杂项

windos和linux的操作区别

操作系统	路径标识符	可执行文件的扩展名	文件结束符
windows	\ 反斜杠	可以不加	ctrl z
linux	/ 斜杠	要加	ctrl d

linux 运行progl.cc生成可执行文件progl `g++ -o progl progl.cc`

多选注释最好每行前加*，注释代码用单行注释

关于include导入头文件

```
#include<标准库头文件>
#include"自定义头文件.h"
```

文件重定向：`$ addItems <infile> outfile`

利用加法程序addItems从infile文件读取销售记录，将输出结果写入outfile中，两文件都处于当前目录中

8比特=1字节，4（或8）字节=1字

关于如何选择数值类型

1. 明确数值大于0，用无符号类型，如 `unsigned int`
2. 整型运算直接用 `int` ，数值大用 `longlong`
3. 数值运算中不用 `char` 和 `bool` ， `char` 的符号问题在各种机器上不一致
4. 浮点型用 `double` ，双精度运算代价不大于单精度， `long double` 运算消耗大，一般不用

类型转换：

1. 当把 `bool` 值赋给非 `bool` 值时，结果为0或1
2. 赋给无符号类型超范围外的值时，结果为该值对总数取模后的余数，如：unsigned char (0~255),给-1，得255
3. 赋给有符号类型范围外的值时，结果是未定义的
4. -> 运算时不混用有符号类型和无符号类型

"字符串"其尾部是空字符，/0

分开书写的字符串也是一个整体

```
std::cout<<"a really really long string litreal"
        "that spans twos lines"<<std::endl;
```

关于用{}进行初始化：

```
long double ld = 3.1415926536
int a{ld},b={ld}; // 会报错，因为存在丢失信息的危险
```

默认初始化：

内置类型的变量在函数体外初始化为0，在函数体内不被初始化（未定义，不可访问）

类的默认初始化由其自己决定（如string类默认初始化是一个空串）

建议初始化每一个内置类型的变量

关于 extern

```
extren int i; //只声明不定义
extren double pi = 3.1415; //定义
```

变量命名规范

1. 有意义
2. 变量名小写，类名首字母大写
3. 单词间有区分，驼峰或者下划线

变量名作用域

始于声明语句，结束于}

作用变量时最好在附近定义

关于引用（左侧引用）：

```
int ival = 1;
int &refval = ival; // 引用refval 必须初始化，且右值只能是一个对象，不能是字面值或表达式
```

指针的四个状态：

1. 指向一个对象
2. 指向紧邻对象所占空间的下一个位置
3. 空指针：没有指向任何对象
4. 无效指针，上述情况之外，不可访问

空指针的生成

```
int *p1 = nullptr; // 等价于 int *p1 = 0;
int *p2 = NULL; //等价于 int *p2 = 0;

// 即使 int a = 0;也不能将其直接赋给指针
int a  = 0;
p1 = a; // 错误
```

任何非0指针对应的有效值都是true

建议初始所有指针

关于 void* 指针

1. 可以存储任意类型对象的地址

引用本身不是一个对象，指针是

```
int * & r = p; // 对指针p的一个引用
```

关于 const （const对象必须初始化）

1. 默认情况下，const对象仅在文件内有效，当多个文件出现同名的const变量时，等同于在不同文件中分别定义了独立的变量 如果需要在一个文件中定义const变量，而在多个文件中可以声明使用，则在声明和定义前加上 extern 即可，表示该变量并非本文件独有
2. 当常量引用绑定到另一种类型时，非法：

```
double dval = 3.14;
const int &ri = dval; //非法的

//上面的代码等价于：
const = int temp = dval;
const int &ri = temp; // 如果ri不是常量，ri改变不会影响dval，无意义
```

3. 指针和const

```
// 指针的类型必须与所指对象保持一致，但对于const有例外
double dval = 3.14
const double *cptr = &dval; // cptr指向一个const double，即不能通过cptr改变dval的值，
底层const

// 指针本身是对象，所以有常量指针 *const ，指向的地址不能改变，本身是常量对象，是顶层co
nst
int *const curErr = &errNumb

// 小结 ：对于一个对象是，指向他的引用和指针可以多const但不能少const
```

定义别名：

```
typedef double wages
using SI = sales_item // c++11
```

auto类型说明符

```
// auto 一般会忽略顶层const，保留底层const

const int ci = i;
auto b = ci; // b是int型，ci的顶层const被忽略
auto e = &ci; // e 是一个指向const int 的指针，底层const保留

// 可以通过手动加const保留顶层const
const auot f = ci;
```

decltype类型指示符

```
decltype(f()) sum = x; // 与auto不同，decltype指导出的变量包括顶层const

// decltype 和 引用
itn i =42 ,*p = &i,&r = i; // i 是整型，p是指针，r是引用

decltype(r+0) b; // r+0是整型
decltype(*p) c; // 错误，*p是&int，必须初始化

/*
    * 补充：
    *decltype((variable)) 的结果是引用
    *decltype(variable) 的结果只有当variable本身是一个引用时都是引用
*/
```

自定义数据结构{}后加；结束

预处理器——头文件保护符

```
#ifdef / # ifndef
#define
```