## 数组

1. 数组维度必须是常量表达式。

unsigned int N **=** 42**;**

int a**[**N**];**//错误，N不是常量表达式。常量表达式的意思是：数值不变，且在编译期值可知。在g++中，上述语句正常工作，可能原因是编译器扩展，也有说c++标准c99支持VLA（variable length array）

1. 四不能：不能使用auto自动推断数组元素类型，数组元素不能是引用类型，数组不能拷贝，数组不能赋值。

int array**[**2**]** **=** **{**1**,**2**};** //正确

auto a**[**2**]** **=** **{**1**,**2**};** //错误，数组元素类型不能使用auto自动推断

int **&** b**[**2**]** **=** /\*\*/ //错误，数组元素类型不能是引用

int c**[]** **=** array**;** //错误，数组不能拷贝

c **=** array**;** //错误，数组不能直接赋值

1. 使用auto获得的数组类型是指针类型；使用decltype获得数组类型。

auto a **=** array；//a的类型是指针类型

decltype（array）a **=** **{**1，2**}**；//decltype（array）是数组类型

1. begin（），end（）函数，头文件iterator中。
2. 数组索引类型是int 而不是unsigned int，索引可以为负值。

int array**[]** **=** **{**1**,**2**,**3**,**4**,**5**};**

int **\***p **=** **&**array**[**2**];**

int k **=** p**[-**2**]**//正确，表示array[0]

1. c风格字符串

const char **\*** a**[]** **=** “aaa”**;**

const char**\*** b**[]=**”bbb”**;**

比较：strcmp，若使用==比较的是指针值

连接：strcat，若使用+，表示两个指针相加，错误

1. 旧接口

string s **=** “sss”**;**

const char**\*** a**[]** **=** s**.**c\_str**();**//a[4] = “sss\0”;

1. 多维数组

除了最后一层范围for循环，其他for循环必须使用引用。

int a**[**2**][**2**]** **=** **{{**1**,**1**,},{**2**,**2**}};**

**for(**auto a1**:** a**){**

**for(**auto a2 **:** a1**){**//错误，因为a1退化为指针，类型为 int\* ，对int\*遍历是错误的

cout**<<**a2**<<**endl**;**

**}**

**}**

int a**[**2**][**2**]** **=** **{{**1**,**1**,},{**2**,**2**}};**

**for(**auto**&** a1**:** a**){**

**for(**auto a2 **:** a1**){**//正确，a1类型为int [2]

cout**<<**a2**<<**endl**;**

**}**

**}**

## 表达式

1. 优先级、结合性、求值顺序

求值顺序确定的只有逻辑与（&&），逻辑或（||），逗号运算符（，），双目运算符（？：），其他运算符不保证运算顺序：

a**+**b**\***c**+**d**;**//优先级保证先算b\*c，结合性保证接下来计算a+然后+d

cout**<<**i**<<++**i**<<**endl**;**//不保证先做++i还是先输出i，结果未定义

f**()+**g**();** //不保证先计算f()；如果f（）和g（）由共用变量，结构未定义

1. typedef

**typedef** int int\_array**[**4**];**//表示int\_array是数组元素类型为int，维度为4

int\_array a**;**//等价与int a[4];

**typedef** char**\*** P**;**

const char**\*** p1**;**//等价与char const \*p1;底层const

const P p2**;**//等价与 P const p2; 即就是 char\* const p2;顶层const

1. decltype（X）当X为左值时，得到类型为引用类型

int **\*** p **;**

decltype**(\***p**)** a**;**//因为解引用得到的是左值，所以 a 的类型是 int&

decltype**(&**p**)** b**;**//取值符 & 获得右值，所以得到 b 的类型是 int\*\*

1. 算术运算符

|  |
| --- |
| 一元正号 +  一元负号 - |
| 乘法运算符 \*  除法运算符 /  取模运算符 % |
| 二元加法 +  二元减法 - |

* 1. 运算结果都为右值
  2. 一元正号、加法、减法可用于指针
  3. 取模运算符 % 操作数都为整数，且符号和操作符左边的数相同：

21 **%** **-**5 **=** 1； 21**/-**5**=-**4；//必须保证 m = （m/n）\*n + m%n = m ，xain't所以有余数符号和操作符左边数相同

**-**21 **%** 5 **=** **-**1； **-**21**/**5**=-**4；

* 1. 整数除法向0取整：

**-**1**/**3**=**0；

**-**4.3**=-**1；

1**/**3**=**0；

1. 关系运算符

|  |
| --- |
| ！ |
| >  >=  <  <= |
| ==  != |
| && |
| || |

只有？：、&&、||、，这四种运算符保证求值顺序

1. 赋值运算符

赋值运算符优先级较低，复合赋值运算符和赋值运算符之间有差异：

a **+=** b**;**//a取值一次

a **=** a **+** b**;**//a取值两次

1. 位运算符

|  |
| --- |
| 求反 ~ |
| <<  >> |
| & |
| 异或^ |
| | |

1. 整型提升
2. 有符号右移未定义，一般来说用于无符号
3. int只保证16位，long保证32位
4. 优先级小于算术运算符，大于关系运算符、赋值运算符、条件运算符

左结合性

1. 条件运算符

只有当两个表达式都为左值时，结果为左值。

1. sizeof
   1. 优先级和解引用\*等同

右结合性

* 1. 优先级等同与解引用\*，右结合律

int **\***p **=** **nullptr;**

**sizeof** **\***p**;**//右结合性兵器优先级等同于 \* ，不对表达式求值所以正确

* 1. c++11新特性允许使用域运算符获取成员大小

class ClassA**{**

public**:**

int a**;**

**}**

**sizeof(**ClassA**::**a**);**//正确，一般来说访问类成员需要类对象，sizeof无须我们提供一个对象

1. 类型转换
   1. 隐式转换：
      1. 算术提升

char a**;**

short b**;**

a**+**b**;**//a 和 b 都转换为 int

* + 1. 指针和数组

int a**[**2**]** **=** **{**2**,**1**};**

int f**(**int**\*** p**){...};**

f**(**a**);**//a 转换为指针

* + 1. 指针

int**\*** pi**;**

void**\*** pv **=** pi**;**//int\* pi转换为 void\* pv

* + 1. bool

**while(**cin**>>**a**){**//istream对象转换为bool类型

/\*...\*/

**}**

* + 1. const

int a **=** 8**;**

const int b **=** a**;**//int 类型 隐式转换为 const int

* + 1. 类类型定义

string s **=** "hello world"**;**//const char\* 隐式转换为 string

派生类转换基类

* 1. 显式转换
     1. static\_cast<type>()：任何具有明确定义的类型转换，并且没有底层const
     2. const\_cast<type>()：去掉const属性
     3. reinterept\_cast<type>()：位模式下对对象重新解释

int a **=** 8**;**

char**\*** p **=** **reinterpret\_cast<**char**\*>(&**a**);**

* + 1. dynamic\_cast<type>()：派生类转换为基类类型

## 语句控制

1. switch
   1. default不影响先匹配case

int a **=** 1**;**

**switch(**a**){**

**default:**cout**<<**"default"**<<**endl**;break;**

**case** 1**:**cout**<<**"1"**<<**endl**;break;**

**}**

//结果输出 1

* 1. 不能跳过变量的初始化语句

int a **=** 1**;**

**switch(**a**){**

**case** 0**:**

int b **=** 1**;** //#1

**break;**

**case** 1**:**

cout**<<**b**<<**endl**;**//错误，不允许跳过 #1

**break;**

**default:**

cout**<<**"default"**<<**endl**;**

**break;**

**}**

1. for

**for(**int i **=** 0 **;** **;** **++**i**)**//省略判断表示始终为true

**{**/\*...\*/**}**

1. while do…while
   1. do..while至少执行一次
   2. do…while()中不允许使用while中的变量。

**do{**

cout**<<**i**<<**endl**;**//错误，使用了while（）括号中的i

**}while(**int i **<** 0**);//注意有分号**

1. 4种跳转
   1. break；可用于switch；for；while（包括do…while）
   2. continue；不能用于switch
   3. goto；
   4. return；
2. try
   1. 没有finally关键字
   2. 没有catch则最终会调用系统的terminate
   3. 只有exception、bad\_cast、bad\_alloc默认初始化，不需要初始值

## 函数

1. 函数参数
   1. 函数实参不保证求值顺序
   2. 作用域和生命周期区别。作用域表示变量可见不可见，生命周期指该对象在内存中存在的一段时间
   3. 局部自动变量执行默认初始化（对于内置类型其值未定义），局部静态变量执行值初始化
   4. 编译器只能执行一次隐式类类型转换

class A**{**

A**(**string s**);**

void f**(**A a**);**

**}**

int main**(){**

f**(**"123"**);**//错误，“123” 类型为 const char\* ，编译器可以将其隐式转换为 string ，但是不能在使用构造函数将其转换为A

A a**(**"123"**);**//正确，“123” 隐式转换为 string ，然后使用构造函数

**}**

1. 函数返回值
   1. 优先级：等同于成员选择符 . 和 –>；结合性：左结合
   2. 左值右值：返回引用的时左值
   3. 返回数组指针：

int **(\***fun**(**int i**))[**10**];**//可以对fun的返回结果解引用，解引用的结果得到大小是10的数组；数组元素是int

auto fun**(**int a**)** **->**int**(\*)[**10**];**//尾置返回类型

int a**[**4**]={**1**,**2**,**3**,**4**};**

decltype**(**a**)** **\*** fun**(**int i**);**//decltype得到维数是4的数组类型，fun返回指向维数是4的数组的指针

1. 函数参数
   1. const形参忽略顶层const

int f**(**const int a**);**

int f**(**int a**);**//错误，等价于上一个函ge数，不能构成重载；赋值时顶层const被忽略

const int aa **=** 1**;**

f**(**a**);**//aa赋值给形参时，aa的const属性被忽略，不会匹配到f(const int a);

* 1. 数组形参
     1. 结束标志（c字符串的结束符’\0’）
     2. 传递数组长度
     3. 传递起始结束位置
  2. main函数参数,自定义参数从argv[1]开始

void main**(**int argc**,**char**\*** argv**[]){**

**for(**int i **=** 0 **;** i **<** argc**;++**i**){**

cout**<<**"argv["**<<**i**<<**"]:"**<<**argv**[**i**]<<**endl**;**

**}**

**}**

**./**main 'aa' 'bb'

输出：

argv**[**0**]:**main

argv**[**1**]:**aa

argv**[**2**]:**bb

* 1. 可变参数列表
     1. 参数列表：initializetor\_list

void f**(**initializer\_list**<**int**>** a**){**

**for(**auto p **:**a**){**

cout**<<**p**<<**" "**;**

**}**

cout**<<**endl**;**

**}**

int main**()**

**{**

f**({**1**,**2**,**3**,**4**});**

initializer\_list**<**int**>** a **=** **{**1**,**2**,**3**,**4**};**

initializer\_list**<**int**>** b **=** a**;**

initializer\_list**<**int**>** c**;**

c **=** a**;**

cout**<<**begin**(**a**)<<**endl**;** //输出0x46e038

cout**<<**begin**(**b**)<<**endl**;** //输出0x46e038；表名拷贝共享列表元素

cout**<<**begin**(**c**)<<**endl**;** //输出0x46e038；表名赋值共享列表元素

**return** 0**;**

**}**

* + 1. 可变参数模板
    2. c省略符号

1. 函数重载
   1. 顶层const不能作为重载条件

引用都是底层const，可以作为重载条件

* 1. cosnt\_cast 与重载

const string**&** f**(**const string**&** s**){}**//#1

string**&** f**(**string**&** s**){**//#2和#1构成重载

**return** **const\_cast<**string**&>(**f**(const\_cast<**const string**&>(**s**)));**

**}**

* 1. 重载作用域

int f**(**double**);**//#1

int f**(**string**);**//#2

int main**(){**

int f**(**int**);**//使得#1，#2不可见

f**(**"111"**);**//错误，f(string)不可见

f**(**2.3**);**//正确，调用 f(int),等价于 f(2);

**}**

名字查找发生于类型检查之前

1. 特殊用途语言特性
   1. 函数默认参数
      1. 使用：具有默认值的参数必须全部放在参数列表右边

void f**(**int j**,**int i **=** 1**);**//正确

void f**(**int i **=** 1**,** int j **);**//错误

* + 1. 声明：函数可以重复声明，但是同一作用域下，默认参数不能被重复赋值

void f**(**int i**,** int j**);**

void f**(**int i **=** 1**,**int j**=**2**);**//正确

void f**(**int i**,** int j**=**2**);**

void f**(**int i **=** 1**,**int j**=**2**);**//错误，j被重复赋值

* + 1. 作用域

int di **=** 1**;**

int dj **=** 2**;**

void f**(**int i **=** di**,**int j**=**dj**);**

int main**(){**

f**();** //输出3

di **=** 2**;**

f**();** //输出4，参数赋值发生在调用阶段，所以 i = di = 2;

int dj **=** 1**;**

f**();** //输出4，dj 隐藏 外层dj，但是没有改变参数中的dj，名字解析发生于赋值之前，此时，i = 2 ,j = 2

**return** 0**;**

**}**

void f**(**int i**,**int j**){**

cout**<<**i**+**j**<<**endl**;**

**}**

* 1. inline和constexpr
     1. inline只是请求，并非要求
     2. constexpr能用于常量表达式的函数
     3. constexpr参数、返回值是字面值类型，只能有一个return

constexpr int g**(**int a**){**

**if(**a **<** 0**){**

**return** **-**1**;**

**}**

**else{**

**return** 1**;**

**}**

**}**

int main**(){**

g**(**1**);**

**return** 0**;**

**}**

error**:** body of constexpr function ‘constexpr int g**(**int**)**’ **not** a **return-**statement

只能出现一个 return 语句

* + 1. constexpr函数隐式内联
    2. constexpr不一定返回常量表达式

constexpr int g**(**int a**){** //若 a 是常量表达式，则函数返回常量表达式；若 a 不是常量表达式，函数返回不是常量表达式

**return** a**\***2**;**

**}**

int main**(){**

int i **=** 1**;**

cout**<<**g**(**i**)<<**endl**;** //正确，说明函数不一定返回常量表达式

int sa**[**g**(**1**)];** //正确，g(1) 是常量表达式

int sa**[**g**(**i**)];** //c++ primer说是错误，g(i)不是常量表达式，g++测试正确，应该是编译器扩展

**return** 0**;**

**}**

* 1. assert和NDEBUG
     1. assert依赖于NDEBUG，如果定义了NDEBUG，则assert什么也不做

#include<iostream>

#include<cassert> //包含头文件

**using** **namespace** std**;**

int main**(){**

int i **=** 1**;**

assert**(**i**>**1**);**

**return** 0**;**

**}**

输出：int main**():** Assertion `i**>**1' failed**.**

g**++** **-**D NDEBUG test**.**cpp **-**o main //作用是在程序开始位置添加 #define NDEBUG

**./**main

2

* + 1. NDEBUG

\_\_func\_\_:函数名

\_\_TIME\_\_:编译时间

\_\_DATE\_\_:编译日期

\_\_LINE\_\_:当前行号

cout**<<**\_\_func\_\_**<<**endl**;**

cout**<<**\_\_DATE\_\_**<<**":"**<<**\_\_TIME\_\_**<<**endl**;**

cout**<<**\_\_LINE\_\_**<<**endl**;**

1. 函数确认
   1. 函数范围
      1. 函数名
      2. 作用域
   2. 可行函数
      1. 函数参数个数
      2. 参数类型
   3. 最佳匹配
      1. 所有参数都不劣于其他
      2. 至少有一个参数优于其他

int f**(**int**);**

int f**(**float**);**

f**(**3.4**)**//产生二义性，3.4是double，其到转换到float和转换到int优先级相同，所以错误

int f**(**int**,**double**);**

int f**(**double**,**int**);**

f**(**1**,**2.4**);**//产生二义性，不满足最佳匹配的两个条件

* 1. 类型转换优先及顺序
     1. 精确匹配：完全匹配，数组和函数与指针的转化，顶层const忽略
     2. const转换
     3. 类型提升
     4. 算术转换
     5. 类类型转换

1. 函数指针
   1. 函数指针的类型包括返回值和参数类型、函数指针不存在类型转换
   2. 重载函数

int f**(**int**);**

int f**(**float**);**

int **(\***p**)(**int**)** **=** f**;**

* 1. 函数指针作为参数

int f**(**int**);**

**typedef** int **(\***FE**)(**int**);** //#1 PE类型是 int (\*)(int)

**typedef** decltype**(**f**)** FE**;** //#2 等价于 #1

**typedef** decltype**(**f**)** **\***PE**;** //#3 对于函数参数来说，其等价于#2 ；对于返回值来说不等价，编译器不能自动将函数类型转换为指针

**using** PE **=** int **(\*)(**int**);**//#4 等价于#3

**using P = int(int);// #5 等价于#2**

* 1. 函数指针作为返回值

总结：函数作为参数时：函数类型和函数指针等价

函数作为返回值：返回类型不能时函数类型，函数类型不能自动转换为函数指针

**using** PE **=** int **(\*)(**int**);**//#4 等价于#3

**using** P **=** int**(**int**);**// #5 等价于#2

F g**(**int**);**//错误，返回类型是函数类型

PE g**(**int**);**//正确，返回类型时函数指针