

Variable Types

在名称中只能使用字母字符、数字和下划线 (_) 。

名称的第一个字符不能是数字。

区分大写字符与小写字符。

不能将 C++关键字用作名称。

以两个下划线或下划线和大写字母打头的名称被保留给实现（编译器及其使用的资源）使用。以一个下划线开头的名称被保留给实现，用作全局标识符。

Integer

short >=16 **[2^15-1] 32,767**

int >=short(32) **[2^31-1] 2,147,483,647**

long >=32 && >=int(32)

longlong >=64 && >=long(64) **[2^63-1] 9,223,372,036,854,775,807**

unsigned (31st bit is not sign

size_t[type of sizeof’s return type]32/64bits

<stdint>intx_t uintx_t INTx_MIN

floating

float 32 7digits accuracy

```
float f1 = 2.34E+10f;
float f2 = f1 + 10; // but f2 = f1
```

double 64 15digits

long double 64 15digits

bool 8 nonzero->>true zero->>false

char 8 signed/unsigned **[2^7-1] 127**

Pointer

int a;

int * const p= a; 不能改变指针取址

int const * p1=a; 不能通过指针修改数据

const int *p2=a;

smart pointer


reference especially for a class object

- using Declarations and using Directives
 - Declarations make particular identifiers available
 - Using directive makes the entire namespace accessible

```
namespace Jill {
    double bucket(double n) { ... }
    double fetch;
    struct Hill { ... };
}

using Jill::fetch; // a using declaration

char fetch;
int main()
{
    using Jill::fetch; // put fetch into local namespace
    double fetch; // Error! Already have a local fetch
    cin >> fetch; // read a value into Jill::fetch
    cin >> ::fetch; // read a value into global fetch
    ...
}
```

Namespace 

Compound

array, vector sizeof: size in bytes

string end with \0

```
#include<string> [lec4][lab4]
```

struct

```
typedef
struct _Student{
    char name[4];
    int born;
    bool male;
} Student;
```

#typedef + alias Student; no need “struct Student s”=>Student s ✓

• In order to align the data in memory, some empty

```
structpadding.cpp
struct Student1{
    int id;
    bool male;
    char label;
    float height;
};

struct Student2{
    int id;
    bool male;
    float height;
    char label;
};
```

union

```
union ipv4address{
    std::uint32_t address32;
    std::uint8_t address8[4];
};

sizeof(union ipv4address) is 4.
```

Union	Address
address32	0
address32	4
address32	8
address32	12
address32	16
address32	20
address32	24
address32	28
address32	32
address32	36
address32	40
address32	44
address32	48
address32	52
address32	56
address32	60
address32	64
address32	68
address32	72
address32	76
address32	80
address32	84
address32	88
address32	92
address32	96
address32	100
address32	104
address32	108
address32	112
address32	116
address32	120
address32	124
address32	128
address32	132
address32	136
address32	140
address32	144
address32	148
address32	152
address32	156
address32	160
address32	164
address32	168
address32	172
address32	176
address32	180
address32	184
address32	188
address32	192
address32	196
address32	200
address32	204
address32	208
address32	212
address32	216
address32	220
address32	224
address32	228
address32	232
address32	236
address32	240
address32	244
address32	248
address32	252
address32	256
address32	260
address32	264
address32	268
address32	272
address32	276
address32	280
address32	284
address32	288
address32	292
address32	296
address32	300
address32	304
address32	308
address32	312
address32	316
address32	320
address32	324
address32	328
address32	332
address32	336
address32	340
address32	344
address32	348
address32	352
address32	356
address32	360
address32	364
address32	368
address32	372
address32	376
address32	380
address32	384
address32	388
address32	392
address32	396
address32	400
address32	404
address32	408
address32	412
address32	416
address32	420
address32	424
address32	428
address32	432
address32	436
address32	440
address32	444
address32	448
address32	452
address32	456
address32	460
address32	464
address32	468
address32	472
address32	476
address32	480
address32	484
address32	488
address32	492
address32	496
address32	500
address32	504
address32	508
address32	512
address32	516
address32	520
address32	524
address32	528
address32	532
address32	536
address32	540
address32	544
address32	548
address32	552
address32	556
address32	560
address32	564
address32	568
address32	572
address32	576
address32	580
address32	584
address32	588
address32	592
address32	596
address32	600
address32	604
address32	608
address32	612
address32	616
address32	620
address32	624
address32	628
address32	632
address32	636
address32	640
address32	644
address32	648
address32	652
address32	656
address32	660
address32	664
address32	668
address32	672
address32	676
address32	680
address32	684
address32	688
address32	692
address32	696
address32	700
address32	704
address32	708
address32	712
address32	716
address32	720
address32	724
address32	728
address32	732
address32	736
address32	740
address32	744
address32	748
address32	752
address32	756
address32	760
address32	764
address32	768
address32	772
address32	776
address32	780
address32	784
address32	788
address32	792
address32	796
address32	800
address32	804
address32	808
address32	812
address32	816
address32	820
address32	824
address32	828
address32	832
address32	836
address32	840
address32	844
address32	848
address32	852
address32	856
address32	860
address32	864
address32	868
address32	872
address32	876
address32	880
address32	884
address32	888
address32	892
address32	896
address32	900
address32	904
address32	908
address32	912
address32	916
address32	920
address32	924
address32	928
address32	932
address32	936
address32	940
address32	944
address32	948
address32	952
address32	956
address32	960
address32	964
address32	968
address32	972
address32	976
address32	980
address32	984
address32	988
address32	992
address32	996
address32	1000
address32	1004
address32	1008
address32	1012
address32	1016
address32	1020
address32	1024
address32	1028
address32	1032
address32	1036
address32	1040
address32	1044
address32	1048
address32	1052
address32	1056
address32	1060
address32	1064
address32	1068
address32	1072
address32	1076
address32	1080
address32	1084
address32	1088
address32	1092
address32	1096
address32	1100
address32	1104
address32	1108
address32	1112
address32	1116
address32	1120
address32	1124
address32	1128
address32	1132
address32	1136
address32	1140
address32	1144
address32	1148
address32	1152
address32	1156
address32	1160
address32	1164
address32	1168
address32	1172
address32	1176
address32	1180
address32	1184
address32	1188
address32	1192
address32	1196
address32	1200
address32	1204
address32	1208
address32	1212
address32	1216
address32	1220
address32	1224
address32	1228
address32	1232
address32	1236
address32	1240
address32	1244
address32	1248
address32	1252
address32	1256
address32	1260
address32	1264
address32	1268
address32	1272
address32	1276
address32	1280
address32	1284
address32	1288
address32	1292
address32	1296
address32	1300
address32	1304
address32	1308
address32	1312
address32	1316
address32	1320
address32	1324
address32	1328
address32	1332
address32	1336
address32	1340
address32	1344
address32	1348
address32	1352
address32	1356
address32	1360
address32	1364
address32	1368
address32	1372
address32	1376
address32	1380
address32	1384
address32	1388
address32	1392
address32	1396
address32	1400
address32	1404
address32	1408
address32	1412
address32	1416
address32	1420
address32	1424
address32	1428
address32	1432
address32	1436
address32	1440
address32	1444
address32	1448
address32	1452
address32	1456
address32	1460
address32	1464
address32	1468
address32	1472
address32	1476
address32	1480
address32	1484
address32	1488
address32	1492
address32	1496
address32	1500
address32	1504
address32	1508
address32	1512
address32	1516
address32	1520
address32	1524
address32	1528
address32	1532
address32	1536
address32	1540
address32	1544
address32	1548
address32	1552
address32	1556
address32	1560
address32	1564
address32	1568
address32	1572
address32	1576
address32	1580
address32	1584
address32	1588
address32	1592
address32	1596
address32	1600
address32	1604
address32	1608
address32	1612
address32	1616
address32	1620
address32	1624
address32	1628
address32	1632
address32	1636
address32	1640
address32	1644
address32	1648
address32	1652
address32	1656
address32	1660
address32	1664
address32	1668
address32	1672
address32	1676
address32	1680
address32	1684
address32	1688
address32	1692
address32	1696
address32	1700
address32	1704
address32	1708
address32	1712
address32	1716
address32	1720
address32	1724
address32	1728
address32	1732
address32	1736
address32	1740
address32	1744
address32	1748
address32	1752
address32	1756
address32	1760
address32	1764
address32	1768
address32	1772
address32	1776
address32	1780
address32	1784
address32	1788
address32	1792
address32	1796
address32	1800
address32	1804
address32	1808
address32	1812
address32	1816
address32	1820
address32	1824
address32	1828
address32	1832
address32	1836
address32	1840
address32	1844
address32	1848
address32	1852
address32	1856
address32	1860
address32	1864
address32	1868
address32	1872
address32	1876
address32	1880
address32	1884
address32	1888
address32	1892
address32	1896
address32	1900
address32	1904
address32	1908
address32	1912
address32	1916
address32	1920
address32	1924
address32	1928
address32	1932
address32	1936
address32	1940
address32	1944
address32	1948
address32	1952
address32	1956
address32	1960
address32	1964
address32	1968
address32	1972
address32	1976
address32	1980
address32	1984
address32	1988
address32	1992

Class

C++ automatically provides the following member functions

> A default constructor if you define no constructors > A copy constructor if you don't define one > An assignment operator if you don't define one > A default destructor if you don't define one > An address operator

Abstraction component: public interface

Encapsulation component: gather the implementation details

Variable const member variable--[lec10 p23] static member variable--inline static size_t student_total = 0; or outside::

public protected[members and friends of itself&derived class father.n++X this.n++√] private

Function const member functions--A function promises NOT to modify the invoking object static member functions-- The only data members it can use are the static data members; public:can be invoked using the class name and the scope-resolution operator

Constructor Copy: operater= or Copy constructor[lec12 p13] static array will be copy normally

• A copy constructor. Only one parameter, or the rest have default values

```
MyTime:MyTime(MyTime & t){ ... }
```

```
MyTime t1(1, 59);
MyTime t2(t1); //copy constructor
MyTime t3 = t1; //copy constructor
```

• Default copy constructor:

- > If no user-defined copy constructors, the compiler will generate one.
- > Copy all non-static data members.

MyTime & MyTime::operator=(MyTime &){...}

```
MyTime t1(1, 59);
MyTime t2 = t1; //copy constructor
t2 = t1; //copy assignment
```

MyTime t2 = 80; constructor

MyTime t3; t3 = 80; = operator

• Question: Can we do the reverse?

```
Stonewt wolfe(285.7);
double hoet = wolfe; // ?? possible ??
```

• Yes, conversion function

- > User-defined type casts

```
Stonewt wolfe(285.7);
double hoet = double(wolfe); // syntax #1
double thisher = (double) wolfe; // syntax #2
```

- > Use a conversion function in this form

```
operator typeName();
```

- ✓ Must be a class method
- ✓ Must not specify a return type
- ✓ Must have no arguments

```
Stonewt::operator double()const
{
    return pounds;
}
```

↑ No return type but has return value

Type cast: Conversion constructor[lec11 p25] | Conversion function: explicit operator [typename] () [p26]

Destructor Passing an object as a function argument somehow causes the destructor to be called

Operator overloading Use only member functions to overload =, (), [], -> | cannot overload . .* ?:: Member/nonmenber: Time operator+(const Time &t)const||friend Time operator+(const Time &t1,t2)[achieve int+Time]

```
MyTime operator+(const MyTime & t) const
{
    MyTime sum;
    sum.minutes = this->minutes + t.minutes;
    sum.hours = this->hours + t.hours;

    sum.hours += sum.minutes / 60;
    sum.minutes %= 60;

    return sum;
}

MyTime & operator+=(const MyTime & t)
{
    this->minutes += t.minutes;
    this->hours += t.hours;

    friend std::ostream & operator<<(std::ostream & os, const MyTime & t)
    {
        std::string str = std::to_string(t.hours) + " hours and "
            + std::to_string(t.minutes) + " minutes.";
        os << str;
        return os;
    }
}

// prefix increment
MyTime& operator++()
{
    this->minutes++;
    this->hours += this->minutes / 60;
    this->minutes = this->minutes % 60;
    return *this;
}

// postfix increment
MyTime operator++(int)
{
    MyTime old = *this; // keep the old value
    operator++(); // prefix increment
    return old;
}
```

friend Complex operator +(double r, const Complex& other);

Friends functions---same access privileges as a member function of the class | can use class op inside[lec11] class---friend class class2::all member functions of class ClassTwo have the right to access the private and protected class members of ClassOne member functions

Dynamic memory&Class

Templates specialization non-type

Inheritance

Needs virtual destructor!

Public

- Public members of the base class
 - Still be public in the derived class
 - Accessible anywhere
 - Protected members of the base class
 - Still be protected in the derived class
 - Accessible in the derived class only
 - Private members of the base class
 - Not accessible in the derived class
- Can use base-class methods

Protected

- Public members and **protected** members of the base class
 - Be **protected** in the derived class
 - Accessible in the derived class only
- Private members of the base class
 - Not accessible in the derived class

Private

- Public members and **protected** members of the base class
 - Be **private** in the derived class
 - Accessible in the derived class only
- Private members of the base class
 - Not accessible in the derived class

Observations on using base-class methods

- ✓ A derived-class destructor **automatically** invokes the base-class **destructor**
- ✓ A derived-class constructor (in a member-initialization list)
 - **automatically** invokes the base-class **default constructor** if **don't specify** in list
 - **explicitly invokes** the base-class constructor specified in list
- ✓ A derived object **automatically uses** inherited base-class methods if the derived class **hasn't redefined** the method
- ✓ **Derived-class methods** can use the **scope-resolution operator** to invoke public and protected base-class methods

FriendClass & Nested types

friend Member Functions

- A single member function of a class is a friend.
- Different from friend functions.
- But very similar to a normal friend function.
- But... declaration problem ...

```
class Sniper
{
private:
    int bullets;
public:
    Sniper(int bullets = 0): bullets(bullets){}
    friend bool Supplier::provide(Sniper &);
};
```

only this function can access bullet

Where Declared in Nesting Class	Available to Nesting Class	Available to Classes Derived from the Nesting Class	Available to the Outside World
Private section	Yes	No	No
Protected section	Yes	Yes	No
Public section	Yes	Yes	Yes, with class qualifier

Exception

```
5 // define your exception class
6 class RangeError{
7 private:
8     int iVal;
9 public:
10     RangeError(int _iVal) {iVal = _iVal;}
11     int getVal() {return iVal;}
12 };
13
14 char to_char(int n)
15 {
16     if(n < numeric_limits<char>::min() || n > numeric_limits<char>::max())
17         throw RangeError(n);
18     return (char)n;
19 }
20
21 void gain(int n)
22 {
23     try{
24         char c = to_char(n);
25         cout << n << " is character " << c << endl;
26     }catch(RangeError &re){
27         cerr << "Cannot convert " << re.getVal() << " to char\n" << endl;
28         cerr << "Range is " << (int)numeric_limits<char>::min();
29         cerr << " to " << (int)numeric_limits<char>::max() << endl;
30     }
31 }
32
33 int main()
34 {
35     gain(-130);
36 }
37
38 return 0;
```

Define your exception class

Throw the exception and invoke the constructor

Catch and handle the exception

IO

C

```
scanf("%xx",&a)
getchar(used to omit \n, " ") gets(&a)

printf
puts(&a) [lab6 e1]
```

Cpp

```
cin.get (remember the omitted)
.good[lec5 p39][lab6 e3]
[lab3]

cout.setf .width .fill .precision
[lab2 e1]
```

Branch

```
If else
while
Goto
Switch
More similar with goto, not if-else if-else
switch.cpp
switch (input_char)
{
    case 'a':
    case 'A':
        cout << "Move left." << endl;
        break;
    case 'd':
    case 'D':
        cout << "Move right." << endl;
        break;
    default:
        cout << "Undefined key." << endl;
        break;
}
```

Arithmetic

95 // decimal	3.14159 // 3.14159
0137// octal	6.02e23 // 6.02 x 10^23
0x5F // hexadecimal	1.6e-19 // 1.6 x 10^-19
	3.0 // 3.0
95 // int	
95u // unsigned int	6.02e23L // long double
95l // long	6.02e23f // float
95ul // unsigned long	6.02e23 // double
95lu // unsigned long	

a<<b a 左移 b 位

int, long, float, double: four kinds of operations

If the operands are not the four types, automatic convert their types

unsigned char a = 255; unsigned char d = a + b // d = 0

unsigned char b = 1;

int c = a + b; // c = 256

Function

```
14 | #define MAX_MACRO(a, b) (a)>(b) ? (a) : (b)
15
16 int main()
17 {
18     int num1 = 20;
19     int num2 = 30;
20     int maxv = max_function(num1, num2);
21     cout << maxv << endl;
22
23     maxv = MAX_MACRO(num1, num2);
24     cout << maxv << endl;
25
26     maxv = MAX_MACRO(num1++, num2++);
27     cout << maxv << endl;
28     cout << "num1" << num1 << endl;
29     cout << "num2" << num2 << endl;
30
31
32 num1=21
num2=32
```

Inline/macro[simply a replacement] macro must use ()!

Defined in class:auto inline

pointer[lec 6]

an augument pass to another function(e.g. qsort)

default arguments

overloading

template&specialization&Instantiations[lab7]

extern/static

```
* "Function templates" vs "template functions".
template<typename T>
T max(T x, T y)
{
    cout << "The input type is " << typeid(T).name() << endl;
    return x > y ? x : y;
}

// instantiates sum=double(double, double);
// instantiates sum=char(char, char);
// instantiates sum=short(short, short);
// instantiates sum=int(int, int);
// instantiates sum=long(long, long);
// instantiates sum=long long(long long, long long);
// instantiates sum=long double(long double, long double);
// Implicitly instantiates product<int>(int, int)
// Implicitly instantiates product<float>(float, float)
// Implicitly instantiates product<double>(double, double)
// Implicitly instantiates product<long double>(long double, long double)
cout << "product = " << product<int>(2, 3) << endl;
cout << "product = " << product<float>(2.2f, 3.0f) << endl;
cout << "product = " << product<double>(2.2, 3.0) << endl;
cout << "product = " << product<long double>(2.2L, 3.0L) << endl;
```


• Java: 1995

- I hate memory management in C/C++!
- I want "Write once, run anywhere", not "write once, compile anywhere".
- Grammar is similar with C++.
- A Java compiler generates *.class files, not executable files.



• Python: 1990

- I hate strict grammar!
- I hate too many data types!



Python是脚本语言 脚本语言跟前面的Java跟前面的c、c++以及其他编译语言有个很大的不同就是



P62 输出流管道重定向

P63 assert macro NONDEBUG

推荐大家用int8_t, uint8_t, int32_t, uint32_t

<https://en.cppreference.com/w/cpp/language/typedef>

Type specifier	Equivalent type	Width in bits by data model				
		C++ standard	LP32	ILP32	LLP64	LP64
short	short int	at least 16	16	16	16	16
short int						
signed short						
signed short int						
unsigned short						
unsigned short int	unsigned short int					
int	int	at least 16	16	32	32	32
signed						
signed int						
unsigned						
unsigned int	unsigned int					
long	long int	at least 32	32	32	32	64
long int						
signed long						
signed long int						
unsigned long						
unsigned long int	unsigned long int					
long long	long long int (C++11)	at least 64	64	64	64	64
long long int						
signed long long						
signed long long int						
unsigned long long						
unsigned long long int	unsigned long long int (C++11)					

你可以缩写为short, 把int删掉 也可以就是这是有厂

你可以缩写为short把int删掉 也可以就是这是有几种变种的写法 就不同的变种



- Width in bits of different data models
- sizeof operator can return the width in bytes.