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
<cstdint>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

```
using Jill::fetch;    // a using declaration
```

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

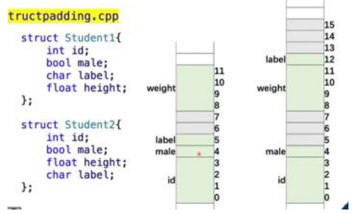
Compound

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 no need "struct Student s"=>Student s √

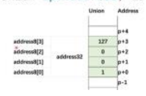
- In order to align the data in memory, some empty



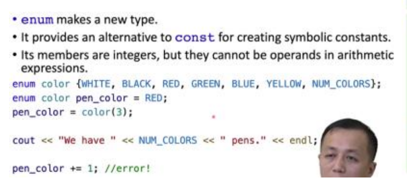
```
tructpadding.cpp
struct Student1{
    int id;
    bool male;
    char label;
    float height;
};

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.
```

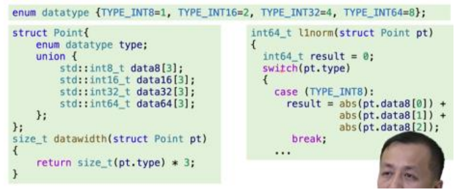


enumeration[lec5p38][lec10p23]



- enum makes a new type.
- It provides an alternative to const for creating symbolic constants.
- Its members are integers, but they cannot be operands in arithmetic expressions.

```
enum color {WHITE, BLACK, RED, GREEN, BLUE, YELLOW, NUM_COLORS};
enum color pen_color = RED;
pen_color = color(3);

cout << "We have " << NUM_COLORS << " pens." << endl;

pen_color += 1; //error!
```



```
enum datatype {TYPE_INT8=1, TYPE_INT16=2, TYPE_INT32=4, TYPE_INT64=8};

struct Point{
    enum datatype type;
    union {
        std::int8_t data8[3];
        std::int16_t data16[3];
        std::int32_t data32[3];
        std::int64_t data64[3];
    };
};
size_t datawidth(struct Point pt)
{
    return size_t(pt.type) * 3;
}
```

```
int64_t l1norm(struct Point pt)
{
  int64_t result = 0;
  switch(pt.type)
  {
    case (TYPE_INT8):
      result = abs(pt.data8[0]) +
               abs(pt.data8[1]) +
               abs(pt.data8[2]);
       break;
    ...
```

Dynamic memory

```
Student xue = Student("XueQikun", 1962, true);
xue.printInfo();

Student * zhou =  new Student("Zhou", 1991, false);
zhou->printInfo();
delete zhou;


Student * class1 = new Student[3]{
    {"Tom", 2000, true},
    {"Bob", 2001, true},
    {"Amy", 2002, false},
};

class1[1].printInfo();
//delete class1; 仅调1个 destructor
delete []class1;
```

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

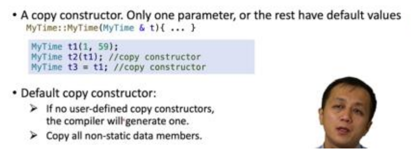
Abstraction component: public interface
Encapsulation component: gather the implementation details

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++× 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

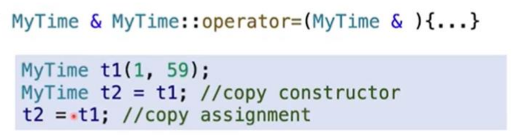


- A copy constructor. Only one parameter, or the rest have default values

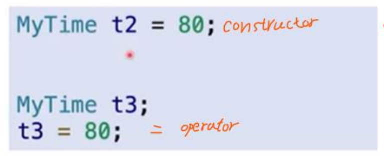
```
MyTime::MyTime(MyTime & t){ ... }
```

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

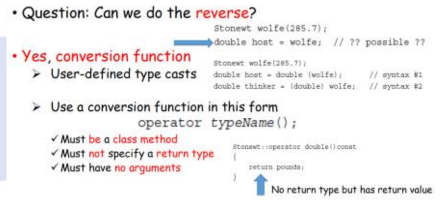
- Default copy constructor:
  - If no user-defined copy constructors, the compiler will generate one.
  - Copy all non-static data members.



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

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



```
MyTime t2 = 80;

MyTime t3;
t3 = 80;
```



- Question: Can we do the reverse?

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

- Yes, conversion function
  - User-defined type casts

```
Stonewt wolfe(285.7);
double host = double (wolfe);      // syntax #1
double thinker = (double) wolfe;   // syntax #2
```

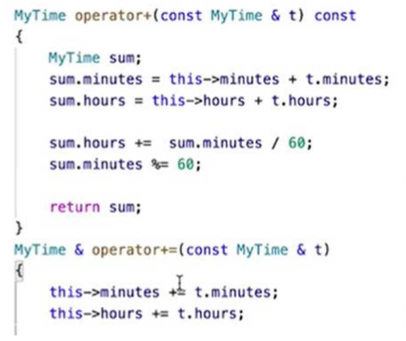
  - Use a conversion function in this form
    `operator typeName();`
    - ✓ Must be a class method
    - ✓ Must not specify a return type
    - ✓ Must have no arguments

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

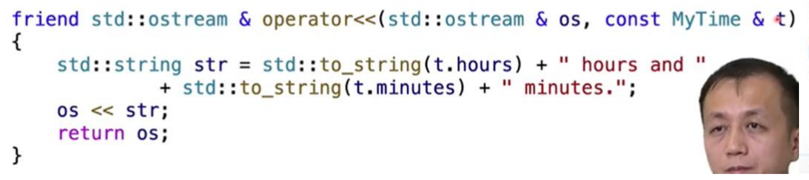
            Type cast: Convertion constructor[lec11 p25] | Conversion function: explicit operater [typename]()[p26]

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

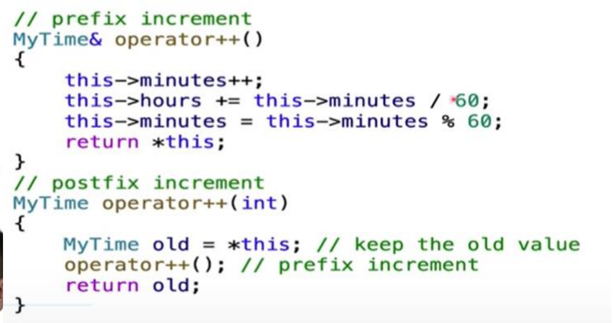
Operator overloading
            Use only member functions to overload =,(),[],-> | cannot overload . .* ?::
            Member/nonmenber: Time operator+(const Time &t)const||friend Time operator+(const Time &t1,t2)[achieve int+Time]



```
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)

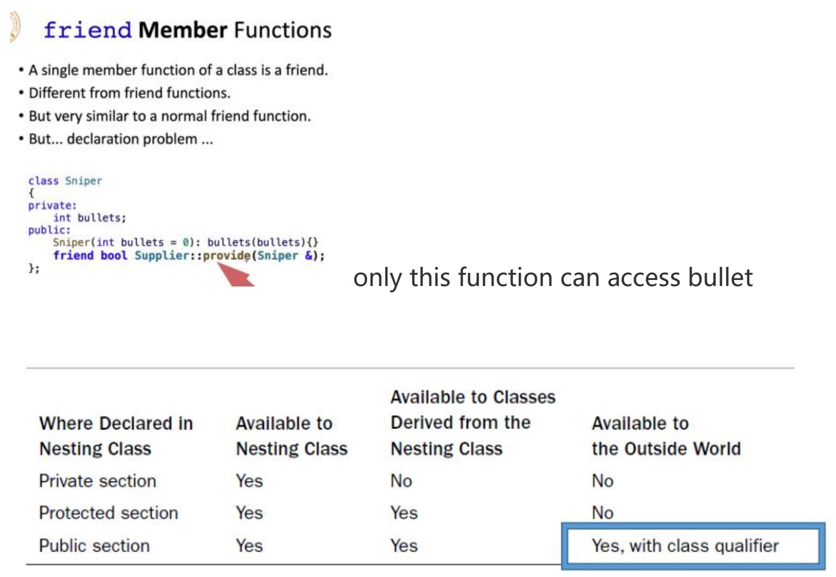
• automatically invokes the base-class default constructor if don't specify in list

• 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

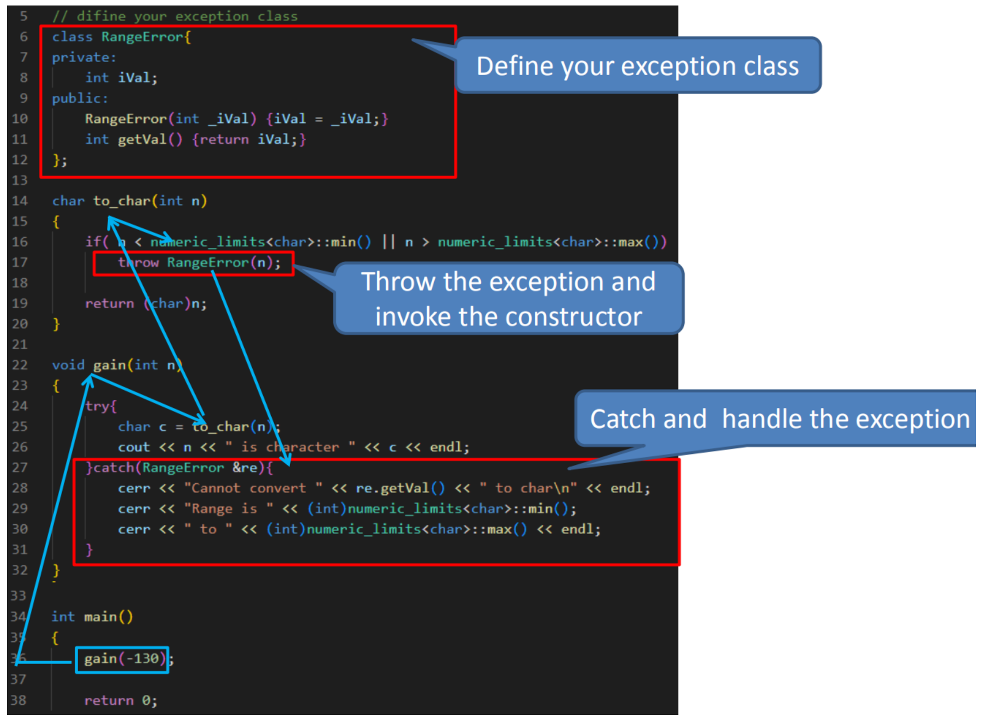
- A single member function of a class is a friend.
- Different from friend functions.
- But very similar to a normal friend function.
- But... declaration problem ...

```
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



```
// difine your exception class
class RangeError{
private:
    int iVal;
public:
    RangeError(int _iVal) {iVal = _iVal;}
    int getVal() {return iVal;}
};

char to_char(int n)
{
    if( n < numeric_limits<char>::min() || n > numeric_limits<char>::max())
        throw RangeError(n);

    return (char)n;
}

void gain(int n)
{
    try{
        char c = to_char(n);
        cout << n << " is character " << c << endl;
    }catch(RangeError &re){
        cerr << "Cannot convert " << re.getVal() << " to char\n" << endl;
        cerr << "Range is " << (int)numeric_limits<char>::min();
        cerr << " to " << (int)numeric_limits<char>::max() << endl;
    }
}

int main()
{
    gain(-130);

    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]

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
```

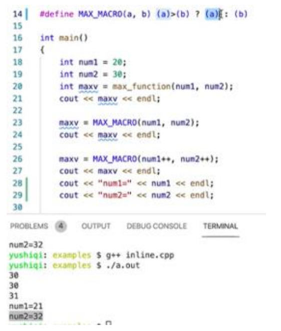
Branch

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;
}
```

Function



```
#define MAX_MACRO(a, b) (a)>(b) ? (a) : (b)

int main()
{
    int num1 = 20;
    int num2 = 30;
    int maxv = max_function(num1, num2);
    cout << maxv << endl;

    maxv = MAX_MACRO(num1, num2);
    cout << maxv << endl;

    maxv = MAX_MACRO(num1++, num2++);
    cout << maxv << endl;
    cout << "num1=" << num1 << endl;
    cout << "num2=" << num2 << endl;
```

```
num2=32
yushiqi: examples $ g++ inline.cpp
yushiqi: examples $ ./a.out
30
30
31
num1=21
num2=32
```

Inline/macro[simply a replacement] macro must use ()!
Defined in class:auto inline

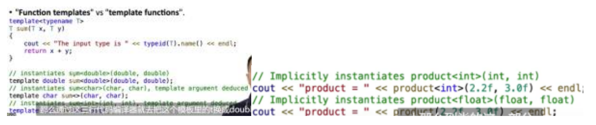
pointer[lec 6]

```
float (*norm_ptr)(float x, float y);
norm_ptr = norm_l1; //Pointer norm_ptr is pointing to norm_l1
norm_ptr = &norm_l2; //Pointer norm_ptr is pointing to norm_l1
```

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

default arguments
overloading

template&specialization&Instantiations[lab7]

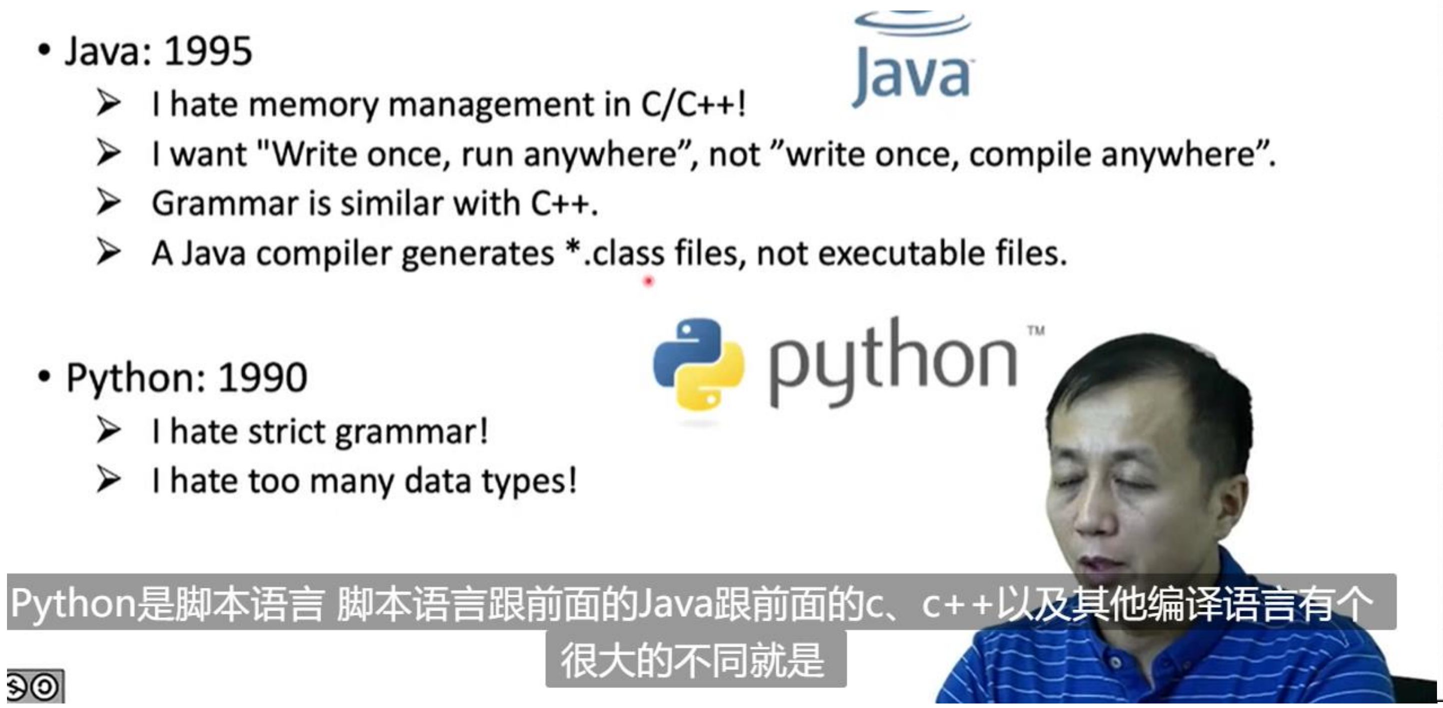


• "Function templates" vs "template functions".

```
template<typename T>
T sum(T x, T y)
{
    cout << "The input type is " << typeid(T).name() << endl;
    return x + y;
}

// instantiates sum<double>(double, double)
template double sum<double>(double, double);
// instantiates sum<char>(char, char), template argument deduced
template char sum<>(char, char);
// instantiates sum<int>(int, int), template argument deduced
template
```

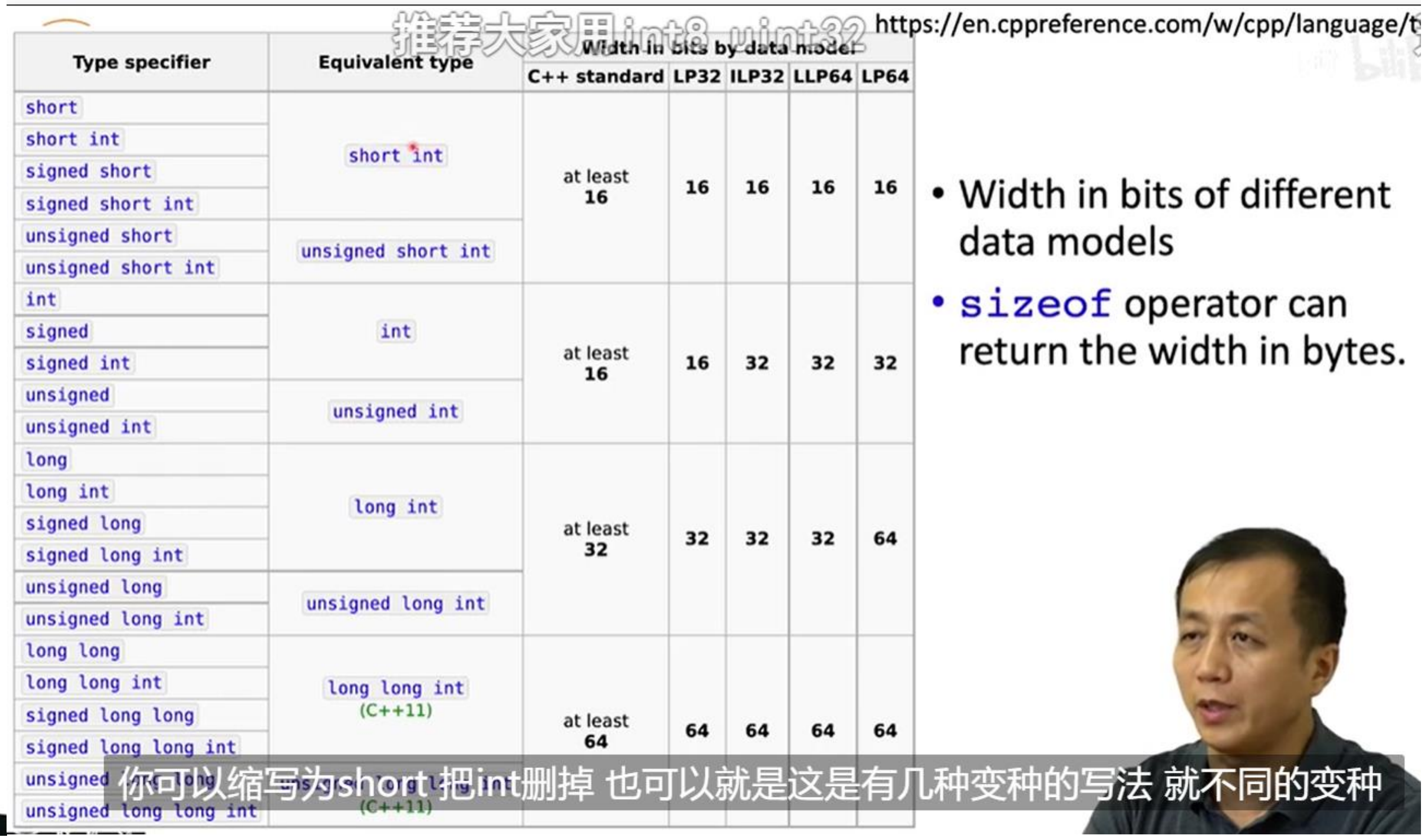
```
// Implicitly instantiates product<int>(int, int)
cout << "product = " << product<int>(2.2f, 3.0f) << endl;
// Implicitly instantiates product<float>(float, float)
cout << "product = " << product(2.2f, 3.0f) << endl;
```

extern/static



- 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



推荐大家用int8, uint32

https://en.cppreference.com/w/cpp/language/t

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 (C++11)					
unsigned long long int						

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

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

P62 输出流管道重定向

P63 assert macro NONDEBUG