程序设计实习

C++ 面向对象程序设计

张勤健
zqj@pku.edu.cn

北京大学信息科学技术学院

2023 年 2 月 24 日

大纲

1. “引用”的概念和应用
2. “const”关键字的用法
3. 动态内存分配
4. 内联函数、函数重载、函数缺省参数

引用的概念

下面的写法定义了一个引用，并将其初始化为引用某个变量。

```
类型名 & 引用名 = 某变量名;
```

引用的概念

下面的写法定义了一个引用，并将其初始化为引用某个变量。

```
类型名 & 引用名 = 某变量名;
```

```
int n = 4;
int &r = n;  // r 引用了 n, r 的类型是 int &
```

引用的概念

下面的写法定义了一个引用，并将其初始化为引用某个变量。

```
类型名 & 引用名 = 某变量名;
```

```
int n = 4;
int &r = n;  // r 引用了 n, r 的类型是 int &
```

某个变量的引用，等价于这个变量，相当于该变量的一个别名。

引用的概念

```
1  int n = 4;
2  int &r = n;
3  r = 4;
4  cout << r;
5  cout << n;
6  n = 5;
7  cout << r;
```

引用的概念

- 定义引用时一定要将其初始化成引用某个变量。
- 初始化后，它就一直引用该变量，不会再引用别的变量了。
- 引用只能引用变量，不能引用常量和表达式。

引用的概念

```
1  double a = 4, b = 5;
2  double &r1 = a;
3  double &r2 = r1;
4  r2 = 10;
5  cout << a << endl;
6  r1 = b;
7  cout << a << endl;
```

引用的概念

```
1  double a = 4, b = 5;
2  double &r1 = a;
3  double &r2 = r1;
4  r2 = 10;
5  cout << a << endl;
6  r1 = b;
7  cout << a << endl;
```

输出

```
10
5
```

引用应用的简单示例

C 语言中，如何编写交换两个整型变量值的函数?

```
1  void swap(int *a, int *b){
2    int tmp;
3    tmp = *a;
4    *a = *b;
5    *b = tmp;
6  }
7  int n1, n2;
8  swap(&n1, &n2); // n1,n2 的值被交换
```

引用应用的简单示例

C 语言中，如何编写交换两个整型变量值的函数?

```
1  void swap(int *a, int *b){
2    int tmp;
3    tmp = *a;
4    *a = *b;
5    *b = tmp;
6  }
7  int n1, n2;
8  swap(&n1, &n2); // n1,n2 的值被交换
```

有了 C++ 的引用：

```
1  void swap(int &a, int &b) {
2    int tmp;
3    tmp = a;
4    a = b;
5    b = tmp;
6  }
7  int n1, n2;
8  swap(n1, n2); // n1,n2 的值被交换
```

引用作为函数的返回值

```
1  int n = 4;
2  int &SetValue() {
3    return n;
4  }
5  int main() {
6    SetValue() = 40;
7    cout << n;
8    return 0;
9  }
```

引用作为函数的返回值

```
1  int n = 4;
2  int &SetValue() {
3    return n;
4  }
5  int main() {
6    SetValue() = 40;
7    cout << n;
8    return 0;
9  }
```

输出：40

常引用

定义引用时，前面加const关键字，即为“常引用”

```
1  int n;
2  const int &r = n;
```

常引用

定义引用时，前面加const关键字，即为“常引用”

```
1  int n;
2  const int &r = n;
```

r 的类型是 const int &

常引用

定义引用时，前面加const关键字，即为“常引用”

```
1  int n;
2  const int &r = n;
```

r 的类型是 const int &
不能通过常引用去修改其引用的内容:

```
1  int n = 100;
2  const int &r = n;
3  r = 200;   //编译错
4  n = 300;  // 没问题
```

常引用和非常引用的转换

`const T &` 和`T &` 是不同的类型!!!

`T &`类型的引用或`T`类型的变量可以用来初始化`const T &`类型的引用。

`const T`类型的常变量和`const T &` 类型的引用则不能用来初始化`T &`类型的引用，除非进行强制类型转换。

单选题

下面程序片段哪个没错?

A. `int n = 4; int &r = n * 5;`

B. `int n = 6; const int &r = n; r = 7;`

C. `int n = 8; const int &r1 = n; int &r2 = r1;`

D. `int n = 8; int &r1 = n; const int r2 = r1;`

单选题

下面程序片段哪个没错?

A. `int n = 4; int &r = n * 5;`

B. `int n = 6; const int &r = n; r = 7;`

C. `int n = 8; const int &r1 = n; int &r2 = r1;`

D. `int n = 8; int &r1 = n; const int r2 = r1;`

答案：D

单选题

下面程序片段输出结果是什么?

```
1  int a = 1, b = 2;
2  int &r = a;
3  r = b;
4  r = 7;
5  cout << a << endl;
```

A. 1
B. 2
C. 7
D. 都不是

单选题

下面程序片段输出结果是什么?

```
1  int a = 1, b = 2;
2  int &r = a;
3  r = b;
4  r = 7;
5  cout << a << endl;
```

- A 1
- B 2
- C 7
- D 都不是

答案：C

定义常量

```
1  const int MAX_VAL = 23;
2  const string SCHOOL_NAME = "Peking University";
```

定义常量指针

不可通过常量指针修改其指向的内容

```
1  int n, m;
2  const int *p = &n;
3  *p = 5; //编译出错
4  n = 4;  // ok
5  p = &m; // ok, 常量指针的指向可以变化
```

定义常量指针

不能把常量指针赋值给非常量指针，反过来可以

```
1  const int *p1;
2  int *p2;
3  p1 = p2;          // ok
4  p2 = p1;          // error
5  p2 = (int *)p1; // ok, 强制类型转换
```

定义常量指针

函数参数为常量指针时，可避免函数内部不小心改变参数指针所指地方的内容

```
1  void myPrintf(const char *p) {
2    strcpy(p, "this"); //编译出错
3    printf("%s", p);   // ok
4  }
```

定义常引用

不能通过常引用修改其引用的变量

```
1  int n;
2  const int &r = n;
3  r = 5; // error
4  n = 4; // ok
```

单选题

下面说法哪种是对的?

- A 常引用所引用的变量，其值不能被修改
- B 不能通过常量指针，去修改其指向的变量
- C 常量指针一旦指向某个变量，就不能再指向其他变量
- D 以上都不对

单选题

下面说法哪种是对的?

- A 常引用所引用的变量，其值不能被修改
- B 不能通过常量指针，去修改其指向的变量
- C 常量指针一旦指向某个变量，就不能再指向其他变量
- D 以上都不对

答案：B

用 new 运算符实现动态内存分配

分配一个变量:

```
1  P = new T;
```

T是任意类型名，P是类型为T * 的指针。
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P。比如：

```
1  int *pn;
2  pn = new int;
3  *pn = 5;
```

用 new 运算符实现动态内存分配

分配一个数组:

```
1  P = new T[N];
```

T：任意类型名
P：类型为T * 的指针
N：要分配的数组元素的个数，可以是整型表达式

动态分配出一片大小为 sizeof(T) * N字节的内存空间，并且将该内存空间的起始地址赋值给P。

```
1  int *pn;
2  int i = 5;
3  pn = new int[i * 20];
4  pn[0] = 20;
5  pn[100] = 30; // 编译没问题。运行时导致数组越界
```

用 delete 运算符释放动态分配的内存

用“new”动态分配的内存空间，一定要用“delete”运算符进行释放

```
1  delete 指针; //该指针必须指向 new 出来的空间
```

```
1  int *p = new int;
2  *p = 5;
3  delete p;
4  delete p; // 导致异常，一片空间不能被 delete 多次
```

用 delete 运算符释放动态分配的数组

用“delete”释放动态分配的数组，要加“[]”

```
1  delete[] 指针;//该指针必须指向 new 出来的数组
```

```
1  int *p = new int[20];
2  p[0] = 1;
3  delete[] p;
```

单选题

表达式“`new int`”的返回值类型是：

A. `int`
B. `int *`
C. `int &`
D. `void`

单选题

表达式“`new int`”的返回值类型是：

A. `int`
B. `int *`
C. `int &`
D. `void`

答案：B

单选题

下面小段程序，哪个是正确的：

A. `char *p = new int; p = 'a'; delete p;`
B. `int *p = new int[25]; p[10] = 100; delete p;`
C. `char *p = new char[10]; p[0] = 'K'; delete[] p;`
D. 都不对

单选题

下面小段程序，哪个是正确的：

A. `char *p = new int; p = 'a'; delete p;`
B. `int *p = new int[25]; p[10] = 100; delete p;`
C. `char *p = new char[10]; p[0] = 'K'; delete[] p;`
D. 都不对

答案：C

内联函数

函数调用是有时间开销的。如果函数本身只有几条语句，执行非常快，而且函数被反复执行很多次，相比之下调用函数所产生的这个开销就会显得比较大。

为了减少函数调用的开销，引入了内联函数机制。编译器处理对内联函数的调用语句时，是将整个函数的代码插入到调用语句处，而不会产生调用函数的语句。

内联函数

函数调用是有时间开销的。如果函数本身只有几条语句，执行非常快，而且函数被反复执行很多次，相比之下调用函数所产生的这个开销就会显得比较大。

为了减少函数调用的开销，引入了内联函数机制。编译器处理对内联函数的调用语句时，是将整个函数的代码插入到调用语句处，而不会产生调用函数的语句。

```
1  inline int _max(int a, int b) {
2    if (a > b) return a;
3    return b;
4  }
```

函数重载

一个或多个函数，名字相同，然而参数个数或参数类型不相同，这叫做函数的重载。

函数重载

一个或多个函数，名字相同，然而参数个数或参数类型不相同，这叫做函数的重载。以下三个函数是重载关系：

```
1  int _max(double f1, double f2) {}
2  int _max(int n1, int n2) {}
3  int _max(int n1, int n2, int n3) {}
```

函数重载

一个或多个函数，名字相同，然而参数个数或参数类型不相同，这叫做函数的重载。以下三个函数是重载关系：

```
1  int _max(double f1, double f2) {}
2  int _max(int n1, int n2) {}
3  int _max(int n1, int n2, int n3) {}
```

函数重载使得函数命名变得简单。
编译器根据调用语句的中的实参的个数和类型判断应该调用哪个函数。

函数重载

一个或多个函数，名字相同，然而参数个数或参数类型不相同，这叫做函数的重载。以下三个函数是重载关系：

```
1  int _max(double f1, double f2) {}
2  int _max(int n1, int n2) {}
3  int _max(int n1, int n2, int n3) {}
```

函数重载使得函数命名变得简单。
编译器根据调用语句的中的实参的个数和类型判断应该调用哪个函数。

```
1  _max(3.4, 2.5);  // 调用 (1)
2  _max(2, 4);  // 调用 (2)
3  _max(1, 2, 3);  // 调用 (3)
4  _max(3, 2.4);  // error, 二义性
```

函数的缺省参数

C++ 中，定义函数的时候可以让最右边的连续若干个参数有缺省值，那么调用函数的时候，若相应位置不写参数，参数就是缺省值。

函数的缺省参数

C++ 中，定义函数的时候可以让最右边的连续若干个参数有缺省值，那么调用函数的时候，若相应位置不写参数，参数就是缺省值。

```
1  void func(int x1, int x2 = 2, int x3 = 3) {}
```

```
1  func(10 ) ; //等效于 func(10,2,3)
2  func(10, 8) ; //等效于 func(10,8,3)
3  func(10, , 8) ; //不行，只能最右边的连续若干个参数缺省
```

函数的缺省参数

函数参数可缺省的目的在于提高程序的可扩充性。

即如果某个写好的函数要添加新的参数，而原先那些调用该函数的语句，未必需要使用新增的参数，那么为了避免对原先那些函数调用语句的修改，就可以使用缺省参数。

单选题

下面说法正确的是：

A. 多个重载函数的参数个数必须不同。
B. 两个函数，参数表相同，返回值类型不同，它们是重载关系。
C. 调用一个第二个和第三个参数都有有缺省值的函数时，可以不写第二个实参而写第三个实参。
D. 使用内联函数的目的是提高程序的运行速度。

单选题

下面说法正确的是：

A. 多个重载函数的参数个数必须不同。
B. 两个函数，参数表相同，返回值类型不同，它们是重载关系。
C. 调用一个第二个和第三个参数都有有缺省值的函数时，可以不写第二个实参而写第三个实参。
D. 使用内联函数的目的是提高程序的运行速度。

答案：D