  - `p = &x;`
- 
- **const 常量指针可以指向同类型的普通变量或同类型的常量**
    - **指向常量的指针 (const 常量指针)**

# 指向常量的指针



SE.cpp - Code::Blocks 16.01

File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help

<global>

Management

Start here x SE.cpp x

```
2 using namespace std;
3
4 int main(){
5     const int *p ;
6
7     const int y = 1024;
8     p = &y;
9
10    int x = 10;
11    p = &x;
12
13
14    return 0;
15 }
16
```

Logs & others

Code::Blocks x Search results x Cccc x Build log x Build messages x CppCheck x CppCheck messages x Cscope x Debugger x

File	Line	Message
		=== Build file: "no target" in "no project" (compiler: unknown) ===
		=== Build finished: 0 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===

D:\workspace\SE.cpp Windows (CR+LF) WINDOWS-936 Line 21, Column 1 Insert Read/Write default 2017年3月28日



# 指针定义

- `const int *p ;`
- `const int y = 1024;`
- `p = &y;`
- `int x = 10;`
- `p = &x;`
- `*p = 4869; //语法错误`
- 不能通过指向常量的指针修改所指向的数据（视为常量）
  - `const`常量指针

# 指针定义

The screenshot shows the Code::Blocks 16.01 IDE with a C++ file named SE.cpp. The code defines a main function that declares a constant integer pointer `p`, assigns it the address of a constant integer `y` (1024), and then attempts to assign the value 4869 to `*p`. This causes a compiler error because `p` points to a read-only memory location. The error message is displayed in the 'Build messages' window at the bottom.

```
3
4 int main(){
5     const int *p ;
6
7     const int y = 1024;
8     p = &y;
9
10    int x = 10;
11    p = &x;
12
13    *p = 4869;
14
15    return 0;
16 }
17
```

Build messages:

File	Line	Message
		=== Build file: "no target" in "no project" (compiler: unknown) ===
D:\workspace\S...		In function 'int main()':
D:\workspace\S...	13	error: assignment of read-only location '* p'
		=== Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===

# const指针

- 指向常量的指针（常量指针）
  - `const type * ptr_name ;`
- 在指针声明语句的类型前加const
  - 指向的对象是常量
    - 可以改变指针的指向
    - 不能改变指针指向的变量的值

# 指向常量的指针

- `int x = 48;`
- `const int * p = &x;`
- `int y = 69;`
- `p = &y;`
- `cout<< *p <<endl;`
- `*p = 1024; //error`
- 说明
  - `p` 是变量 (可修改)
  - `*p` 是常量 (不可修改)

# 指向常量的指针

- `void mystrcpy(char *dest, const char *source)`
- `{`
- `while( *dest++ = *source++ );`
- `}`
- `const char *source`
  - 防止作为数据源的数组遭到破坏

# 指针常量

- 指针本身是常量
  - `type * const ptr_name ;`
- 在指针声明语句中的指针名前加const
  - 指针本身是常量
    - 必须初始化
    - 不能改变指针的指向
    - 可以改变指针指向的变量的值

# 指针常量

- `int x = 48;`
- `int * const p = &x;`
- `*p = 1024;`
- `int y = 69;`
- `p = &y;` **//error**
- **说明**
  - **p 是常量 (不可修改)**
  - **\*p 是变量 (可修改)**

# 指针常量

- `const int b = 28 ;`
- `int * const pi = &b ;`

- **错误**

- **不能将 `const int *` 转换成 `int *`**
- **否则，将使得修改常量合法化 (**error**) !**
  - 例如, `*pi = 48;`



# 指向常量的指针常量

- 语法

- `const type * const ptr = 初始值;`

- 必须初始化

- 不能改指针的指向

- 不能改变指针指向的变量的值

# 指针与函数

- `int sum(int abc[], int n);`
- 等价形式
  - `int sum(int *abc, int n);`
- 必须给出数组大小的原因
  - 形参abc是指针变量而不是数组
  - 不能用`sizeof()`求出数组元素的个数

# void指针的作用

- void指针可以接受任何类型普通变量的地址
- 必须强制类型转换
  - void指针才可间接访问目标变量
- 通过强制类型转换
  - void指针可以绕过访问控制从而破坏数据隐藏

# void指针

- `void *p;`
- `int a = 10;`
- `char c = 'a';`
- `p = &a;`
- `p = &c;`
- `cout << *((char *)p) << endl;`

# 函数指针

- 每个函数都有地址
- 函数指针
  - 指向函数地址的指针
  - 通过函数指针可以调用相应的函数
    - 函数返回类型与参数列表一致

# 函数指针

- 定义函数指针

- `Type (* func)(type a, type b, . . . , type n);`

- `(* func)`

- 不能省略小括号

- `()` 的优先级大于 `*`

- `Type * func()` 代表 返回指针的函数

# 函数指针

- 一个函数不能赋给一个不一致的函数指针
  - `int fn1(char x, char y);`
  - `int fn3(int a);`
  - `int (*fp1)(char a, char b);`
  - `fp1 = fn1; // OK`
  - `fp1 = fn3; // error`

# 通过函数指针调用函数

- `int fn1(char x, char y);`
- `int (*fp1)(char a, char b);`
- `fp1 = fn1;`
- `fp1('a', 'b');`
- `(*fp1)('a', 'b');` //兼容C的形式



# 函数指针用作函数参数

- `#include <iostream>`
- `#include <algorithm>`
- `using namespace std;`
- `int sortCriterion(int a,int b){`
- `return a>b;`
- `}`
- `int main(int argc, char** argv){`
- `int a[7]={8,4,9,6,1,0,24};`
- `sort(a, a+7, sortCriterion);`
- `for(int i=0; i<7; ++i){`
- `cout<<a[i]<<endl;`
- `}`
- `return 0;`
- `}`

# 用typedef简化函数指针的形式

- 使用typedef声明函数指针类型的别名
  - `typedef int (*FUN)(int a, int b);`
  - `FUN funp;`
  - FUN 是一个函数指针类型
    - 指向一个拥有两个整数参数并返回一个整型数的函数
    - 指针类型名 (**不是指针变量**)

# 用typedef简化函数指针的形式

```
typedef int (*fun)(int a,int b);
```

```
void test(fun f){  
    cout << f(12,112) << endl;  
}
```

```
int mymax(int x,int y){  
    return x > y ? x : y;  
}
```

```
int main(){  
    test(mymax);  
    return 0;  
}
```

# 任务肆拾伍

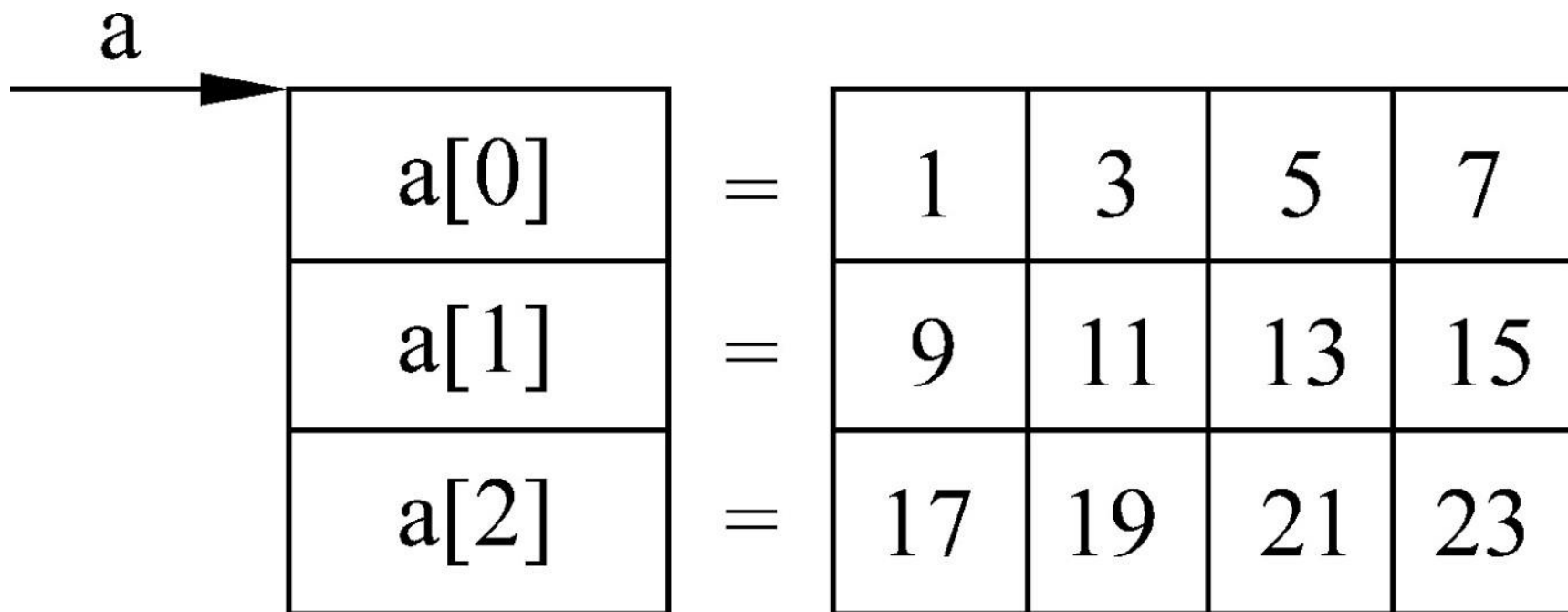
- 计算定积分
  - 根据定积分的定义，计算不同被积函数的定积分。

# 指针与二维数组

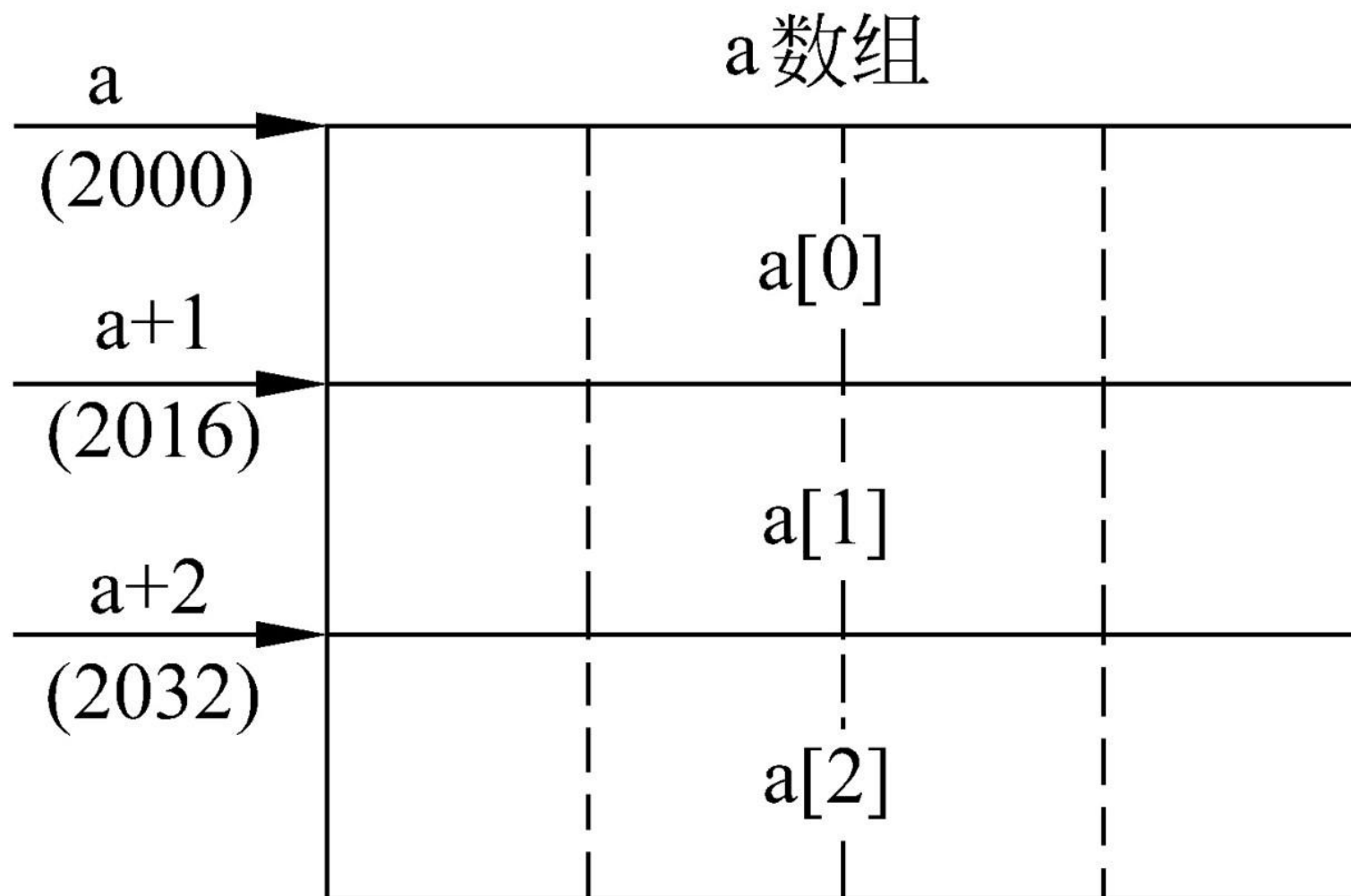
a →

2000 1	2004 3	2008 5	2012 7
2016 9	2020 11	2024 13	2028 15
2032 17	2036 19	2040 21	2044 23

## 二维数组是数组的数组



## 二维数组每行的地址



## 二维数组每列的地址

	$a[0]$	$a[0]+1$	$a[0]+2$	$a[0]+3$
$a$	2000 1	2004 3	2008 5	2012 7
$a+1$	2016 9	2020 11	2024 13	2028 15
$a+2$	2032 17	2036 19	2040 21	2044 23



# 指针与二维数组

- 取得`a[i][j]`的地址

- `&a[i][j]`

- `a[i]+j`

- 访问数组元素

- `a[i][j]`

- `*(a[i]+j)`

- `*(* (a+i)+j)`

# 指向二维数组的指针变量

- 声明语句

- `Type (*指针变量)[列的大小];`

- 该变量只能指向与其具有相同“列的大小”的二维数组

- `int a[2][3]={...};`

- `int (*p)[3] = a;`

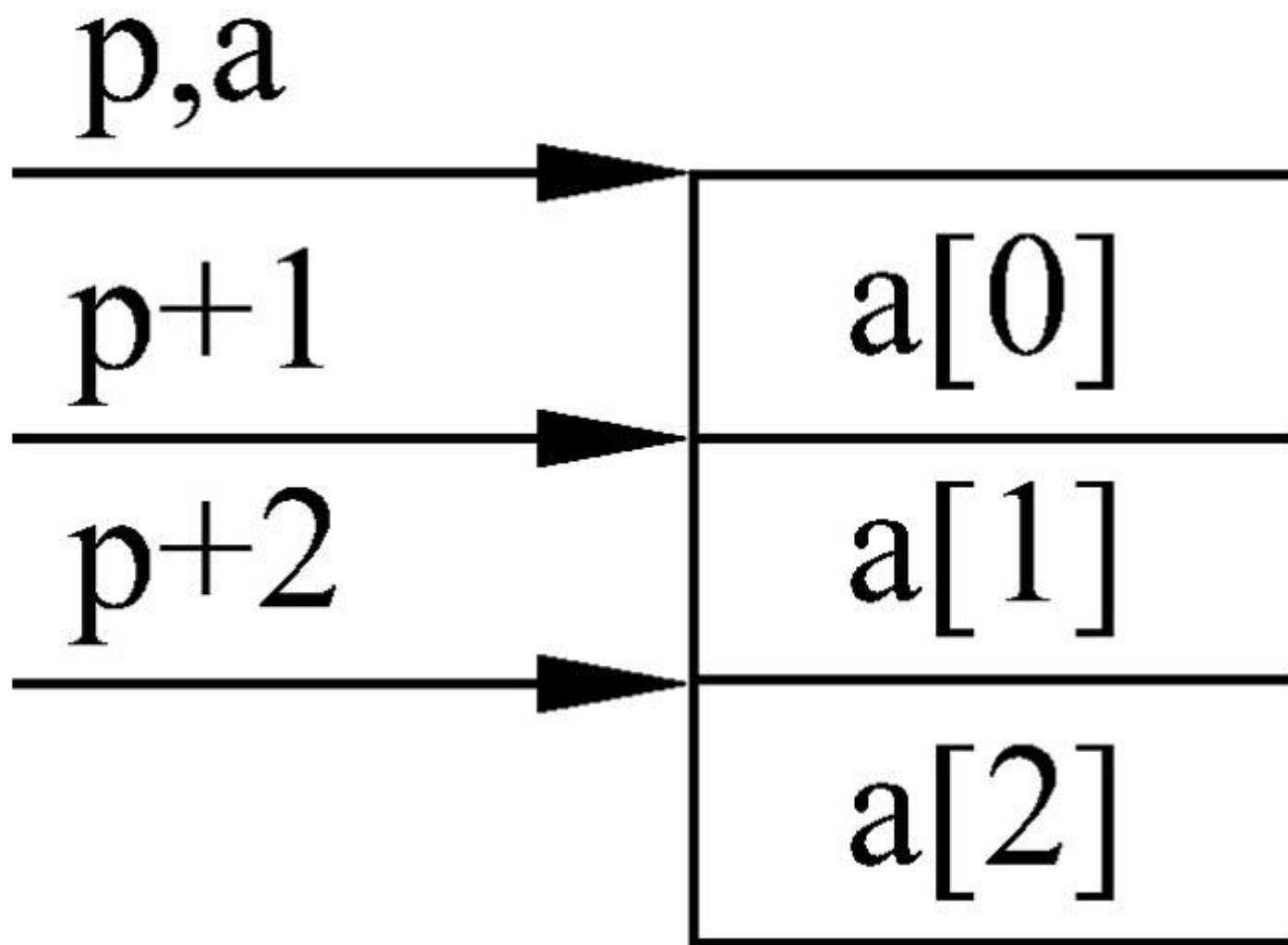
# 指向二维数组的指针变量

- 声明语句

- Type (\*指针变量)[列的大小];

- 该变量只能指向与其具有相同“列的大小”的二维数组
- 小括号不能省略

## 不能使用指向普通变量的指针去指向二维数组



# 指针数组

- 声明语句
  - `Type *指针变量[数组大小];`
- `int *p[3];`
  - 三个整型指针变量
- 数组元素都是指针变量
- 数组大小必须是整型常量

# 指向指针的指针

- 二级指针

- `type ** ppname;`

- 保存同类型指针变量的地址

- 传递指针数组给函数就是传递二级指针给函数

# 命令行参数

- C++程序只不过是操作系统调用的函数
- `int main(int argc, char ** argv)`
  - 固定形式
  - `char ** argv` 等价于 `char * argv[]`
  - `argc`          参数个数
  - `argv`          参数数组

# 命令行参数

- `#include <iostream>`
- `using namespace std;`
- `int main(int argc, char** argv)`
- `{`
- `for(int i=0; i<argc; ++i){`
- `cout<<"arg " <<i<<" : " <<argv[i]<<endl;`
- `}`
- `return 0;`
- `}`



# 命令行参数

- 命令行参数
  - 系统以空格作为区分参数的标志
  - 包含空格的参数应该以双引号包围

**待续.....**