# 运算符优先级

第一梯队

[] () . ->

# 指针数组

指针数组本意是数组，数组元素存放的是指针。数组的大小由指针数量决定。

创建方式

int\* data[3]{0 };

操作

data[0] = new int(0);

### 动态构建指针数组

构建

int\*\* pArray = new int\*[10];//动态构建了一个指针数组，长度是10

使用

pArray[0] = new int(10);

# 数组指针

指向数组的指针，首先这是一个指针，它指向一个数组。大小是一个指针的大小，无法使用数组指针确定数组的大小，因为它只是一个指针

操作

Int data[n] = {1};

Int (\*p)[n] = &data;//声明数组指针并通过取数组的指针进行赋值

使用

获取数组内的值data[n] = \*(\*p + n);

或 （\*p）[index]

数组变量值是个特殊的数据，他本身就是指针，指向数组空间中第一个数据的地址

data[0] 等同于 \*data

data[1] 等同于 \*(data+1)

注意：

1. 时刻要注意你操作的是指针数组还是数组指针，两者从最后一个词语能确定类型。
2. 指针数组操作过程中注意，他本身也是数组，也具备数组地址指向的是第一个数据地址
3. 操作数组指针切记，他指向的是一个数组的地址，那么对它做接引用操作，获得的是指向的数组的地址，那么这个地址就是数组中的首元素地址
4. 数组本身就是一个指针，可以进行基本的加减法运算，加减的操作相当于是在做指针位置移动操作，移动大小与数组的数据类型有关
5. 数组进行函数参数传递时，传递的是指针值，所以无法使用for冒号进行迭代操作

# 可变参数

程序设计中，经常会遇到传入的参数个数有变动的情况，我们在参数个数不定的时候可以选择使用可变参数来提高程序的编写逻辑，用…代替可变参数

格式

返回类型 函数名(变量类型 变量名，…)

void Say(int i, ...)

### 参数类型相同的可变参数

函数中必须提供一个参数，并且可变参数符号在函数末尾。调用时，注意填入的可变参数类型必须和可变参数符号之前的参数类型相同。

void He(int i, ...)

调用

He(2, 66, 45);

函数中获得参数内容

同类型可变参数在参数列表中被构建为数组传递，并且在参数个数基础上增加一个元素到末尾，用于标记参数尾端。（一般为0，指针也适用）

void He(int i, ...)

{

int \*p = &i;

*cout* << p[0] << *endl*;//2

*cout* << p[1] << *endl*;//66

*cout* << p[2] << *endl*;//45

*cout* << p[3] << *endl*;//0

*cout* << p[4] << *endl*;//无意义数据

}

如果参数为指针注意检查null（下面是指针操作）

void Say(int\* pN, ...)

{

int\*\*ppN = &pN;

while (\*ppN)

{

*cout* << \*\*ppN << *endl*;

ppN++;

}

}

### 参数类型不同的可变参数

首先不建议使用参数不同的可变参数，如果有需求，建议使用函数重载。

原因：参数类型不同，那么函数内部进行编写逻辑解析参数时**必须要知道参数类型，否则无法完成参数解析**！这本身增加了工作量。可以完全使用运算符重载解决此类问题，并且更加清晰。

# 预编译

在编译阶段帮助编译器进行第一次代码分析，扫描整个程序工程源代码，对其进行初步的转换，产生新的源代码，提供给程序使用

### #符号

标注了预编译的起始点，并且#是该行的第一个字符，#符号后是指令关键字

#define，#undef指令

#define定义了一个标识符及一个串，标识符为该宏的名称，源程序在预编译阶段会将程序中出现的所有宏用其定义的串进行替换么，称为宏替换

结构

#define 宏名 字符串

#define 宏名（参数表） 宏体 //**括号和宏名之间不能留有空格**

#undef指令，取消一个已经定义过的宏

宏一般使用大写字母进行声明，宏属于纯粹的文本替换，不会进行逻辑检查，所以在使用过程中需要万分注意，尤其是运算符优先级问题，可能会导致奇怪的问题。

宏的操作技巧

1. \用来切换行，编译器遇到宏中包含\时，将向下读取一行进行拼接

#define SayHello(*X*) *cout*<<\

*X*<<\

*Endl*

1. 使用#进行双引号标记

#define SIGN(*X*) #X //输出结果 "X" 使用双引号将内容进行圈选

1. 借助##可以进行前后语句的拼接

#define *INT* int

#define *PINT* *INT*##\*//拼接结果 int\*

1. 使用#@可以构建单引号操作

#define SIGN(*X*) #@X//结果最后是 ‘X’//单引号圈选

宏常用来做什么

1. 定义数据常量。但是不如使用const关键字声明的常量，因为宏本身不参与语法检查
2. 进行简单的逻辑代码替换，方便我们复用代码，简化代码编写量
3. 用来构建简单的逻辑函数。目的降低函数调用开销。但是会带来代码的膨胀，阅读性下降。
4. 简化复杂的数据类型编写，例如指针
5. 批量生成逻辑代码

宏的问题

宏本身不是逻辑单元，是文本操作，不参与运行时态决策

### #if #elif #else #endif指令

逻辑条件指令，可以根据给出的条件对程序代码有选择性的进行筛选组合，方便构建不同的操作逻辑工程，例如针对平台代码

**注意：**

1. 处理逻辑中，所有的条件参考内容，需要使用预处理指令构建的参考数据（例如宏）
2. #endif指令标记了逻辑的结束
3. #if指令标记了逻辑的开始
4. 此逻辑不会出现在程序运行阶段
5. 表达式支持逻辑关系运算符，逻辑运算符

### #ifdef #ifndef #endif指令

用来检查某些宏是否被定义，一般用来防止宏或是头文件重复引入（新版本C++使用了另一个编译指令）

### #Line指令

\_\_LINE\_\_ 显示当前指令所在的行数

\_\_FILE\_\_显示当前指令所在的文件夹位置

主要用途可以用来在调试信息打印时，定位问题所在

### #error 指令

一般用来做逻辑预处理时提供错误指示，当编译时提供错误信息

格式

#error 版本错误

编译时，控制台将提示错误“版本错误”

这是编译阶段出现的提示，不是运行阶段提示

### #pragma command

该指令用来设定编辑器的状态或是指示编辑器完成一些特定的动作，他有许多不同的参数

1. #pragma once

保证头文件被包含一次，并且只包含一次

1. #pragma message

可以在编译阶段，向控制台输出一句话

#pragma message("OK")