extern

通过变量的声明和定义解释其意义，int j =>声明并定义了j；extern int j =>仅仅声明；

指针与引用

- 引用便是变量的别名：int a = 100; int &b = a; =>b是a的别名。
- 指针便是指向变量地址的变量。int a = 100; int *b = &a; b为指向a的指针。

const/constexpr

默认情况下，const修饰的仅在当前文件内有效。

const int b = 100;

const int bb = 200;

const int * a = &b; -> 这是指向常量的指针 -> 也叫底层const。

int *const aa = &bb; -> 这是常量指针 -> 也就是 顶层const。

注：普通的常量都是顶层const；只有指针既有顶层const，也有底层const。

decltype

int i=1,*j=&i;

decltype((*j)) a = i; // 加双层括号都算表达式，永远为引用类型

decltype(*j) aa ; //报错，不知道怎么初始化

命名空间using & namespace ->然后知道了域以及 ::

namespace std{ ~ } => 申明并定义了一个命名 空间；

using namespace std; => 说明std域里面所有变量或者方法都不用加std；

using namespace std::cout; => 说明在std域里面，只有cout 不用加std；

注：头文件不要使用命名空间，一方紊乱；

string

string的长度 " ".size(); -> 返回的是一个无符号类型的值，通过string::size_type可以知道。

字符串用+连接过程中必须有声明的string变量参与，否则报错。从左到右两两依次相加，都存在变量参与就不会报错，这样理解就好。

字符数组 在存放字符串时注意大小，后需要加一个 '\0' => char a[3] = "ab"; //刚好，多了就报错

标准类库vector:标准对象的集合

vector a; => 声明并定义的一个容器a,存在默认初始化;

有一下初始化方法：

vector v1；

vector v2(v1);

vector v3 = v1; <=> 上一个

vector v4(n,val); => 包含n个重复的 val，如果没有val，则n个初始化为T的值

vector v5{a,b,c,d.....}

vector v6 = {a,b,c,d.....}

case 标签必须是常量表达式

范围for循环中，如果要进行写操作，循环变量必须声明成引用类型。

```
for ( declaration variable: expression)     // 说的是variable
    statement
```

关于参数传递及函数返回值

```
int main(int argc,char* argv[]){} // 其中argc为参数个数，argv[0]为参数名字，argv[1]为第一个参数...
```

```
int (*func(int i))[3]{  } //这个一直没实现，不知道什么情况。
auto func(int i) -> int(*)[3] { }   // 这个可以实现
decltype ( 数组) *func(int i) {     }  // 这个也实现了
```

含有可变形参的函数

C++11 标准提供了两种主要的方法：

1. 如果实参类型相同，可以传递一个名为initializer_list 标准类库
2. 如果实参不同，也就所谓的可变参数模板

```
initializer_list<T> lst;     // 默认初始化，T类型的空列表
initializer_list<T> lst{a,b,c...};  // 初始化，值为a，b，c...副本
lst2(lst); lst2 = lst;  // 拷贝或赋值一个initializer_list对象，共享列表中元素
lst.size()
lst.begin()
lst.end()
```

函数默认实参、内联函数、constexpr函数

```
void func(int a = 1,int b = 2, int c){ }         // 默认实参
inline int func(int a){     }        // 内联函数，编译时将展开，执行更快，用于规模小，频繁调用的语句块
constexpr int new_sz() { return 42;}     // 是指能用于常量表达式的函数，函数形参和返回值必须都是字面量类型，且必须有返回值
constexpr int foo = new_sz();
```

调式相关知识

```
assert(expr); //expr为false，则报错停顿了，true则没事，继续；需引入#include <cassert>
#define NDEBUG //使所有调试失效，必须放在最前面
c++编译器为每个函数定义了几个静态变量：
    cout << __func__ << endl;    存放函数名字
    cout << __FILE__ << endl;    存放文件路径
    cout << __LINE__ << endl;    存储行号
    cout << __TIME__ << endl;    存放时间
    cout << __DATE__ << endl;    存放日期
```

函数指针

```
bool (*pf)(const string&,const string&); // 未初始化的函数指针pf
bool func(const string& a, const string& b){     }    // 一个函数
pf = func;  //使pf指向func函数。
pf = &func ;  //同上，可以省略&
// 用decltype获取函数类型，返回的是函数类型，而不是函数指针；
decltype(func) *ppf(const string&,const string&);    // 一个未初始化的函数指针ppf
```

直接初始化与拷贝初始化

```
string s1("nice");  //直接初始化
string s2 = "nice"; // 拷贝初始化
string s3 = {"nice"}     // 这是什么初始化，还没彻底搞清楚
```

注1：无符号类型会将负数的第一位变成1，所以会变成很大的数

注2：在C语言中的库一般用name.h，现在在C++用cname替换，但内用一样的，但更符合C++规范