

知识点1【c++语法检查增强】

知识点2【c++对结构体的增强】

1、c中定义结构体变量需要加上struct关键字，c++不需要

2、c中的结构体只能定义成员变量，不能定义成员函数。c++即可以定义成员变量，也可以定义成员函数

知识点3【c++新增bool类型】

知识点4【三目运算符功能增强】 $a > b ? a : b$

1、c语言三目运算表达式返回值为数据值，为右值，不能赋值

2、c++语言三目运算表达式返回值为变量本身(引用)，为左值，可以赋值

知识点5【c++中const】

a、c语言的const修饰全局变量 默认是（外部链接的）

总结：在c语言中

2、c++中的const 深入理解

知识点6【尽量const替换#define】

1、宏没有类型 const有

2、宏的作用域是整个文件 const的作用域 以定义情况决定

3、宏不能作为命名空间的成员 const可以

知识点7【引用】（重要）给已有变量取个别名

语法：

注意：

1、引用必须初始化

2、引用一旦初始化 就不能再次修改别名

知识点8【引用 给数组 取个别名】

1、方式一：梁哥法

2、法法二：配合typedef

知识点9【引用作为函数的参数】

知识点10【引用作为函数的返回值】

当函数返回值作为左值 那么函数的返回值类型必须是引用。

知识点11【引用的本质】(了解)

知识点12【指针的引用】(了解)

知识点13【常引用】

1、引导出常引用(重要)

2、常量的引用(了解)

知识点1【c++语法检查增强】

3.3 全局变量检测增强

c 语言代码：

```
int a = 10; //赋值，当做定义
int a; //没有赋值，当做声明
```

```
int main(){
    printf("a:%d\n",a);
    return EXIT_SUCCESS;
}
```

此代码在 c++ 下编译失败,在 c 下编译通过.

3.4 C++中所有的变量和函数都必须有类型

c 语言代码:

```
//i 没有写类型，可以是任意类型
int fun1(i){
    printf("%d\n", i);
    return 0;
}
//i 没有写类型，可以是任意类型
int fun2(i){
    printf("%s\n", i);
    return 0;
}
//没有写参数，代表可以传任何类型的实参
int fun3(){
    printf("fun333333333333333333\n");
    return 0;
}
```

以上c代码c编译器编译可通过，c++编译器无法编译通过

3.5 更严格的类型转换

在 C++，不同类型的变量一般是不能直接赋值的，需要相应的强转。c 语言代码:

```
typedef enum COLOR{ GREEN, RED, YELLOW } color;
int main(){

    color mycolor = GREEN;
    mycolor = 10;
    printf("mycolor:%d\n", mycolor);
    char* p = malloc(10);
    return EXIT_SUCCESS;
}
```

以上 c 代码 c 编译器编译可通过，c++编译器无法编译通过。

知识点2 【c++对结构体的增强】

- 1、c中定义结构体变量需要加上struct关键字，c++不需要
- 2、c中的结构体只能定义成员变量，不能定义成员函数。c++即可以定义成员变量，也可以定义成员函数

C语言:

```

1 struct stu
2 {
3     int num;
4     char name[32];
5     /* c语言 不允许在结构体中 定义成员函数
6     void func(void)
7     {
8         printf("我是结构体中的func");
9     }
10    */
11 };
12 void test01()
13 {
14     //C语言必须加struct
15     struct stu lucy = {100, "lucy"};
16 }

```

C++:

```

1 struct stu
2 {
3     int num;
4     char name[32];
5
6     //c++语言 允许在结构体中 定义成员函数
7     void func(void)
8     {
9         printf("我是结构体中的func");
10    }
11
12 };
13 void test01()
14 {
15     //C++语言不用加struct
16     stu lucy = {100, "lucy"};
17
18     //调用结构体中的成员方法（成员函数）
19     lucy.func();
20 }

```

知识点3 【c++新增bool类型】

标准 c++ 的 `bool` 类型有两种内建的常量 `true` (转换为整数 1) 和 `false` (转换为整数 0) 表示状态。这三个名字都是关键字。 `bool` 类型只有两个值，`true` (1 值)，`false` (0 值) `bool` 类型占 1 个字节大小 给 `bool` 类型赋值时，非 0 值会自动转换为 `true` (1), 0 值会自动转换 `false` (0)

```
1 void test02()
2 {
3     bool mybool;
4     cout<<"sizeof(bool) = "<<sizeof(bool)<<endl;//1字节
5     mybool = false;
6     cout<<"false = "<<false<<endl;//0
7     cout<<"true = "<<true<<endl;//1
8 }
```

知识点4 【三目运算符功能增强】 `a>b?a:b`

1、c语言三目运算表达式返回值为数据值，为右值，不能赋值

```
1 void test02()
2 {
3     int a = 10;
4     int b = 20;
5     printf("C语言:%d\n", a>b?a:b);//20
6
7     //a>b?a:b整体结果 右值（不能被赋值）
8     a>b?a:b = 100;//err不能被赋值
9 }
```

2、c++语言三目运算表达式返回值为变量本身(引用)，为左值，可以赋值

```
1 void test02()
2 {
3     int a = 10;
4     int b = 20;
5     cout<<"c++中:"<<(a>b?a:b)<<endl;
6
7     //a>b?a:b整体结果是变量本身(引用) 左值（能被赋值）
8     a>b?a:b = 100;//b =100
9 }
```

[左值和右值概念] 在 c++中可以放在赋值操作符左边的是左值，可以放到赋值操作符右面的是右值。有些变量即可以当左值，也可以当右值。左值为 Lvalue，L 代表 Location，表示内存可以寻址，可以赋值。右值为 Rvalue，R 代表 Read，就是可以知道它的值。比如：int temp = 10; temp 在内存中有地址，10 没有，但是可以 Read 到它的值。

能被赋值的就是左值 不能被赋值的就是右值

知识点5 【c++中const】

1、c语言中的const特点

```
1  const int a = 10; //不要把a看成常量
2  //a的本质 是变量 只是 只读变量
```

a、c语言的const修饰全局变量 默认是（外部链接的）

fun.c

```
1  //c语言的const修饰全局变量 默认是（外部链接的）
2  //外部链接:其他源文件 可以使用
3  const int num = 100; //只读的全局变量 内存放在文字常量区（内存空间只读）
4
```

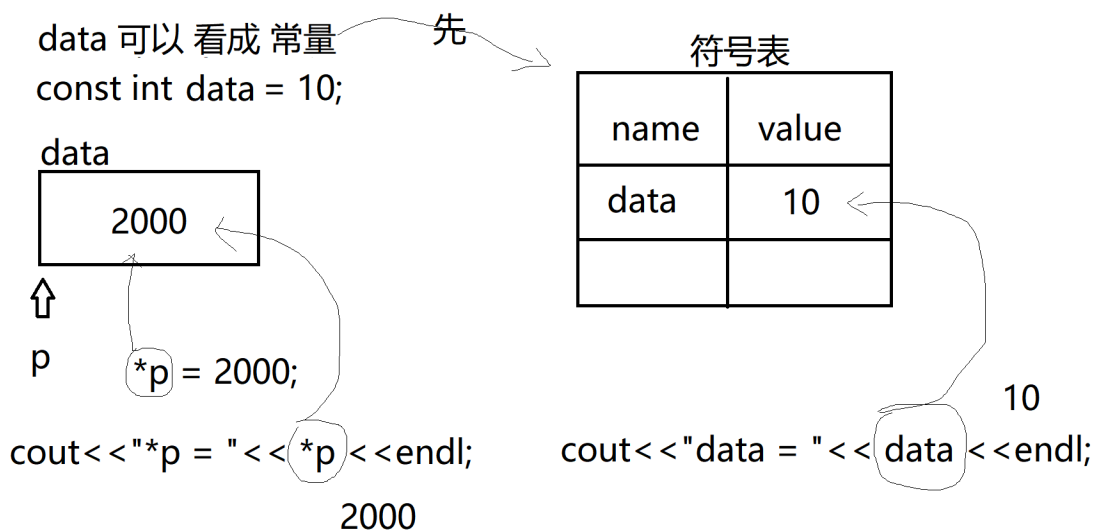
main.c

```
1  //对fun.c中的num进行声明(不要赋值)
2  extern const int num;
3
4  void test03()
5  {
6      printf("num = %d\n", num);
7      //num = 200; //err num只读
8
9      //C语言中const 修饰变量名 说明变量名为只读（用户不能通过变量名data进行赋值）
10     const int data = 100; //局部只读变量 内存在栈区（内存可读可写）
11     //data = 200; //err
12
13     printf("data = %d\n", data);
14     //但是：如果知道data的地址 可以通过地址间接的修改data所对应空间的内容
15     int *p = (int *)&data;
16     *p = 2000;
17     printf("data = %d\n", data); //ok 200
18 }
```

总结：在c语言中

- 1、const修饰全局变量num 变量名只读 内存空间在文字常量区（只读）、不能通过num的地址 修改空间内容
- 2、const修饰局部变量data 变量名只读 内存空间栈区（可读可写），可以通过data地址 间接的修改空间内容

2、c++中的const 深入理解



fun.cpp

```
1 //const修饰的全局变量 默认是内部链接（只在当前源文件有效 不能直接用于其他源文件）
2 //const int num = 100;
3 //如果必须用在其他源文件 使用只读的全局变量 必须加extern将num转换成外部链接
4 extern const int num = 100;
5
```

main.cpp

```
1 //声明
2 extern const int num;
3 struct Person
4 {
5     int num;
6     char name[32];
7 };
```

```

8 void test04()
9 {
10  cout<<"全局num = "<<num<<endl;//err 不识别num
11
12  //1、c++中 对于基础类型 系统不会给data开辟空间 data放到符号表中
13  const int data = 10;
14  //data = 100;//err 只读
15  cout<<"data = "<<data<<endl;
16  //2、c++中当 对data 取地址的时候 系统就会给data开辟空间
17  int *p = (int *)&data;
18  *p = 2000;
19  cout<<"*p = "<<*p<<endl;//空间内容修改成功 2000
20
21  cout<<"data = "<<data<<endl;//data 还是10为啥?
22
23  //2、当以变量的形式 初始化 const修饰的变量 系统会为其开辟空间
24  int b = 200;
25  const int a = b;//系统直接为a开辟空间 而不会把a放入符号表中
26  p = (int *)&a;
27  *p = 3000;
28  cout<<"*p = "<<*p <<endl;//3000
29  cout<<"a = "<<a <<endl;//3000
30
31  //3、const 自定义数据类型(结构体、对象) 系统会分配空间
32  const Person per = {100,"lucy"};
33  //per.num = 1000;//err
34  cout<<"num = "<<per.num<<", name = "<<per.name<<endl;//100 lucy
35  Person *p1 = (Person *)&per;
36  p1->num = 2000;
37  cout<<"num = "<<per.num<<", name = "<<per.name<<endl;//2000 lucy
38 }

```

运行结果：


```
全局num = 100  
data = 10  
*p = 2000  
data = 10  
*p = 3000  
a = 3000  
num = 100, name = lucy  
num = 2000, name = lucy
```

总结：c++总结

- 1、const int data = 10; //data先放入符号表
- 2、如果对data取地址 系统才会给data开辟空间
- 3、const int a = b; //b是变量名 系统直接给a开辟空间 而不放入符号表
- 4、const 修饰自定义数据 系统为自定义数据开辟空间

知识点6 【尽量const替换#define】

const 和#define 区别总结:

1. const 有类型，可进行编译器类型安全检查。#define 无类型，不可进行类型检查。
2. const 有作用域，而#define 不重视作用域，默认定义处到文件结尾.如果定义在指定作用域下有效的常量，那么#define 就不能用。

1、宏没有类型 const有

```
1 #define MAX 1024
2 const short my_max =1024;
3 void func(short i)
4 {
5     cout<<"short函数"<<endl;
6 }
7 void func(int i)
8 {
9     cout<<"int函数"<<endl;
10 }
11 void test05()
12 {
13     func(MAX); //int 函数
14
15     func(my_max); //short函数
16 }
```

2、宏的作用域是整个文件 const的作用域 以定义情况决定

```
1 void my_func(void)
2 {
3     //作用范围 是当前复合语句
4     const int my_num = 10;
5
6     //作用范围 当前位置 到文件结束
7     #define MY_NUM 10
8 }
9 void test06()
10 {
11     //cout<<"my_num = "<<my_num<<endl; //err 不识别
12     cout<<"MY_NUM = "<<MY_NUM<<endl; //ok 能识别
13 }
```

3、宏不能作为命名空间的成员 const可以

```
1 namespace A {
2     // const可以作为成员
3     const int my_a=100;
4
5     //MY_A 属于文件 不属于A
6     #define MY_A 200
```

```

7 }
8 void test07()
9 {
10  cout<<"my_a = "<<A::my_a<<endl;
11  //cout<<"MY_A = "<<A::MY_A<<endl;//err
12  cout<<"MY_A = "<<MY_A<<endl;
13 }

```

知识点7【引用】（重要）给已有变量取个别名

语法：

- 1、&和别名 结合 表示引用
- 2、给某个变量取别名 就定义某个变量
- 3、从上往下替换

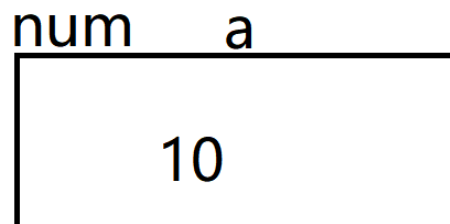
```

1 int num = 10;
2 int &a = num; //此处 &不是取地址 而是标明 a是引用变量（a 是 num的别名）

```

int num = 10;

int &a = num;



a完全等价于num

注意：

- 1、引用必须初始化
- 2、引用一旦初始化 就不能再次修改别名

```

1 int num = 10;
2 int &a = num;
3
4 int data = 20;
5 a = data; //不是data别名为a 而是将data值赋值a(num)

```

案例：

```

1 int num = 10;

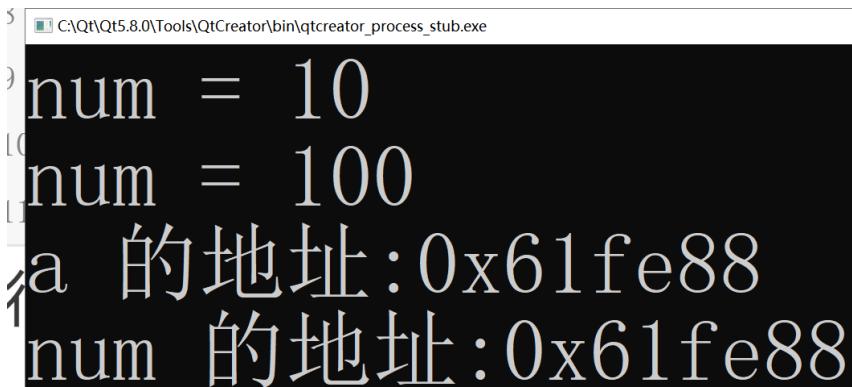
```

```

2  int &a = num;//a就是num的别名 a==num
3
4  cout<<"num = "<<num<<endl;//10
5  //对a赋值 == 对num赋值
6  a=100;
7  cout<<"num = "<<num<<endl;//100
8
9  //a是num的别名 所以num和a具有相同的地址空间
10 cout<<"a 的地址:"<<&a<<endl;
11 cout<<"num 的地址:"<<&num<<endl;

```

运行结果:



```

5  C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_process_stub.exe
num = 10
num = 100
a 的地址:0x61fe88
num 的地址:0x61fe88

```

知识点8 【引用 给数组 取个别名】

1、方式一：梁哥法

```

1  void test02()
2  {
3      int arr[5] = {10,20,30,40,50};
4      //需求: 给arr起个别名
5      int (&my_arr)[5] = arr;//my_arr就是数组arr的别名
6
7      int i=0;
8      for(i=0;i<5;i++)
9      {
10         cout<<my_arr[i]<<" ";
11     }
12     cout<<endl;
13 }

```

2、法法二：配合typedef

```

1 void test03()
2 {
3     int arr[5] = {10,20,30,40,50};
4     //1、用typedef 给数组类型 取个别名
5     //TYPE_ARR就是一个数组类型（有5个元素 每个元素位int）
6     typedef int TYPE_ARR[5];
7
8     //myArr就是数组arr的别名
9     TYPE_ARR &myArr=arr;
10
11     int i=0;
12     for(i=0;i<5;i++)
13     {
14         cout<<myArr[i]<<" ";
15     }
16     cout<<endl;
17 }

```

知识点9 【引用作为函数的参数】

```

1 void my_swap1(int a,int b)
2 {
3     int tmp = a;
4     a = b;
5     b=tmp;
6 }
7 void my_swap2(int *a,int *b)//a=&data1,b =data2;
8 {
9     int tmp = *a;
10    *a = *b;
11    *b = tmp;
12 }
13
14 void my_swap3(int &a, int &b)//a=data1,b=data2
15 {
16     int tmp = a;
17     a = b;
18     b= tmp;
19 }
20 void test04()

```

```

21 {
22     int data1 = 10,data2=20;
23     cout<<"data1 = "<<data1<<" , data2 = "<<data2<<endl;
24     //my_swap1(data1,data2);//交换失败
25     //my_swap2(&data1,&data2);//交换成功
26     my_swap3(data1,data2);//交换成功(推荐)
27     cout<<"data1 = "<<data1<<" , data2 = "<<data2<<endl;
28 }

```

知识点10 【引用作为函数的返回值】

给函数的返回值 取个别名

```

1 //引用作为函数的返回值类型
2 int& my_data(void)
3 {
4     int num = 100;
5     return num;//err 函数返回啥变量 引用就是该变量的别名
6     //函数的返回值是引用时 不要返回局部变量
7 }
8
9 int& my_data1(void)
10 {
11     static int num = 200;
12     return num;//ok
13 }
14 void test05()
15 {
16     //ret是别名 ret是num的别名
17     int &ret = my_data();
18     //cout<<"ret = "<<ret<<endl;//非法访问内存
19
20     int &ret1 = my_data1();//ret1是num的别名
21     cout<<"ret = "<<ret1<<endl;
22 }

```

当函数返回值作为左值 那么函数的返回值类型必须是引用。

```

int& my_data2(void)
{
    static int num = 10;
    cout<<"num = "<<num<<endl;

    return num;
}

void test06()
{
    //函数的返回值 作为左值
    my_data2() = 2000;

    my_data2();
}

```

C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_process_stub.exe

```

num = 10
num = 2000

```

知识点11 【引用的本质】（了解）

引用的本质在c++内部实现是一个**指针常量**. `Type& ref = val; // Type* const ref = &val;`
 c++编译器在编译过程中使用常指针作为引用的内部实现，因此引用所占用的空间大小与指针相同，只是这个过程是编译器内部实现，用户不可见

```

1 int data = 10;
2 int &a = data; //a就是data的别名
3 //编译器内存转换: int * const a = &data;
4
5 a=100; //等价于data=100
6 // *a = 100; // *a == data

```

知识点12 【指针的引用】（了解）

```

1 #include<stdlib.h>
2 #include<string.h>
3 void my_str1(char **p_str) //p_str = &str
4 {
5     // *p_str == *&str == str
6     *p_str = (char *)calloc(1,32);
7     strcpy(*p_str, "hello world");
8
9     return;
10 }
11 void my_str2(char* &my_str) //char* &my_str = str; my_str等价str

```

```

12 {
13     my_str = (char *)calloc(1,32);
14     strcpy(my_str, "hello world");
15     return;
16 }
17 void test07()
18 {
19     char *str = NULL;
20     //需求: 封装一个函数 从堆区 给str申请一个空间 并赋值为"hello world"
21     //my_str1(&str);
22     my_str2(str);
23     cout<<"str = "<<str<<endl;
24     free(str);
25 }

```

知识点13 【常引用】

1、引导出常引用 (重要)

```

1 typedef struct
2 {
3     int num;
4     char name[32];
5 }STU;
6 void myPrintSTU1(STU tmp)//普通结构体变量作为形参 开销太大
7 {
8     cout<<sizeof(tmp)<<endl;
9     cout<<"学号:"<<tmp.num<<", 姓名:"<<tmp.name<<endl;
10 }
11 void myPrintSTU2(const STU &tmp)//STU &tmp=lucy;tmp是lucy的别名 tmp没有开辟独立空间
12 {
13     //tmp.num = 2000;//err 因为tmp为常引用
14     cout<<"学号:"<<tmp.num<<", 姓名:"<<tmp.name<<endl;
15 }
16 void test08()
17 {
18     STU lucy = {100,"lucy"};
19
20     //需求:定义一个函数 遍历lucy成员(读操作)
21     myPrintSTU2(lucy);

```



```
22 }
```

2、常量的引用（了解）

```
1 void test09()  
2 {  
3     //给常量10取个别名 叫num  
4     //int &针对的是int , 10是const int类型  
5     //const int 针对的是const int, 10是const int类型  
6     const int &num = 10;  
7  
8     cout<<"num = "<<num<<endl;//10  
9 }
```