一、宏

> 宏替换是C/C++系列语言的技术特色，C/C++语言提供了强大的宏替换功能，源代码在进入编译器之前，要先经过一个称为“预处理器”的模块，这个模块将宏根据编译参数和实际编码进行展开，展开后的代码才正式进入编译器，进行词法分析、语法分析等等

1. 宏变量

> 宏变量和const 修饰的在定义语义上没有什么不同，都是可以用来定义常量，但在与const的定义进行对比时，没有任何优势可言，所以建议使用const来定义常量。

```
#define MAX 30

int main(){
        int scores[MAX]; //表示一个班30个人的成绩数组。

        return 0 ;
}
```

2. 条件宏

> 条件宏最常出现在头文件中，用于防止头文件被反复包含。

- 头文件的条件宏

```
#ifndef STUDENT_H
#define STUDENT_H
……
……
#endif
```

- 用于判断编译的条件宏

> 通过DEBUG宏，我们可以在代码调试的过程中输出辅助调试的信息。当DEBUG宏被删除时，这些输出的语句就不会被编译。更重要的是，这个宏可以通过编译参数来定义。因此

> 通过改变编译参数，就可以方便的添加和取消这个宏的定义，从而改变代码条件编译的结果.

```
#define DEBUG 0
#define REALEASE

#include<iostream>

using namespace std;

int main() {
    #ifdef DEBUG
        cout <<"debug模式下打印" << endl;
    #elif REALEASE
        cout <<"release模式下打印" << endl;
    #else
        cout <<"普通模式下打印" << endl;
    #endif

    //下面可言继续编写原有的逻辑
    cout << "继续执行逻辑代码~~~"<<endl;

    return 0 ;
}
```

二、枚举

> 在C++里面定义常量，可以使用 `#define和const` 创建常量， 除此之外，还提供了枚举这种方式，它除了能定义常量之外，还表示了一种新的类型，但是必须按照一定的规则来定义。在枚举中定义的成员可以称之为 **枚举量** ，每个枚举量都能对应一个数字，默认是他们的出现顺序，从0开始。
>
> C++的枚举定义有两种方式， **限定作用域** 和 **不限定作用域** ，根据方式的不同，定义的结构也不同。

1. 两种定义方式

1. 限定作用域

> 使用 enum class 或者 enum struct 关键字定义枚举，枚举值位于 enum 的局部作用域内，枚举值不会隐式的转化成其他类型

```
enum class Week{MON,TUS,WEN,THU,FRI,STU,SUN};


int main(){

        int val = (int) Week::TUS ; //打印会是1

        return 0 ;
}
```

2. 不限定作用域

> 使用 enum 关键字定义，省略 class | struct ， 枚举值与枚举类型位于同一个作用域，枚举值会隐式的转化成整数，默认是从0开始，依次类推。 不允许有重复枚举值，因为他们属于同一个作用域。

```
enum traffic_light{red, yellow , green};

//匿名的未限定作用域
enum{red, yellow , green}; //重复定义会报错，因为red\yellow\green 已经定义过了。

//手动给枚举量 设置对应的数值
enum{red = 10, yellow =20 , green =30};

int main(){

    //使用 域操作符来获取对应的枚举量
        int a=  traffic_light::red;
        int b = ::red;

        return 0 ;
}
```

2. 枚举的使用

> 枚举的目的：增加程序的可读性。枚举类型最常见也最有意义的用处之一就是用来描述状态量。
>
> 固定一个名字， 固定的几个名字。不允许程序员在外来的编码中修改名称。
>
> 性别： 男 、 女 nan nv man women male female 1 0

```cpp
#include <iostream>
#include <vector>
#include <string>

using namespace std;

enum Gender{MALE , FEMALE};

class teacher{

public:
    string name;
    Gender gender;

    teacher(string name , Gender gender):name(name),gender(gender){}
};

int main(){

    teacher t1("张三" ,  Gender::MALE);
    teacher t2("李丽丽" ,  Gender::FEMALE);
    teacher t3("李四" ,  Gender::MALE);

    vector<teacher> v;
    v.push_back(t1);
    v.push_back(t2);
    v.push_back(t3);

    for(teacher t : v){
        switch (t.gender){
            case Gender::MALE: //男
                cout <<"男老师" << endl;
                break;
            case Gender::FEMALE:
                cout <<"女老师" << endl;
                break;
            default:
                cout <<"性别错误" << endl;
                break;
```

```
        }
    }
    return 0 ;
}
```

练习

> 一个容器里面有若干个学生，现在需要统计一下里面的男生有多少个，女生有多少个。 请使用枚举来定义性别。

三、异常处理

1. 异常处理

异常时指存在于运行时的反常行为，这些行为超出了函数的正常功能的范围。和python一样，c++ 也有自己的异常处理机制。 在c++中，异常的处理包括 `throw表达式` 和 `try 语句块` 以及 `异常类` 。如果函数无法处理某个问题，则抛出异常，并且希望函数的调用者能够处理，否则继续向上抛出。如果希望处理异常，则可以选择捕获。

```
void test(){
    try{
        autoresult = do_something();
    }catch(Some_error){
        //处理该异常
    }
}

int dosomething(){

    if(条件满足){
        return result;
    }else{
        throw Some_error(); //抛出异常
    }
}
```

2. 不使用异常机制

- 终止程序

> 可以使用 abort | exit 来终止程序的执行

```
int getDiv( int a , int b){
    if(int b == 0 ){
            abort(); // 或者是 exit(4) //括号内为错误的代码，可以使用常量定义
        }
    return a / b;
}
```

- 显示错误代码

> 与直接终止程序的突兀对比，错误代码显得更为友好些，同时也可以根据错误代码给出相应的提示。

```
int getDiv( int a , int b){
    if(int b == 0 ){
            //abort(); // 或者是 exit(4) //括号内为错误的代码，可以使用常量定义
        return -1000001;// 外部可以对此代码进行处理
        }
    return a / b;
}
```

3. 使用异常机制

1. 捕获异常

> 若程序想对异常进行处理，以达到让程序继续友好的执行下去，可以使用捕获异常。
> exception 是所有异常的父类，runtime_error 可以作为运行时出现的异常捕获。一旦发生异常，那么后面的语句将不会执行。
>
> 一般捕获异常，采用try{} catch(){}的结构来捕获异常，catch可以有多个。 可以使用catch(...) 来表示捕获所有异常 ，但它必须出现在所有catch的最后。

```
try{
    //执行的代码逻辑
}catch(runtime_error err ){ //捕获的异常类型。
    //捕获到异常，执行的逻辑
    cout << err.what() << endl; //打印错误信息
}
```

2. 抛出异常

> 函数内部如果不想处理异常，可以选择抛出异常（throw），进而由调用的它函数处理，若该函数仍未处理，则继续往上抛出。注意：若抛出的语句位于try语句块内，则优先匹配try语句匹配的异常，若没有匹配上，才选择向上抛出。 throw可以抛出任意类型的异常，要求是这些类型必须是这些类的对象可复制和移动。同样抛出异常时，后面的语句不会执行。

```
int calcDiv(int a, int b){

    if(b == 0){
        throw  runtime_error("除数不能为0 ");
    }
    return a / b;
}
```

3. noexcept

> 如果预先知道某个函数不会抛出异常，那么可以在函数定义的参数列表后面跟上关键字 noexcept ，通常会存在于移动构造函数 和 移动赋值函数中。即便某个函数声明不会抛出异常，但是在内部真的抛出异常，编译器仍然允许编译通过，但是在运行时为了确保不抛出异常的承诺，会调用terminate 终止程序的执行，甚至不会向上传递异常。

```
stu(stu && s) noexcept { //移动赋值函数
}

void operator=(stu && s) noexcept{ //表示不会抛出异常。

}
```