# Introduction

## First program

|  |
| --- |
| //preprocessor directive  #include <iostream>  //namespace should be declared outside of functions  using namespace std;  int main(void){  cout << "hello\n" << endl;  } |

The header file "iostream" is file name

"cout" is one instance of the class iostream

A namespace is an abstract container or environment created to hold a logical grouping of unique entitles(blocks). "std" will make the standard facilities available throughout the program.

<< is defined in std::ostream::operator known as insertion operator. That overloaded as a member function.

std::cout<< "hi"

std::cin>>

## iostream.h

std::cout

|  |  |
| --- | --- |
| std::cout << 5 | writes integer 5 to standard output  if declare namespace std, std:: could be ignored.  cout<<5 |
| int bytes=255  cout<<bytes | decimal |
| cout << "hexadecimal=" << hex << bytes; | std::hex |
| cout<< oct<<bytes; | std:oct |
| cout<<hex<<bytes | std::hex |
| #include <iostream> // std::cout, std::endl  #include <iomanip> // std::setbase  int main () {  std::cout << std::setbase(16);  std::cout << 110 << std::endl;  return 0;  } | default is decimal  std::cout << std::setbase(16); force hexadecimal |
| char Char = 'X', Minus = '-';  float Float = 2.5;  cout << Char << Minus << Float; | output: X-2.5 |
| char Char = 'X';  int Int = Char;  cout << Char << " " << (int)Char << " " << Int << " " << (char)Int; | output: X 88 88 X |
| int i=3,j=++i,k=++i;  cin>>i;  cout<<k+i<<j-i<<i\*i; | output: 10-125 |
| float x=2.5;  cout << scientific << x; | output: 2.500000e+000 |
| float x=2.5;  cout << scientific << "x"; | output: x  Note compare it with the above |
| int i=8;  **cout << i << hex << i+i << oct << i;** | output: 81010 |
| double b = 1e15;  double s = 1e-15;  cout << fixed << b+s; | output: 1000000000000000.000000  std.fixed fix float point notation |
| double b = 1e15;  double s = 1e-15;  cout << scientific << b+s; | output: 1.000000e+015 |
| #include <iostream>  #include <iomanip> //setprecission  using namespace std;  int main(void){  double b = 1e15;  double s = 1e-15;  cout << fixed << setprecision(2) << b+s;  return 0;  } | output: 1000000000000000.00 |

std::cin

|  |  |
| --- | --- |
| int i;  std::cin >> i; | reads integer from the standard input |
|  |  |
|  |  |
|  |  |
|  |  |

## cmath.h

|  |  |
| --- | --- |
|  | sqrtf() square root |
|  |  |
|  |  |
|  |  |
|  |  |

## bool type

|  |  |
| --- | --- |
| bool yes = !false;  bool no = !yes;  if(!no) cout << "true";  else cout << "false"; | output: true |
| bool yes = !false;  bool no = !yes;  if(!no) cout << true;  else cout << false; | output: 1 |
|  |  |
|  |  |

## namespace

define namespace:

namespace <space name>{ …}

use namespace:

using namespace <space name>;

<space name>::<name>

use one object in namespace:

using <space name>::<name>

|  |  |
| --- | --- |
| #include <iostream>  use namespace std; | cout, cin, and edl belong to std |
| namespace Universe{int Galaxy=1;}  namespace Universe{int Planet = Galaxy+2; }  int main(void){  Universe::Galaxy \*=2;  //scope of namespace  **{**  **using namespace Universe;**  **Planet++;**  **}**  cout << Universe::Galaxy << Universe::Planet;  return 0;  } | output: 24  it is ok to define multiple block for one namespace.  There is scope limits for namespace {} |
| namespace space { char a= 'a', b= 'b';}  int a=1, b=2;  int main(void) {  **using space::a;**  **cout << a << " " << b;**  return 0;  } | output: a b  specify namespace  **Note: using namespace space::a; is invalid** |
| namespace space { char a= 'a', b= 'b';}  int a=1, b=2;  int main(void) {  using space;  cout << space::a << " " << space::b;  return 0;  } | compiling error: expected nested-name-specifier before 'space'  correct: using namespace space; |
| namespace outspace { int x=1; int y=2;}  namespace inspace { float x=3.0; float y=4.0;}  int main(void) {  { using namespace inspace;  cout << x; }  { using namespace outspace;    using inspace::y; cout << y; }  return 0;  } | output: 34  Note: "using namespace outspace;" is useless,"using inspace::y" override outspace::y. |

syntax error

|  |  |
| --- | --- |
| namespace A{ int A;}  namespace B{int A;}  int main(void){  A::A=B::A=1;  cout << A+1;  return 0;  } | compiling error.  correct: cout << B::A+1; |
| namespace space { char a= 'a', b= 'b';}  int a=1, b=2;  int main(void) {  using namespace space;  **cout << a << b;**  return 0;  } | compiling error:  ambiguous a and b  correct: cout << space::a << space::b; |

## NULL pointer

In C, NULL pointer is defined by ==NULL. But in default, "#define NULL 0;" which may cause confusion. In C++11, the pointer literal nullptr is introduced.

In C++, for null pointer ptr, ptr==NULL, ==0, !ptr, or ptr=nullptr will return 1

|  |  |
| --- | --- |
| void \*ptr = nullptr;  if(!ptr){  cout << NULL;  } | void \*ptr = 0;  void \*ptr=NULL;  are equal |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## sizeof()

The sizeof keyword gives the amount of storage, in bytes, associated with a variable or a type (including aggregate types). This keyword returns a value of type size\_t.

sizeof(string)=24

sizeof(char)=1

sizeof(short)=2, sizeof(int)=4, sizeof(long)=4, sizeof(long long)=8

sizeof(float)=4, sizeof(double)=8, sizeof(long double)=12,

string type

In sizeof(string), string is std::string. sizeof(std::string) returns the size of the class instance and its data members =24 bytes depending on platform, not the length of the string.

|  |  |  |
| --- | --- | --- |
|  | output |  |
| string a="abc";  cout<< sizeof(string); | 24 | 4\*3+4=  string class instance contains a pointer, and a struct |
| cout << (sizeof string); | 24 | the same as the above |
| string a="abc";  cout<< sizeof(a); | 24 | sizeof(a) is equal to sizeof(string) or sizeof(std::string)  string class instance contains a pointer, and a struct |
| string a="a\0bc\0";  cout<< sizeof(a); | 24 |  |
| cout<< sizeof("a\0bc"); | 5 | 4+1 characters |
| cout<< sizeof("a\0bc\0"); | 6 | 5+1 characters |
| string a="abc", \*p;  cout<< sizeof(p); | 4 | pointer size = 4 |
| string a="abc", \*p;  cout<< sizeof(\*p); | 24 | ==sizeof(a) |
|  |  |  |

size of int, float, double

|  |  |  |
| --- | --- | --- |
|  | output |  |
| int a;  cout<< sizeof(a=10); | 4 | expression a=10 succeeds return 1, which memory storage is 4 bytes |
| int a=100;  cout<< sizeof(a); | 4 | In C++, a is instance of int class. the pointer size is 4 bytes |
| double a=100.23;  cout<< sizeof(a); | 8 |  |
| int a=1;  cout<< sizeof(a=3) << a; | 14 | sizeof(a=3)=1, a=4 |
|  |  |  |
|  |  |  |
|  |  |  |

size of function

|  |  |  |
| --- | --- | --- |
|  | output |  |
| int f(){ return 100; }  int main(void){  cout<< sizeof(f()) << endl;  return 0;  } | 4 | sizeof(f()) == sizeof (int) |
| double f(){ return 10.23; }  int main(void){  cout<< sizeof(f());  return 0;  } | 8 |  |
|  |  |  |
|  |  |  |
|  |  |  |
| void f(){ int i=100; }  int main(void){  cout<< sizeof(f()) << endl;  return 0;  } | 1 | sizeof(void)=1 |
| void f(int &p){ p+=100; }  int main(void){  int i=100;  **cout<< sizeof(f(i)) << i**;  return 0;  } | 1100 | in sizeof(), f() can't be called because f(i) will be replaced with int type when compiling. In this snippet, the function is not being called. |

pointer

|  |  |  |
| --- | --- | --- |
|  | output |  |
| int i=4, \*p; p=&i;  cout<< sizeof(p)<<sizeof(\*p); | 44 |  |
| double i=4, \*p; p=&i;  cout<< sizeof(p)<<sizeof(\*p); | 48 |  |
| float i=4, \*p; p=&i;  cout<< sizeof(p)<<sizeof(\*p); | 44 |  |
| char i='a', \*p; p=&i;  cout<< sizeof(p)<<sizeof(\*p); | 41 |  |
| string i="abc", \*p; p=&i;  cout<< sizeof(p)<<sizeof(\*p); | 424 |  |
| float i[]={1,2,3}, \*p; p=i;  cout<< sizeof(p)<<sizeof(\*p); | 44 | \*p is equal to value of i[0], then equal to sizeof(float) |
| char i[]="ABC", \*p; p=i;  cout<< sizeof(p)<<sizeof(\*p); | 41 | sizeof(\*p) -> sizeof(i[0]) -> sizeof(char) |

## overloading operators

<string> + is used for concatenate multiple strings.

<money>

## sequence points

The order of execution is unspecified.

|  |  |
| --- | --- |
| class A{  public:  A(){  a[0]=1;  a[1]=0;  }  int a[2];  int b(void){  int x=a[0];  a[0]=a[1];  a[1]=x;  return x;  }  };  int main(){  A a;  a.b();  **cout << a.b() << a.a[1] << endl;**  return 0;  } | output: 01  Note: for cout statement, a.b() is not executed before a.a[1] |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# OOP

## class and object

object is instances of class

### constructor and instantiation

constructor a special non-static member function of a class. constructor is used for initializing objects of a class.

1.constructors must have the same name with class name and cann't be called directly like other class methods.

2. constructors without arguments are called default constructors. No constructor is ok for a class.

3. A constructor will be automatic run once when an object is created. If no constructor is define, a default constructor will be created for the object.

4. constructors that take another object of the same type as the arguments are copy constructor. In that case, constructors don't need run again.

1. class should be instantiated to objects before use except static methods. objects could be updated by obj=<value> (only one attr) or obj.<attr>=<value> (multiple attr).

class A{

int a;

public:

A(){ a=0; }

A(int i){ a=i; }

void get(){ cout << this->a;}

};

Here are all possible instantiations

|  |  |
| --- | --- |
| **A a;**  **a.get();** | output: 0  no arguments |
| A a(4);  a.get(); | output: 4  arguments constructor overload |
| A a=3;  a.get(); | output: 3;  equal to a(3) |
| A a;  a=6;  a.get(); | output: 6  update constructor |
| A a=4, b;  b=a; b.get(); | output:4  assign constructor |
| A \*p = new A();  \*p=6;  p->get(); | output: 6  pointer pattern |
| A a();  a.get(); | wrong |
|  |  |

2. public and private: attributes usually are private, which can't be accessed by objects. methods are public. If not explicit, all are private. **keep eyes on declaration of public and private.**

3. attributes are usually defined in private, and initialized in constructor. If no initialization, unknown value are returned but no compiling error. private attributes or methods can' be access by objects. **Keep eyes on the use like "obj.var=0;" in main(). can't use "cls::var" outside of class. Inside class, it is ok to use "var=0;", "this->var=0;", "cls::var=0;"**

**use class or objects as attribute**

|  |  |
| --- | --- |
| class A{  public:  string s;  A(string s){ this->s = s;}  };  class B{  public:  string s;  **B(A a){ this->s = a.s;}**  void print(){cout<<s;}  };  int main(void) {  A a("hello");  **B b=a; //equal to "B b(a);"**  b.print();  return 0;  } | **output: hello** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

4. constructor has not return, and can't be as other methods in class. overloading constructor: void, values, references of objects

5. destructor would better be defined after constructor. All destructors matching all objects would be executed when returning in main().

|  |  |
| --- | --- |
| class A{  public:  A(){ a[0]=1; a[1]=0; }  int a[2];  int b(void){  int x=a[0];a[0]=a[1];a[1]=x;  return x;  }  };  int main(){  A a; a.b();  **cout << a.b() << a.a[1] << endl;**  return 0;  } | output:01  **Note: execution order doesn't follow the order in cout statements** |
| class A{  public:  A(){ a.a=a.b=1;}  struct {int a,b;} a;  int b(void);  };  int A::b(void){  int x=a.a; a.a=a.b; a.b=x;  return x;  };  int main(){  A a; a.a.a = 0; a.b();  cout << a.b() << a.a.b << endl;  return 0;  } | output: 10  Note: a.a.b is executed before a.b() in cout statement. |
| class A{  public:  int a;  A() { a=0; cout << "A" << a; }  A(int b) {cout << "B" << b;}  };  int main(){  **A a(0);**  **a=4;**  return 0;  } | output: B0B4  **Update instance: instantiate class A with a(0), and then update instantiation.** |
| #include <iostream>  using namespace std;  class Class {  public:  Class(void) { cout << "constructed-1" << endl; }  Class(int v) { value = v; cout << "constructed-2" << endl; }  ~Class(void) { cout << "destructed val = " << value << endl; }  void IncAndPrint(void) {  cout << "value = " << ++value << endl;  }  int value;  };  int main(void) {  Class \*ptr1, \*ptr2;    ptr1 = new Class;  ptr2 = new Class(2);  ptr1 -> value = 1;  ptr1 -> IncAndPrint();  ptr2 -> IncAndPrint();  delete ptr2;  delete ptr1;  return 0;  } |  |
| class A{  public:  int data[3];  private:  int cnt;  public:  void put(int v){data[cnt++]=v;}  int take() {**int c=cnt;cnt=0;**return c;}  };  int main(){  A a;  a.take();  a.put(a.take());  a.put(1);  cout << a.data[0] << endl;  return 0;  } | output:0  attribute initialization: Run a.take() first time: "int c=cnt;" c is not initialized because cnt is not initialized. |
| class A{  float v;  public:  A(){v=1.0;}  **A(A &a) {A::v=a.v; cout<<"1";}**  ~A(){cout<<"0";}  float set(float v){  A::v=v;return v;}  float get(float v){return A::v;}  };  int main(){  A a, \*b=new A(a),  \*c=new A(\*b);  c->get(b->get(a.set(1.0)));  delete b;  delete c;  return 0;  } | output: 11000  note: 1. overloading: pass reference of object  2. three instances: run constructor three times, and run destructor three times when return in main(). |
| class A{  public:  A(float v){A::v=v;}  float v, b;  float set(float v){A::v=v; return v;}  float get(float v){return A::v;}  };  int main(){  **A \*a=new A(1.0), \*b=new A(\*a);**  cout<< a->get(b->set(a->v));  return 0;  } | **output:1**  **Note: A \*b=new A(\*b); but constructor doesn't define pointer argument** |

### attributes

1. set default values of attributes in constructor

2. declare attributes is default private. It is ok to public or protected.

3.for private attributes, build methods to access or update private methods. and configure those methods as public, protected.

|  |  |
| --- | --- |
| class A{  int x,y,z;  public:  A(){x=y=z=0;}  **A(int a, int b): x(a), y(b){ z=x+y; }**  void print(){ cout << x << y << z; }  };  int main(void){  A a(1,2); a.print();  A b; b.print();  return 0;  } | output:123000  1. two constructors: one default gives default values. another pass arguments  2.**"A(int a, int b): x(a), y(b){ z=x+y; }" is equal to "**A(int a, int b) { x=a, y=b, z=x+y; }**".**  **"x(a),y(b)" is initializer list.** |
| class A{  **public: int x,y,z;**  void print(){ cout << x << y << z; }  };  int main(void){  A b; b.print();  return 0;  } | output: unpredictable values  though no compiling errors, attributes x,y,y are not initialized but allocated memory address. |
| class A{  **public: int x,y,z;**  void print(){ cout << x << y << z; }  };  int main(void){  **A b; b.x=b.y=b.z=0; b.print();**  return 0;  } | output:000  initialize/update attributes of objects. In real world, it not suggested. And attributes should be public type. |
| class A{  **private: int x,y,z;**  public:  A(){x=y=z=0;}  A(int a, int b) { x=a, y=b, z=x+y; }  **void setX(int a){ x=a; }**  **void getX(){ return x;}**  void print(){ cout << x << y << z; }  };  int main(void){  A b; b.setX(1); b.print();  return 0;  } | output:100  1.For security, attributes are usually configured as private, which can't be accessed by other classes. and build methods for updating or getting attributes. |
| class A{  private: int x,y,z;  public:  A(){x=y=z=0;}  A(int a, int b) { x=a, y=b, z=x+y; }  void print(){ cout << x << y << z; }  protected:  void setX(int a){x=a;}  void getX(){return x;}  };  int main(void){  A b; b.setX(1); b.print();  return 0;  } | compiling error.  main() can't access attributes because those are protected methods, which is only open to derived class. |
|  |  |
|  |  |
|  |  |

### change constructor

1. assign constructor differing from initialization

For class A,

copy constructor: "A a1; A a2=a1;" object a1 is initialized, and a2 is initialized by the value of as is passed by value of a1. The two share the same memory

assign constructor. "A a1, a2; a2=a2" objects a1 and a2 are initialized. both of them have values. Then a2's values are updated by a1.

|  |  |
| --- | --- |
| class Uno{  public: Uno() { cout << "x";}  };  Uno foo(Uno d){  **Uno e=d; return e;**  }  int main(void) {  Uno u; foo(u);  return 0;  } | output: x  Uno e=d; copy instructor.  the two objects are different |
| class Uno{  public: Uno() { cout << "x";}  };  Uno foo(Uno d){  **Uno e; e=d; return e;**  }  int main(void) {  Uno u; foo(u);  return 0;  } | output: xx  Uno e; e=d; e is different object |
|  |  |
|  |  |
|  |  |

2. copy constructor only if class type is same.

copy newly objects: "A a; A b=a;"

|  |  |
| --- | --- |
| class A{  int val;  public:  A() {val=1;}  int get(){return ++val;}  };  int main(){  // copy object  **A a, b=a;**  b.get();  cout << a.get() << b.get();  return 0;  } | output:23  "A a, b=a;" and "A a,b; b=a;" result in same result |
| class A{  int \*val;  public:  A() {val = new int; \*val=0;}  int get(){return ++(\*val);}  };  int main(){  A a, b=a;  cout << a.get() << b.get();  return 0;  } | output:21 |
| class A{  int \*val;  public:  **A() {val=new int; \*val=0;}**  **A(A &a){val=new int; \*val = a.get();}**  int get(){return ++(\*val);}  };  int main(){  A a, b=a;  cout << a.get() << b.get();  return 0;  } | output:22  Note: "b=a" equal "b(a)"  b, and a are two different objects. initialized using different constructor. |
| class N{  public: float x;  N(){ x=0.0; }  N(float a){ x=a; }  N(N &n){ x=n.x; }  string operator==(N &n){  if(this!= &n) return "true"; else return "false";  }  };  int main(void) {  N a(1.1), \*b=&a, c=a, d; d=a;  cout << (a==\*b) << (a==c) << (a==d);  return 0;  } | output: falsetruetrue  except object pointer \*b, the object c and d are copy of a, differing from a. |
|  |  |
|  |  |
|  |  |

pass objects as arguments or return statement into function: "class A{}; A a; void f(A a);"

2. convert constructor: child-> parent, parent

static\_cast<A \*>(b); child->parent. done when compiling

dynamic\_cast<B \*>(a); child->parent, or parent->child in some cases

|  |  |
| --- | --- |
| class X{};  class Y: public X{};  class Z: public X{};  int main(){  Z \*z=new Z(); X \*x=new X(); Y \*y=new Y();  **x=z;  cout<< (x==z) <<endl;**  // wrong: z=y; not inheritance  //wrong: y=z; no inheritance  //wrong: z=x; can't convert parent to child  return 0;  } | output: 1  correct: convert child object to parent.  can't convert parent class to child  can't convert if no inheritance |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

3. pass constructor by value or by reference

|  |  |
| --- | --- |
| class A{  int \*val;  public:  A() {val=new int; \*val=0;}  A(A &a){val=new int; \*val = a.get();}  int get(){return ++(\*val);}  };  int main(){  **A a, b=a;**  cout << a.get() << b.get();  return 0;  } | output: 22 |
|  |  |
| class A{  public: void out(){cout << "A" << endl;}  };  class B: public A{  public: void out(){cout << "B" << endl;}  };  int main(){  **A \*a = new B();**  a->out();  **static\_cast<B\*>(a)->out();**  return 0;  } | output: AB  object of child class could be converted to that of parent class or reverse back to child class.  Note: but child class can't be converted to parent class. |
| class A{  public:  A():val(0) {}  int val;  virtual void run(){cout<<val;}  };  class B: public A {};  class C: public B {  public:  void run(){cout<<val+2;}  };  void Do(A \*a){  B \*b;  C \*c;  if(b=dynamic\_cast<B\*>(a))  b->run();  if(c=dynamic\_cast<C\*>(a))  c->run();  a->run();  }  int main(){  A \*a=new C();  Do(a);  return 0;  } |  |
| #include <iostream>  using namespace std;  class Class {  int data;  public:  Class(int value) : data(value) {}  void increment(void) { data++; }  int value(void) { return data; }  };  int main(void) {  Class o1(123);  **Class o2 = o1;**  **Class o3(o2);**    o1.increment();  cout << o1.value() << endl;  cout << o2.value() << endl;  cout << o3.value() << endl;  return 0;  } | the object o2 and o3 are copy of object o1. but they are different object and each has its own data field.  A(A &) |
| class A{  public:  A(): val(0){}  int val;  void inc(){++val;}  };  **void Do(A a){a.inc();}**  int main(){  A a;  Do(a);  a.inc();  cout<< a.val;  return 0;  } | output:1  pass object to function. The object is local. inner changes would not affect on the object. |
| class A{  public:  A(): val(0){}  int val;  **int inc(){ ++val; return val--;}**  };  void Do(A \*a){   a->val = a->inc();}  int main(){  A a;  Do(&a);  cout<< a.val << a.inc();  return 0;  } | output:12 |
|  |  |
|  |  |
|  |  |

5. update attributes by constructor

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

### destructor

For class X:

if "X x;", destructor will be automatically run within the scope.

if "X \*x = new X();", destructor will not run unless "delete x;".

|  |  |
| --- | --- |
| class X{  public:  X() { cout << 1; };  ~X() { cout << 2; };  };  int main(void) {  X x;  cout << 0;  return 0;  } | output:102  **"X x;" so destructor will be run once main() is done.** |
| class X{  public:  X() { cout << 1; };  ~X() { cout << 2; };  };  void exec(){ X x; }  int main(void) {  X \*x = new X();cout << 0;  return 0;  } | **output: 10**  **create class pointer. destructor will not run if no delete statement** |
| class X{  public:  X() { cout << 1; };  ~X() { cout << 2; };  };  int main(void) {  **X \*x = new X();delete x;**cout << 0;  return 0;  } | output: 120  **compared with the above**  **for class pointer, "delete x;" will trigger destructor.**  **If remove "delete x;" behind of "cout<<0;", the output will be 102.** |
| class A{  public:  string s;  A(string s){ this->s = s; cout << "Con";}  **~A(){cout << "De";}**  };  int main(void) {  A a("hello");  **a.~A();**  return 0;  } | output: ConDeDe  destructor could be launched separately. but constructor can't run again. for example "a.A("good");" is wrong. |
| class X{  public:  X() { cout << 1; };  ~X() { cout << 2; };  };  **void exec(){ { X x;} }**  int main(void) {  exec();cout << 0;return 0;  } | output:120  Note: the scope of "X x;" is within exec() rather main(). so destructor is run before "cout<<0;" |
| class E { public: E(int i) { cout << i; } };  class X{  static int c;  public:  X() { if(c++>2) throw E(c);};  ~X() { if(c++>2) throw E(c);};  };  int X::c=0;  void f(int i) { X a, b;  cout << i; }  int main(void) {  try { f(0); f(1); }  catch(...){ cout << 1; }  return 0;  } | output: 041  0 is determined by f(0)  4 is determined by constructor of f(0)  1 is determined by cout<<1 |
| class X{  public:  X() { cout << 1; };  ~X() { cout << 2; };  };  X \*exec(){  X \*x = new X();  throw string("0");  return x;  }  int main(void) {  X \*x;  try { delete exec(); }  catch (string &s){ cout << s;}  return 0;  } | output: 10  Note: 1. "X\* x;" has not impact on outcome.  2. delete have not impact on outcome because exception is thrown before return statement. |
| class A{  string a;  public:  A(){a="a"; cout << "A";}  A(string a){cout << a;}  ~A(){cout << "a";}  };  void f(A\* d){  **A e("3"); \*d = e;**  }  int main(void) {  **A \*a = new A(); f(a); delete a;**  return 0;  } | ouput:A3aa  1. two patterns to instantiate class. for pointer pattern, delete should be run or no destructor.  2. \*d=e is equal to d=&e |
| class A{  public:  string s;  A(string s){ this->s = s; }  **~A(){ delete s;}**  };  int main(void) {  A a("hello");  return 0;  } | compiling error:  don't use "delete s;" in destructor because "~A(){}" will delete all attributes without deleting them one by one. |
| class X{  public:  X() { cout << 1; };  ~X() { cout << 2; };  };  void exec(){  **{ X x;}**  **throw string("0");**  }  int main(void) {  **try { exec();}**  **catch(string &s){ cout << s; }**  return 0;  } | output:120  {X x;} constructor and destructor will run in the scope once throw exception |

### static

A static variable is never allocated on a stack. They are allocated space on different static storage. when we declare a static variable in a class, this variable is shared by all the objects of that class.

As static variables are initialized only once and are shared by all objects of a class, **the static variables are never initialized by a constructor.** Instead, the static variable should be explicitly initialized outside the class only once using the scope resolution operator (::).

When the first object is created, all static data of primitive type is initialized to zero when no other initialization is present.

|  |  |
| --- | --- |
| class Test {  public:  static void funS1(void) { cout << "static" << endl; }  static void funS2(void) { funS1(); }  };  int main(void) {  Test object; Test::funS2();object.funS2();  return 0;  } | output:  static  static  class->static methods or  instance->static methods |
| class Test {  public:  void funN1(void) { cout << "non-static" << endl; }  static void funS1(void) { funN1(); }  };  int main(void) {  Test object;Test::funS1();object.funS1();  return 0;  } | compiling error  static method can't access not-static methods |
| class Test {  public:  static void funS1(void) { cout << "static" << endl; }  void funN1(void) { funS1(); }  };  int main(void) {  Test object;object.funN1();  } | non-static can access static method |
|  |  |
|  |  |

static attributes

1. static attribute should be declared in class. That can't be initialized in constructor, but out of the class.

2. can't be call static attribute by class cls::<var> or by object obj.<var>

|  |  |
| --- | --- |
| class A{  **public:static int a;**  A() {a++;}  A(A &aa) {a++;}  };  **int A::a=1; // initialization out of class**  int main(){  A a, b(a), c(b);  **cout << A::a << endl;**  **cout << a.a << endl;**  return 0;  } | output: 4 4  define static variable |
| class A{  public: static int age;  A(){}  ~A(){}  void setAge(int a=10){age=a;}  void print(){ cout<<age;}  };  int A::age = 0;  int main(void) {  A a, c, \*b;   b = new A();  **a.setAge(30);b->setAge(20);  c.setAge();**  a.print();b->print();c.print();  return 0;  } | output:101010  though define multiple objects, A::age is always determined by last setting. |
| class E {};  class X{  **public: static int c;**  X(int a) { c=a; };  ~X() { if(c++>2) throw new E; }  };  **int X::c=0;**  void f(int i){  X\*t[2];  for(int j=0;j<i;j++)  t[j]=new X(i+1);  for(int j=0;j<i;j++)  delete t[j];  }  int main(void) {  try { f(2); }  catch(...) { cout << X::c;}  return 0;  } | output:4 |

### const class variable

**A**const**class field must be initialized inside an initialization list within any of the class constructors.** Any other assignment will be rejected.

a const **object mustn’t be modified during its life**.

|  |  |
| --- | --- |
|  |  |
| const a =3.14;  const b= a\*a;  cout << b; | compiling error  correct: const float a=3.14; |
|  |  |
|  |  |
|  |  |

### object pointer

object has class type. object type casting is not allowed except inheritance.

type casting of object should be explicitly done by object pointer.

|  |  |
| --- | --- |
| class A{public: void print(){cout<<"A";}};  class B{public: void print(){cout<<"B";}};  int main(void) {  B sc[2]; sc[0].print();  **A \*bc = (A\*) sc;**  for(int i=0; i<2;i++)  (bc++)->print();  return 0;  } | output:BAA  Note: class A and B have not relation.  **wrong: A\*bc = sc;** |
| class A{  public: void print(){cout<<"A";}  };  class B: public A{  public: void print(){cout<<"B";}  };  **void fun(A \*obj){ obj->print(); }**  int main(void) {  A a; fun(&a);  B b; fun(&b);  return 0;  } | output:AA  The argument of fun() only accept object of base class A. convert B to A  wrong: **void fun(B \*obj){ obj->print(); } can't convert from based class 'A\*' to derived class 'B\*'** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## inheritance

Pattern:

class X{…};

class Y {…};

class Z {…};

class A : public X, Y, Z { … };

Note: except attributes and method, constructor, destructor and operator= cannot inherit from base class.

class inheritance: derived class inherit attributes and methods from base class.

|  |  |
| --- | --- |
| class X{  public:int v;  void put(int x){ v=x; }  int get(void){ return v; }  };  **class Y: public X{**  public:  Y(){ put(0); }  void write(int x){ put(x+1); }  int read(void){ return get()-1; }  };  int main(){  **Y \*y=new Y(); //derived class**  y->write(1);cout<<y->read()<<endl;  return 0;  } | output: 1  class Y inherit attribute v, methods put() and get() from class X  Note: this is simplest case. 1. There is no overloading on attributes/constructor/methods. 2. No upcast or downcast of object. 3. access is public. 4. one inheritance. no inheritance chain or other relationship. |

pass object to class is equal to inheritance

|  |  |
| --- | --- |
| class A{  public:int a;  A() {a =0;}  A(int b) {a=b+1;}  };  class B{  public:**A a;**  **B(): a(0){}**  };  int main(){  B \*b= new B();   cout << b->a.a << endl;  return 0;  } | output: 1  "B(): a(0){}" explicit define constructor in class B, which inherit from constructor of class A. |
| class A{  public:  int a;  A() {a =0; cout << "AA";}  A(int b) {a=b+1;cout << "A";}  };  class B{  public: A a;  B(): a(0){a=4; cout << "B";}  };  int main(){  B \*b= new B(); cout << b->a.a << endl;  return 0;  } | output: AAB5  constructor of class B inherit that from class A. a(0) is object of class A in constructor. Then the object a is updated with 4. |
| class A{  public:int a;  A() {a=1;}  A(int aa) {a=2;}  **A(A &aa) {a=3;}**  };  int main(){  A a(1), b(a);  cout << a.a + b.a << endl;  return 0;  } | output:5  Note: one overloading constructor is pass instance of itself. |

### inheriting constructors

Order of Constructor Call with Inheritance in C++. The order of the destructor is exactly the reverse

1. Construction always starts with the base class. If there are multiple base classes then, construction starts with the leftmost base. If there is a virtual inheritance then it's given higher preference).

2. Then the member fields are constructed. They are initialized in the order they are declared

3. Finally, the class itself is constructed

|  |  |
| --- | --- |
| class A{  **public:A(){cout<<"Ano";}**  };  class B: public A{  **public:B(int s){cout<<"Bint";}**  };  int main(void) {  B b1(10);  return 0;  } | output:AnoBint  construction order: start from base class: A() -> B(int s) |
| class A{  public:  A(){cout<<"Ano";}  **A(int s){cout<<"Aint";}**  };  class B: public A{  **public:B(int s){cout<<"Bint";}**  };  int main(void) {  B b1(10);  return 0;  } | output:AnoBint  construction order: A() -> B(int s)  **Note: constructor A(int s) is overridden by constructor B(int s)** |
| class A{  public:  **A(int s){cout<<"Aint";}**  };  class B: public A{  **public:B(int s){cout<<"Bint";}**  };  int main(void) {  B b1(10);  return 0;  } | compiling error  "A(){cout<<"Ano";}" is missing. calling constructor always start from base class. Or remove all constructor of class A. If constructors are required in class A, at least A() should be defined. |
| class A{  public:  A(){cout<<"Ano";}  A(string s){cout<<"Astr";}  A(A &a){cout<<"Aobj";}  };  class B: public A{  public:  B(){cout<<"Bno";}  B(string s){cout<<"Bstr";}  B(int s){cout<<"Bint";}  };  int main(void) {  B b1(10);B b2(b1);  return 0;  } | output: AnoBintAobj  order: A::A() -> B::(int s), A::(A &a) |
| class A{  public:  A(){cout<<"Ano";}  A(string s){cout<<"Astr";}  A(A &a){cout<<"Aobj";}  };  class B: public A{  public:  B(){cout<<"Bno";}  B(string s){cout<<"Bstr";}  B(int s){cout<<"Bint";}  };  int main(void) {  B b1(10);  B b2("good");  return 0;  } | output: AnoBintAnoBstr  order : A::A() -> B::(int s), A::A() ->B::(string s) |
| class A{  public:  A() {cout<<"A";}  ~A() {cout<<"a";}  };  class B: public A{  public:  B() {cout<<"B";}  ~B() {cout<<"b";}  };  class C: private B{  public:  C() {cout<<"C";}  ~C() {cout<<"c";}  };  int main(void) {  B b; C c;  return 0;  } | output: ABABCcbaba  constructor storage is stack type: "last in, and first out.":  for constructor: order starts from based to derived.  "B b;" constructor: AB  "C c" constructor: ABC  for destructor, order is reversed. first c and then b  "C c" destructor: cba  "B b;" destructor: ba |
| class A{  public:  A() {cout<<"A";}  ~A() {cout<<"a";}  };  class B{  public:  B() {cout<<"B";}  ~B() {cout<<"b";}  };  class C: public A, public B{  public:  C() {cout<<"C";}  ~C() {cout<<"c";}  };  int main(void) {  C c;  return 0;  } | output: ABCcba  inheritance: C->A, C->B |
| class A{  public: int x;  A(){x=0;}  };  class B{  public: int x;  B(){x=1;}  };  class C: public A, public B{  public: int x;  C(int x){ this->x=x; A::x=x+1; }  void print(){ cout<<x<<A::x<<B::x; }  };  int main(void) {  C c(1); c.print();  return 0;  } | output:121 |

Here are inheritance of three classes

Constructors start from base class in order and finally is derived class.

Destructors are in reversed order.

|  |  |
| --- | --- |
| class A{  public:A() {cout<<"A";}~A() {cout<<"a";}  };  **class B: public A{**  public: B() {cout<<"B";}~B() {cout<<"b";}  };  **class C: private B{**  public: C() {cout<<"C";}~C() {cout<<"c";}  };  int main(void) {  B b; C c;  return 0;  } | output: ABABCcbaba  **inheritance: C->B->A**  constructor storage is stack type: "last in, and first out.":  for constructor: order starts from based to derived.  "B b;" constructor: AB  "C c" constructor: ABC  for destructor, order is reversed. first c and then b  "C c" destructor: cba  "B b;" destructor: ba  Note: private A will not affect on running of constructors |
| class A{  public:A() {cout<<"A";}~A() {cout<<"a";}  };  class B{  public: B() {cout<<"B";}~B() {cout<<"b";}  };  **class C: public A, public B{**  public: C() {cout<<"C";}~C() {cout<<"c";}  };  int main(void) {  C c;  return 0;  } | output: ABCcba  **inheritance: C->A, C->B** |
| class A{  public:A() {cout<<"A";}~A() {cout<<"a";}  };  **class B: public A{**  public: B() {cout<<"B";}~B() {cout<<"b";}  };  **class C: protected A, protected B{**  public: C() {cout<<"C";}~C() {cout<<"c";}  };  int main(void) {  C c;  return 0;  } | output: AABCcbaa  inheritance: B->A, C->A,B  Note: "**class C: protected A, protected B " or** "**class C: protected B, protected A " will not raise compiling error, but should be " class C:  protected B "** |
| class A{  public: int x;  A(){x=0;}  };  class B{  public: int x;  B(){x=1;}  };  class C: public A, public B{  public: int x;  C(int x){ this->x=x; A::x=x+1; }  void print(){ cout<<x<<A::x<<B::x; }  };  int main(void) {  C c(1); c.print();  return 0;  } | output:121 |
| class B {  private:  B() { cout<<"B"; }~B() { cout<<"b"; }  };  **class C: public B{**  public: C() {cout<<"C";}~C() {cout<<"c";}  };  int main(void) {  C c;  return 0;  } | compiling error  can't set constructor as private in class B. |

### upcast and downcast

upcasting is to convert object or pointer of derived-class to base class. treat derived-class as base class.

parent\_obj = child\_obj.

downcast is to convert object or pointer of based class to derived class. downcasting is not allowed without an explicit type cast because "is-a" relationship.

child\_obj = parent\_obj is wrong.

class A{

public: void out(){cout<<"A";}

};

class B: public A{

public: void out(){cout<<"B";}

};

**class B inherit from class A. Access type is public.**

|  |  |  |
| --- | --- | --- |
| A a; a.out();  B b; b.out(); | AB | common approach.  for b, B::out() override A::out() |
| A \*pa = new A(); pa->out();  B \*pb = new B(); pb->out(); | AB | pointer approach |
| **A a = B(); a.out();** | **A** | upcast: it is ok to convert child to parent |
| **pa = pb; pa->out();**  **pa = new B(); pa->out();** | **AA** | upcast |
| **B b = A();** |  | wrong: can't convert parent to child |
| pb=pa; |  | wrong: can't convert parent to child |
| pb = (B\*) &a; pb->out();  pb = (B\*) pa; pb->out(); | BB | downcast explicitly |
|  |  |  |
|  |  |  |

|  |  |
| --- | --- |
| class A{  public: void out(void){ cout << "A"; }  };  class B: public A{  public: void out(){ cout << "B"; }  };  int main(){  A \*a = new B();  **a->out(); static\_cast<B \*>(a)->out();**  return 0;  } | output: AB  upcast  class B inherits from class A |
| class zero {public: void out(){cout<<0;}};  class one: public zero {public: void out(){cout<<1;} };  class two: public zero {public: void out(){cout<<2;} };  int main(void) {  one obj1; two obj2;  **zero \*ptr;**  ptr=&obj1; ptr->out();  ptr=&obj2; ptr->out();  **zero obj;**  obj=obj1; obj.out();  obj=obj2; obj.out();  return 0;  } | output: 0000  inheritance: zero<-one, zero<-two  pointer \*ptr or object obj is class zero. |
| class zero {public: void out(){cout<<0;}};  class one: public zero {public: void out(){cout<<1;} };  class two: public one {public: void out(){cout<<2;} };  int main(void) {  zero obj0, \*ptr0;  one obj1, \*ptr1;  two obj2, \*ptr2;  //wrong: ptr1=&obj0; ptr1->out();  ptr1=&obj2; ptr1->out();  //wrong: obj1=obj0; obj1.out();  obj1=obj2; obj1.out();  return 0;  } | output:11  inheritance: zero<-one<-two  wrong: obj1=obj0; obj1.out();  no known conversion for argument   1 from 'zero' to 'one&&' |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

### errors of inheritance

1. can't convert class in initialization if the two classes have not relation

2. child class access private members of parent class

3. duplicate loading: loaded parent classes have inheritance.

4. duplicate definition of class

|  |  |
| --- | --- |
| class A{  public: void out(void){ cout << "A"; }  };  class B: public A{  public: void out(){ cout << "B"; }  };  class C: public B,A{  }; | compiling error  super class B and A can't co exists because B inherit from A |
| class A{  public: void out(void){ cout << "A"; }  };  class B{  public: void out(){ cout << "B"; }  };  class C: public A,B{  }; | compiling error  super class A and B have the same name members which are ambiguous. |
| class B{  public:  void out(){ cout << "B"; }  void toB(){ cout << "b"; }  };  class C{  public:  void toC(){ cout << "c"; }  };  int main(){  B \*b = new C();  return 0;  } | compiling error  class B and C haven't not relationship. can't convert \*C to \*B in initialization. |
| class B{  public: void out(){ cout << "B"; }  };  class C: B{};  int main(){  C \*c = new C(); c->out();  return 0;  } | compiling error: B is not accessible based of C  correct: "class C: public B{};" |
| class B{  void out(){ cout << "B"; }  };  class C: public B{};  int main(){  C \*c = new C();  c->out();  return 0;  } | compiling error: inaccessible  B::out() is private. |
| class A{  int a=3;  public:  void out(void){ cout << "out"; }  };  class C: public A{};  int main(){  C \*c = new C();  cout << c->a;  return 0;  } | compiling error: inaccessible  A::a is private. |
| class A{  public: void out(void){ cout << "out"; }  };  class A{  public: void in(){ cout << "in"; }  };  class C: public A{};  int main(){  C \*c = new C();  c->out();  return 0;  } | compiling error  previous definition of class A |
| class X{};  class Y: public X{};  class Z: public X{};  int main(){  Z \*z=new Z();  Y \*y=new Y();  z=y;  cout<< (z==y) <<endl;  return 0;  } | can't convert Y\* to Z\* in assignment  Note: objects of various class (different class and no inheritance)can't be converted |
| class A{  friend void f();  private: int field;  public:  int set(int x){return field = ++x;}  int get(){return ++field;}  };  void f(A &a){a.field /= 2;}  int main(){  A a;  a.set(2);  f(a);  cout << a.get() << endl;  return 0;  } | errro: inaccessible to private attribute field  Note: though f() is friend of class A , attribute field is private |

### access specifier

public, private, or protect

A **protected** member variable or function is very similar to a private member but it provided one additional benefit that they can be accessed in child classes which are called derived classes.

public inheritance: public in base class keep public in derived class. protected remain protected. can't access private

protected inheritance: public and protected in base class are protected in derived class

private inheritance: public and protected in base class are private in derived class

*if a component is declared as public and its class is inherited as public, the resulting access is public*.

|  |  |
| --- | --- |
| class Uno{  protected: char y;  public: char z;  };  **class Due: protected Uno{**  public:  void set(){ y='a'; z='z'; }  void out(){cout << ++y << --z;}  };  int main(void) {  Due b;  b.set(); b.out();  return 0;  } | output: by  Either "class Due: protected Uno" or "class Due: public Uno" is ok in this case. |
| class A{  char a;  protected: char b;  public: char c;  A(){a='a';b='b';c='c';}  };  class B: public A{  char d;  public:  void set(){ c='e'; d='d';}  void get(){ cout << c << d;}  };  int main(void) {  B b;  b.set();   b.get();  return 0;  } | output: ed |
|  |  |
|  |  |
| class A{  public: int p;  A(){r=2;}  private: int q;  **protected: int r;**  };  **class B: private A{};**  **class C: public B{**  public: void out(void){ cout<< r;}  };  int main(void) {  C c;c.out();  return 0;  } | error: 'int A::r' is protected within this context  in Class C, can access protected r defined in grandparent class A usually. but parent of B is private A. so A.r is private for child class C. |
| class A{  char a;  protected: char b;  public: char c;  A(){a='a';b='b';c='c';}  };  class B: public A{  char d;  public: void set(){ c='e'; d='d';}  };  int main(void) {  B b;b.set();   cout << b.c << b.d;  return 0;  } | 'char B::d' is private within this context  char d in class B is private. can't be used in main() |
|  |  |
|  |  |

### virtual methods

A virtual function is a member function which is declared within a base class and is overriden by a derived class.

virtual methods are bound at execution time. non-virtual methods are bound at compiling.

1. Virtual functions cannot be static.
2. A virtual function can be a friend function of another class.
3. Virtual functions should be accessed using pointer or reference of base class type to achieve run time polymorphism.
4. The prototype of virtual functions should be the same in the base as well as derived class.
5. They are always defined in the base class and overridden in a derived class. It is not mandatory for the derived class to override (or re-define the virtual function), in that case, the base class version of the function is used.
6. A class may have [virtual destructor](https://www.geeksforgeeks.org/virtual-destructor/) but it cannot have a virtual constructor.

|  |  |
| --- | --- |
| class A{  public: virtual void out(){cout << "A";}  };  class B: public A{  **public: virtual void out(){cout << "B";}**  };  class C: public A{  **public: virtual void out(){cout << "C";}**  };  int main(void) {  **A \*a = new B(),  \*b = new C();**  b->out();   a->out();  return 0;  } | output: CB  Note: inheritance: A🡨B, A🡨C  implementation inheritance: virtual methods defined  class A,B, C could act as base class  A::\*a-> B::out(), A::\*b -> C::out()  **Note: In this case, removing virtual from class B and C would not impact on outcome** |
| class A{  **public: void out(){cout << "A";}**  };  class B: public A{  **public: void out(){cout << "B";}**  };  class C: public A{  **public: void out(){cout << "C";}**  };  int main(void) {  A \*a = new B(),  \*b = new C();  b->out();   a->out();  return 0;  } | output: AA  Not virtual keyword included  A::\*a, A::\*b is upcasting |
| class A{  **public: void out(){cout << "A";}**  };  class B: public A{  **public: virtual void out(){cout << "B";}**  };  class C: public A{  **public: virtual void out(){cout << "C";}**  };  int main(void) {  A \*a = new B(),  \*b = new C();  b->out();   a->out();  return 0;  } | output: AA  virtual exist in derived class will not impact on outcomes |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| class base {  public:  virtual void print()  {  cout << "print base class" << endl;  }     void show() {  cout << "show base class" << endl;  }  };    class derived : public base {  public:  void print(){  cout << "print derived class" << endl;  }    void show(){  cout << "show derived class" << endl;  }  };    int main(){  base\* bptr;  derived d;  bptr = &d;    // virtual function, binded at runtime  bptr->print();    // Non-virtual function, binded at compile time  bptr->show();  } | output:  print derived class  show base class |
| class A{  public:  A():val(0) {}  int val;  **virtual void run(){cout<<val;}**  };  class B: public A {};  class C: public B {public:  void run(){cout<<val+2;}};  void Do(A \*a){  B \*b; C \*c;  **if(b=dynamic\_cast<B\*>(a))** b->run();  **if(c=dynamic\_cast<C\*>(a))**c->run();  a->run();  }  int main(){  **A \*a=new C();**  Do(a);  return 0;  } | output: 222  **"A \*a=new C();" is equal to**  **A\* a; C c; a=&c;**  if use dynamic\_cast, the base class A must define virtual method. because dynamic\_cast do conversion during execution time. |
| class A{  public:  A():val(0) {}  int val;  void run(){cout<<val;}  };  class B: public A {public: virtual void run(){cout<<val+2;}};  class C: public B { };  void Do(A \*a){  B \*b; C \*c;  if(b=static\_cast<B\*>(a)) b->run();  if(c=dynamic\_cast<C\*>(b))c->run();  a->run();  }  int main(){  A \*a=new C();  Do(a);  return 0;  } | output:220 |
|  |  |
| class X{public: void shout(){cout<<"X";}};  class Y: public X {public: void shout(){cout<<"Y";}};  class Z: public Y {public: void shout(){cout<<"Z";}};  int main(){  Z \*z = new Z();  z->shout();  static\_cast<Y \*>(z)->shout();  static\_cast<X \*>(z)->shout();  Y \*y = new Z();  y->shout();  static\_cast<Z \*>(y)->shout();  static\_cast<X \*>(y)->shout();  return 0;  } | output: ZYXYZX  1. z.shout() override parent. convert to Y \*, then y.shout() override z.shout() and x.shout()  2. static\_cast and dynamic\_cast are equal  3**. it is ok to convert child to parent. illegal to convert parent to child** |
| class X{public: virtual void shout(){cout<<"X";}};  class Y: public X {public: void shout(){cout<<"Y";}};  class Z: public Y {public: void shout(){cout<<"Z";}};  int main(){  X \*x = new Z();  x->shout();  dynamic\_cast<Y \*>(x)->shout();  dynamic\_cast<Z \*>(x)->shout();  cout<<endl;  Y \*y = new Z();  y->shout();  dynamic\_cast<X \*>(y)->shout();  dynamic\_cast<Z \*>(y)->shout();  cout<<endl;  X \*xx = new Y();  xx->shout();  dynamic\_cast<Y \*>(xx)->shout();  dynamic\_cast<Z \*>(xx)->shout();  cout<<endl;  return 0;  } | output:ZZZ ZZZ YY  x.shout() is virtual method. \*x or \*t is overridden by z.out(). \*xx is overridden by y.shou(). |
| **class X{public: virtual void shout(){cout<<"X";}};**  class Y: public X {public: void shout(){cout<<"Y";}};  class Z: public Y {public: void shout(){cout<<"Z";}};  int main(){  Z \*z = new Z();  z->shout();  static\_cast<X \*>(z)->shout();  static\_cast<Y \*>(z)->shout();  return 0;  } | output:ZZZ  x.shout() is virtual method. x.shout() and y.shout() is overridden by z.shout() |
| int main(){  Y \*y = new Z();  y->shout();  dynamic\_cast<X \*>(y)->shout();  dynamic\_cast<Z \*>(y)->shout();  cout << endl;  X \*x = new Z();  x->shout();  // wrong: dynamic\_cast<Y \*>(x)->shout();  // wrong: dynamic\_cast<Z \*>(x)->shout();  cout << endl;  Y \*yy = new Y();  dynamic\_cast<X \*>(yy)->shout();  // wrong: dynamic\_cast<Z \*>(yy)->shout();  cout << endl;  return 0;  } | output:ZXZ X X  y.shout() is overridden by z.shout(). convert class Y\* (children) to class X\*(parent). y.shout() is virtual method, overridden by x.shout() |
|  |  |

## overloading

**13 Overloading**

1 When two or more different declarations are specified for a single name in the same scope, that name is said to be overloaded. By extension, two declarations in the same scope that declare the same name but with different types are called overloaded declarations. Only function and function template declarations can be overloaded; variable and type declarations cannot be overloaded.

### overload attributes

|  |  |
| --- | --- |
| class A{  public:string s;  A(){s="A"; cout<<s;}  A(string s){cout<<"Astr";}  };  class B: public A{  public:string s;  B(){cout<<"Bno";}  B(string s){s=s; cout<<s;}  };  int main(void) {  B b2("good");  return 0;  } | output:Agood  when b2 is initialized, B::s overrides A::s |
| class A{  public:int x;  A(){x=0;}  A(int x){this->x=x;}  };  class B: public A{  public:using A::x;  B(){}  B(int x){this->x = x;}  };  int main(void) {  B c1, c2(5);  cout << c1.x << c2.x << endl;  return 0;  } | output:05  Note Adding "using A::x" will not impact outcome. this statement just explicitly declare inheritance |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

### overloading of constructor

|  |  |
| --- | --- |
| class A{  public:  int s;  A(){cout<<"Ano";}  A(string s){cout<<"Astr";}  };  class B: public A{  public:  B(){cout<<"Bno";}  B(string s){cout<<"Bstr";}  };  int main(void) {  B b2("good");  return 0;  } | output: AnoBstr  B::B(string s) override A::A(string s) |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

### overloading methods

|  |  |
| --- | --- |
| class A{  public: void out(void){ cout << "A"; }  };  class B: public A{  public: void out(){ cout << "B"; }  };  class C: public B{  public: void out(){ cout << "C"; }  };  int main(){  A a;B b;C c;  a.out();  b.out();  c.out();  return 0;  } | output:ABC  B.out() override A.out() in B  C.out() override B.out() in C |
| class A{  public: void out(void){ cout << "A"; }  };  class B: public A{  public: void out(){ cout << "B"; }  };  class C: public A{  };  int main(){  A a;B b;C c;  a.out();  b.out();  c.out();  return 0;  } | output: ABA |
| class A{  public:  void out(void){ cout << "A"; }  void toA(){ cout << "a"; }  };  class B: public A{  public:  void out(){ cout << "B"; }  void toB(){ cout << "b"; }  };  class C: public B{  public:  void toC(){ cout << "c"; }  };  int main(){  **A \*a = new B();  B \*b = new C();**  a->out(); b->out();  return 0;  } | output: AB  though new B() but the \*a is instance of class A  convert \*B to \*A, \*C to \*B in initialization. |
|  |  |
|  |  |

### overload operator

bool operator==(). The approach below are equal

|  |  |
| --- | --- |
| class person{  private:  int age;  public:  person(int a){  this->age=a; }  // overload == compare age  **inline bool operator==(const person &ps) const{**  if (this->age==ps.age)  return true;  return false;  }  };  int main(){    person p1(10), p2(20);    if(p1==p2) cout<< "Age is equal!";    else cout<< "Age is different";    return 0;  } |  |
| class person{  public:  int age;  person(int a){  this->age=a; }  };  // out of class  bool operator==(person const &ps1, person const &ps2){  if (ps1.age==ps2.age)  return true;  return false;  }  int main(){    person p1(10), p2(20);    if(p1==p2) cout<< "Age is equal!";    else cout<< "Age is different";    return 0;  } |  |
|  |  |
|  |  |

operator+

|  |  |
| --- | --- |
| enum T { A=2, B=-1, C };  **T operator+(T t, int i){**  T C = T(2);  switch(t){  case A: return T(A);  case B: return static\_cast<T>(i);  default: return (T)1;  }  }  int main(void) {  **T i = A+2;**  cout << i << C;  return 0;  } | output:20  1. pass enum value to operator+. so T C =T(2) will not affect outer enum T. So C=0.  2. overload enum: "A+2" is illegal for enum type, but operator+ is overloaded. |
| class A{  double re, im;  public:  A(): re(1), im(1) {}  A(double r, double i): re(r), im(i) {}  **A operator+(A &); //declaration**  void out(){cout << re << im;}  };  //define operator+  **A A::operator+ (A &a){**  **A c(this->re + a.re, this->im + a.im);**  **return c;**  **}**  int main(void) {  A x(1,2), y, z;  z = x+ y; z.out();  return 0;  } | output: 23 |
| class complex{  double re, im;  public:  complex(): re(1), im(0.4) {}  complex operator> (complex &t);  void Print(){ cout << re <<" "<< im; }  };  complex complex::operator> (complex &t){  complex temp;  temp.re = this->re ? t.re : 2;  temp.im = this->im ? t.im : 0.6;  return temp;  }  int main(void) {  complex c1, c2, c3;  c3= c1 > c2;  c3.Print();  return 0;  } | output: 1 0.4 |
| class Box{  private: int length, width, height;  public:  Box(int length, int width, int height){  this->length = length;  this->width = width;  this->height = height;  }  Box(int length){  this->length = this->width = this->height = length;  }  int getVolume(void){  int v= this->length \* this->height \* this->width;  return v;  }  **int operator+(const Box &b2){**  **this->length += b2.length;**  **this->width += b2.width;**  **this->height += b2.height;**  **}**  };  int main(){  Box b1(1,2,4), b2(5);  cout << b1.getVolume() << endl;  cout << b2.getVolume() << endl;  **b1 + b2;**  cout << b1.getVolume() << endl;  return 1;  } |  |

operator ++ and operator --

// Declare prefix and postfix increment operators.

Point& operator++(); // Prefix increment operator.

// Declare prefix and postfix decrement operators.

Point& operator--(); // Prefix decrement operator.

post operator

Point operator++(int);

Point operator--(int);

|  |  |
| --- | --- |
| enum e {a=1,b,c,d};  //prefix operator  **e& operator--(e &x){ x=b; return x; }**  int main(void) {  e f=c;  **cout << int(--f) <<endl;**  return 0;  } | output:2 |
| enum e {a=1,b,c,d};  **e& operator--(e &x, int){ x=b; return x; }**  int main(void) {  e f=c;  **cout << int(f--) <<endl;**  return 0;  } |  |
| class Int{  public:  int v;  Int(int a){v=a;}  Int& operator--(){ ++v; return \*this; }  Int& operator--(int v){ v+=2; return \*this; }  };  ostream& operator<< (ostream &o, Int &a){  return o << a.v++;  }  int main(void) {  Int i=0; cout << --i << i--;  return 0;  } | output:12 |
| enum T {A=2, B=-1,C};  class Int{  public: T v;  Int(T a){ v = a; }  **Int& operator++(){ v = T(v+2); return \*this;}**  };  **ostream& operator<<(ostream &o, Int &a){**  **++a;   return o << a.v;**  **}**  int main(void) {  Int i = B;cout << i;  return 0;  } | output: 1 |

operator<<

|  |  |
| --- | --- |
| class Int{  public: int v;  Int(int a){v=a;}  };  **ostream& operator<<(ostream &o, Int &a){**  **return o << a.v++;**  **}**  int main(void) {  Int i = 1;cout << i;  return 0;  } | output:1  1. should pass two reference arguments  2. The below is ok.  void operator<<(ostream &o, Int &a){   o << a.v++;  } |
| class Int{  public: int v;  Int(int a){v=a;}  Int& operator[](int x){ v += x; return \*this;}  };  ostream& operator<<(ostream &o, Int &a){  return o << a.v;  }  int main(void) {  Int i=2;  **cout << i.v << i[2];**  return 0;  } | output: 44  **Both  "cout << i[2];" and  "cout << i.v << i[2];" are ok.** |
| class Int{  public: int v;  Int(int a){v=a;}  Int& operator++(int x){ v += 2; return \*this;}  };  ostream& operator<<(ostream &o, Int &a){  return o << a.v;  }  int main(void) {  Int i=0;  **cout << i++ << i.v;**  return 0;  } | output:20  The first << is overloaded by operator<<, but the 2nd << is not. so first run i.v=0, then trigger i++->i.v=2.  if "cout << i.v << i++;", the ouput is 22. |
|  |  |
|  |  |
|  |  |
|  |  |

### syntax errors

1. only overload class or enum type operator. Built-in data type can't be overloaded.

|  |  |
| --- | --- |
| class A{  public: int v;  A():v(1) {}  A(int i): v(i) {}  **void operator\*\*(int a) { v\*=a; }**  };  int main(void) {  A i(2);  **i\*\*2;**  cout << i.v <<endl;  return 0;  } | \*\* can't be overloaded. \* is unary operator |
| enum e {a=1,b,c,d};  e& operator--(e &x){ x=b; return x; }  int main(void) {  e f=c;  cout << int(f--) <<endl;  return 0;  } | overloading should be prefix operator. |
| string operator> (float l, float r){  if( int(l) > int(r) ) return "true";  return "false";  }  int main(void) {  float l=2.0, r=2.9999;  cout << (l > r);  return 0;  } | must have an argument of class or enumerated type  It is not allowed to overload an operator for a built-in type including int, float, double, char, etc.  correct: create class ,remove global operator to class operator |
| class Int{  public: int v;  Int(int a){v=a;}  };  ostream& operator<<(Int &a){  return cout << a.v;  }  int main(void) {  Int i=1;  cout << i;  return 0;  } | compiling error.  operator<< is binary operator. should pass two arguments  should be  ostream& operator<<(ostream &o, Int &a){  return o << a.v;  } |
| class N{  public: float x;  N(){x=0.0;}  N(float a){x=a;}  N(N& n){x=n.x;}  **N& operator<<(N &y){ return \*new N(x\*10); }**  };  int main(void) {  N a(2.0), b(4.0);  N c = a << 1;  cout << c.x;  return 0;  } | compiling error  mismatch data type: a<<1;  correct a<<b; |
| class Int{  public: int v;  Int(int a){v=a;}  **Int& operator--(){ ++v; return \*this;}**  };  ostream& operator<<(ostream &o, Int &a){  return o << --a.v;  }  int main(void) {  Int i=2;  cout << i;  return 0;  } | output:1  "o << --a.v" doesn't apply operator-- because a.v is int type rather than object. Soo a.v=2, --a.v, a.v=1  **The below will load operator-- : o << (--a).v;** |
| class N{  public: float x;  N(){ x = 0.0; }  N(float a){ x=a; }  N(N& n){ x=n.x; }  };  N& operator=(N &y, float f){ return \*new N(f); }  int main(void) {  N a; a = 2.0;  cout << a.x;  return 0;  } | compiling error   'N& operator=(N&, float)' must be a nonstatic member function  operator= can not be defined as global operator |
|  |  |

## composition

|  |  |
| --- | --- |
| using namespace std;  class A {  public:  void Do(void) { cout << "A is doing something" << endl; }  };  class B {  public:  void Do(void) { cout << "B is doing something" << endl; }  };  class Compo {  public:  A f1;  B f2;  };  int main(void) {  Compo co;  co.f1.Do();  co.f2.Do();  return 0;  } |  |
|  |  |
|  |  |
|  |  |

## special class

### friend class

**Friend Class** A friend class can access private and protected members of other class in which it is declared as friend. It is sometimes useful to allow a particular class to access private members of other class. For example, a LinkedList class may be allowed to access private members of Node.

Following are some important points about friend functions and classes:   
**1)** Friends should be used only for limited purpose. too many functions or external classes are declared as friends of a class with protected or private data, it lessens the value of encapsulation of separate classes in object-oriented programming.  
**2)** Friendship is not mutual. If class A is a friend of B, then B doesn’t become a friend of A automatically.  
**3)** Friendship is not inherited (See [this](https://www.geeksforgeeks.org/g-fact-34/)for more details)  
**4)** The concept of friends is not there in Java.

|  |  |
| --- | --- |
| #include <iostream>  using namespace std;  class A {  friend class B;  friend class C;  private: int field;  protected: void print(void) { cout << "It's a secret, that field = " << field << endl; }  };  class C {  public: void DoIt(A &a) { a.print(); }  };  class B {  public: void DoIt(A &a, C &c) {  a.field = 111; c.DoIt(a); }  };  int main(void) {  A a; B b; C c;  b.DoIt(a,c);  return 0;  } |  |
| class A{  friend class B;  int a;  public:  A(): a(1) {}  int f() { return a; }  };  class B{ public:static void f(A &a){a.a++;} };  int main(){  A a; B::f(a);  cout<< a.f()<<endl;  return 0;  } | output: 2 |
| **class B;**  class A{  **friend class B;**  int a;  public:  A(): a(4) {}  **void f(B &b, A &a);**  int out(void) { return a; }  };  class B{  **friend class A;**  int b;  public:  B(): b(2){}  **void f(A &a){a.a /=b;}**  };  **void A::f(B &b, A&a){ b.f(\*this); }**  int main(){  A a; B b;a.f(b, a);  cout<< a.out()<<endl;  return 0;  } | output: 2  Note: pay attention to order of class A and B. "Class B;" should be declared before definition.  " void A::f(B &b, A&a){b.f(\*this);}" should declared after class B. |
|  |  |
|  |  |
|  |  |

### const class

### abstract classes

Abstract methods defined in a based class provide an interface for derived classes. Abstract can not be implemented at base class. Declaring abstract is specified by placing "=0" in it declaration as follows

|  |  |
| --- | --- |
| class A {  public: int length;  **virtual double getVolume() = 0;**  };  class Box: public A {  public:  Box(int x){length=x;}  **double getVolume(){**  **return length\*length\*length;**  **}**  };  int main(void) {  Box b(10); cout<< b.getVolume();  return 0;  } | output: 1000 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| class A {  public: int length;  virtual double getVolume() = 0;  void print(){cout<<"A";}  };  int main(void) {  **A a;**  return 0;  } | error: cannot declare variable 'a'  to be of abstract type 'A'  class A is abstract class, which can't be implementated. |
| class A {  public: int length;  **virtual double getVolume() = 0;**  };  class Box: public A {  public:  Box(int x){length=x;}  int getVolume(){  return length\*length\*length;  }  }; | conflicting return type specified for  virtual method  abstract is double, implementation should also be double. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## coding errors

1. object access private attribute

2. wrong data type of arguments when instantiate class.

3. initialize static variable in constructor.

4. miss semicolon end of class

5. access attribute which is not initialized.

|  |  |
| --- | --- |
| class A{  int a;  };  int main(){  A a;  a.a=1/2;  cout << a.a << endl;  return 0;  } | attribute of class A.a is private by default. can't access a.a directly.  correct: public: int a; |
| class A{  public:  static int a;  A() {a=1;a++;}  A(A &aa) {a++;}  };  int main(){  A a, b(a), c(b);  cout << A.a << endl;  return 0;  } | Two errors:  1.static variable a can't be initialized within constructor. that should be initialized outside of class: "int A::a=1;"  2. can't directly apply "A.a". should be "A::a" or "a.a". A is class. a is instance of class A. |
| class Test {  bool isVal() const {  return isVal;  }  private:  bool isVal;  }; | overloading is ok for functions. variable isVal can't overload the function isVal() |
| class A{  float v;  A(){v=1.0;}  A(A &a) {A::v=a.v; cout<<"1";}  ~A(){cout<<"0";}  float set(float v){  A::v=v;  return v;  }  float get(float v){  return A::v;  }  };  int main(){  A a, \*b=new A(a),  \*c=new A(\*b);  c->get(b->get(a.set(1.0)));  delete b;  delete c;  return 0;  } | All attributes and methods are private if no explicit declaration.  correct: insert "public: " before constructor. |
| class A{  public:  A():val(0) {}  int val;  void run(){cout<<val;}  };  class B: public A {  public: virtual void run(){cout<<val+2;}  };  class C: public B { };  void Do(A \*a){  B \*b; C \*c;  if(b=dynamic\_cast<B\*>(a))  b->run();  if(c=dynamic\_cast<C\*>(b))  c->run();  a->run();  }  int main(){  A \*a=new C();  Do(a);  return 0;  } | cannot dynamic\_cast 'a' (of type 'class A\*') to type 'class B\*' (source type is not polymorphic)  correct: if(b=static\_cast<B\*>(a)) |
| class X{  protected:  int v;  };  class Y: protected X{  Y(): v(0){}  };  int main(){  Y \*y = new Y();  cout << y->v;  delete y;  return 0;  } | 'int X::v' is protected within this context |
|  |  |
|  |  |

# Template

## Template

## STL sequential container

## STL associative containers

## non-modifying STL algorithms

## modifying STL algorithms

## sorting STL operations

## STL merge operations

## STL utilities an functional libary

# STL: container

## basic

sequence containers: sequential means data are ordered by the order of inputs, and stored in contiguous memory. STL defines **string, vector, linked list, deque**.

association containers ( element is key-value pattern): map, multi-map, set, multi-set

container adapters: stack, queue, priority-queue

methods 1 : create, size/length, max\_size, empty, copy, concatenate

methods 2: update, insert (into start/end/middle), delete (from start/end/middle/all)

|  |  |  |
| --- | --- | --- |
|  | **action** | **Note** |
| **sequential container**:  1. data is ordered by the input order. | array | the same as C. size-fixed array. can't be changed after it is initialized |
| string | In C, string is not a built-in data type, and denoted as a char array. In C++, string is defined by class string like char vector. |
| vector | dynamic array. allow random access. |
| list | double-linked list (struct) |
| forward\_list | single\_linked list (struct) |
| deque | double-ended queue (de-que) |
|  |  |  |
| **associative container**:  1. store data in key-value pair.  2. value is sorted. | map | 1. map only stores unique keys.  2. There is only one value for every key  3. Keys are in sorted order.Searching for an element is very fast. |
| multi-map | key |
| set | distinct and sorted values.  All values must be in ascending or descending.  All values must be unique. |
| multi-set | sorted values. but allow duplicate values |
|  |  |  |
| **container operator**: | stack | a sequential container with one end. Stack only works on the top element namely access/add/delete the top one. "Last In - First Out"  methods: push(), pop(), peek(), isFull(), isEmpty()  don't support random access |
| queue | a sequential container with front and rear side. dequeue is to remove element from the front side. Enqueue is to add element from the rear side. "First In - First Out"  methods: push(), pop(), peek(), isFull(), isEmpty()  don't support random access |
| priority-queue |  |
|  |  |  |
| tree | heap | a complete binary tree where all the levels of a tress are fully filled-- any parent must have exact two children. Among all nodes, values of parents are either >= or <= values of children.  min-heap: an ordered balanced binary tree. value of parent(root) is <= that of children.  max-heap: an ordered balanced binary tree. value of parent(root) is >= that of children. |
|  |  |

|  |  |  |
| --- | --- | --- |
| **action** | **code** | **Note** |
| string |  |  |
| vector |  |  |
| linked list |  |  |
| deque |  |  |
|  |  |  |
| map |  |  |
| multi-map |  |  |
| set |  |  |
| multi-set |  |  |
|  |  |  |
| stack |  |  |
| queue |  |  |
| priority-queue |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Difference of array, vector, and list:**

They stores sequential elements.

**Array** is lower-level data structure in C/C++. An array stores a fixed-size sequential collection of elements with same data type. The memory size is fixed, equal to data type times by size of elements. being smaller compared with vector, and more access efficient.

**Vector** is defined in the class <vector> of STL of C++. Vector is class-object based. Vector is dynamics: could insert update, delete or replace or insert elements. So vector required more memory.

Elements of array and vector are stored in contiguous memory. Array contains elements with their memory location, and access element by memory address. Vector enable direct access to certain element using subscript operator, which takes more times but are safer.

**List** is defined in the class <list> of STL of C++. Elements are not contiguous stored, which internally works as **double-liked list** (struct type) in C. List has not index for elements. It can't allow random access, for example to access a certain element by index.

## Create

For creating a new container, usual for patterns are followed no matter what the container is. For example, String is defined by class string. declare a string is equal to instantiate object of class string. So Value assignment to string object, or the use of constructor are acceptable.

copy constructor: **string s="ABC";**

create empty object and then assign it. : **string s; s="ABC";**

initializer list: **string s "ABC";**

create object: **string s("ABC");**

initial identical elements: c.assign(<number>, <value> )

|  |  |  |
| --- | --- | --- |
| **action** | **code** | **Note** |
| string | #include <string>  string s1 = "ABC"; //declare and initialize  string s2; s2="ABC"; //declare then initialize  string s3("ABC"); // constructor pattern  string \*p; p = new string("pizza"); //pointer pattern |  |
|  | string sky;  **sky.assign(4, '\*');** | sky=\*\*\*\* |
| vector | #include <vector>  vector <int> v1 = {1, 2, 3};  vector <int> v2 {1, 2, 3};  vector <int> v3(3); v3={1, 2, 3};  vector <int> v4; v4={1, 2, 3}; | vector is dynamic array  2. "vector <int> v2();" is wrong |
|  | vector<int> v;  v.assign(3, 3); | v=333 |
| list | #include <list>  list <int> mylist = {1,2,3};  list <int> mylist {1,2,3};  list <int> mylist; mylist.push\_back(1);  list <int> mylist(10, 0);//size=10. all is 0.  list <int> mylist(); | "list <int> mylist[4] = {1,2,3};  " is wrong. |
| deque | #include <deque>  deque <int> d = {1,2,3};  deque <int> d {1,2,3};  deque <int> d; d.push\_back(1);  deque <int> d(); //empty deque  deque <int> d(10);//size=10  deque <int> d(10, 0);//size=10,values=0 |  |
| pair | #include <utility>  pair<char, int> p = {'a',1};  pair<char, int> p {'a',1};  pair<char, int> p, p= {'a',1};  pair<char, int> p, p.first = 'a', p.second=1; |  |
| map | #include <map>  map <char, int> m {{'a',3}};  map <char, int> m = {{'a',3}};  map <char, int> m{};//empty  map <char, int> m, m['a']=4;  map <char, int> m, m.insert(pair<char, int> p {'d',4}); |  |
| multi-map | #include <map>  typedef multimap<const char, int> myType;  myType m;  m.insert(myType::value\_type('a',2));  m.insert(myType::value\_type('a',3)); |  |
| set | #include <set>  set<int> s = {1,2,3};  set<int> s; s.insert(1);  int a[]={1,2,3}; set<int> s; s.insert(a[0]);  vector<int> v={1,2,3}; set<int> s(v.begin(),v.end()); | wrong: set<int> s {1,2,3}; |
| multi-set | #include <set>  multiset<int> s = {1,1,3};  multiset<int> s; s.insert(1);  int a[]={1,1,3}; multiset<int> s; s.insert(a[0]);  vector<int> v={1,1,3}; multiset<int> s(v.begin(),v.end()); |  |
|  |  |  |
| stack | #include <stack> | stack is "last-in, first-out" (LIFO). The two basic operations is PUSH (put an element on the top), and POP (remove the top element). |
| queue | #include <queue> |  |
| priority-queue |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**create vector**

vector is dynamic array, so don't need to specify size when declaring it.

initialize a int vector

|  |  |
| --- | --- |
| vector <int> v;  v.push\_back(20); | int vector |
| vector <int> v(3); | int vector with size=3, value=0 |
| vector <int> v(3, 10); | int vector with size=3, value=10 |
| vector <int> v{1,2,3}; | int vector with various values |
| vector <int> v={1,2,3}; | the same as the above |
| vector<int> v(3);  fill(v.begin()+1,v.end(), 10); | int vector with size=3, and update the values of 2nd-end with 10 |
|  |  |

|  |  |
| --- | --- |
| vector <int> a={1,2,3};  cout << a[0]; | integer type vector |
| vector <char> a={'A','B'};  cout << a[1]; | char type vector |
| vector <string> a={"ABC", "DEF"};  cout << a[1]; | string type vector |
| vector <float> t;  t = {3.,2.,1.}; |  |
| vector <float \*> t = {new float[1], new float[1], new float[1] };  for(int i=0; i<t.size(); i++){  float \*p = t[i];  \*p=i;  }  cout << \*t[0] << \*t[1] << \*t[2] << endl;  for(int i=0; i<t.size(); i++){  delete [] t[i];  } | float pointer array type |
| vector <char> t(5) | it is ok to specify size |

## Get

1. overload operator[]: Random access by index in array, string, vector. Access value by key in map/multimap.

2. Iterator: array, string, vector, map, set

\*c.begin() return the first element

\*(c.begin()+n) return the (n+1)th element

\*(c.end()-1) returns the last element

\*next(c, n) return the (n+1)th element

<container>::iterator iter=c.begin(); advance(iter,n); \*iter; get the (n+1)th element

3. container adapters: get the top, get the last

|  |  |  |
| --- | --- | --- |
| **action** | **code** | **Note** |
| string | string s = "ABC"; cout << s[0]; | A |
| string s="abc"; cout << s[3] <<s[-1]; | no output if index is out of bounds or is negative. |
| string s="abc"; cout << \*s.begin()<< \*(s.begin+1); | ab. |
| string s="abc"; cout << \*(s.end()-1); | c. get the last element.  Note: \*s.end() return '\0' |
| string s="abcd"; cout << \*next(s.begin(),2); | c. |
| string s="abcd"; cout << \*next(s.end(),-2); | c. |
| string s="abcd";  string::iterator iter=s.begin();  advance(iter, 2); cout << \*iter; | c.  Note: advance() has no returns |
| vector | vector <string> s = {"Mon", "Tue",\  "Wed", "Thu", "Fri"};  cout << s[1]; |  |
| vector<int> v {1,2,3}; cout << v[6]; | unpredictable value |
| vector<int> v {1,2,3}; cout << \*v.begin(); | 1 |
| vector<int> v {1,2,3}; cout << \*(v.begin()+1); | 2 |
| vector<int> v {1,2,3}; cout << \*next(v.begin(),1); | 2 |
| vector<int> v {1,2,3}; cout << \*next(v.end(),-2); |  |
| vector<int> v {1,2,3}; cout << \*(v.end()-2); | 2 |
| vector<int> v {1,2,3};  vector<int>::iterator iter = v.begin();  advance(iter, 1); cout << \*iter; | 2 |
| linked list |  |  |
| deque |  |  |
|  |  |  |
| map | map <char, int> m = {{'a',3}, {'b',2}};  cout << m['b']; | output: 2 |
| map <char, int> m = {{'a',3}, {'b',2}};  cout << m['c']; | output:0. the key 'C'doesnot exist |
|  | map<char, int> m = {{'a',3}, {'b',2}};  map<char, int>::iterator iter = m.begin();  cout<< iter->first << iter->second; | a3 |
|  | map<char, int> m = {{'c',3}, {'a',1}, {'b',2}};  map<char, int>::iterator iter = m.begin();  advance(iter, 1);  cout<< iter->first << iter->second; | b2 |
| multi-map |  |  |
| set |  |  |
| multi-set |  |  |
|  |  |  |
| stack |  |  |
| queue |  |  |
| priority-queue |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Insert

sequence containers: insert to the first, append (insert to the last), insert in the middle

c.push\_back(<el>)  
c.push\_front(<el>)

c.insert(<iter>,<el>)

**insertion in string**

str.push\_back()

**str.insert()**

str.append()

overload operator+

string s= "ABC";

|  |  |  |
| --- | --- | --- |
|  | code | Note |
| push to the first | s='a'+s;  s="abc"+s; | overload operator+ |
| append the last | s.push\_back('a');  s.push\_back('abc'); | str.push\_back() |
| s +='a';  s+= "abc"; | overload operator+ |
| s.append("abc"); | output:ABCabc |
| s.append(2, 'a'); | output: ABCaa |
| insert one element | s.insert(2, 'a'); | str.insert() |
| insert some elements | s.insert(2, "abc"); | output:ABabcC |
|  | string a = "a";  string b = a+"b";  b=a+"b"+b;cout<< b; | output: abab  use + to concatenate strings. |
|  | string s = "A" + "B"; | wrong: concatenate operator + can't work for two literals. At least one if variable |
|  | string a = "a";  string b=a+'2'; | b is a2  concatenate character |

codes

|  |  |
| --- | --- |
| string quote = "Whyserious?", anyword = "monsoon";    quote.insert(3, 2, ' ').insert(4, anyword, 3, 2);    cout << quote; | output: Why so serious? |
| string s="AB";  s.append(s).push\_back(s[s.length()-1]);  cout<<s; | ABABB |
| string s1[]={"A", "Z"}, s="";  for(int i=0; i<2;i++)  cout << s.append(s1[i]).insert(1,"\_"); | output: A\_A\_\_Z |

**insertion in vector**

v.insert()

v.push\_back()

Example: vector <int> v={1,2,3};

|  |  |  |
| --- | --- | --- |
|  | code | output |
| push to the first | v.insert(v.begin(), 0); | vector.insert() |
| append the last | v.push\_back(4); | vector.push\_back() |
| v.insert(v.end(),4); | vector.insert() |
| insert one element | v.insert(v.begin()+1, 4); | output: 1,4,2,3 |
| insert some elements | v.insert(v.begin()+1, {4,5}); | output: 1,4,5,2,3 |
| concatenate two vectors | v.insert(v.end(), {4,5}); |  |
|  | vector <int> v2={4,5};  v.insert(v.end(), v2.begin(), v2.end()); |  |
|  |  |  |

**associative containers**

overload operator[]

c.insert()

|  |  |  |
| --- | --- | --- |
|  |  |  |
| map  insert/replace one element. | map <char, int> m = {{'a',1}};  m['b']=2; | {'a':1, 'b':2} |
| map <char, int> m = {{'a',1}};  **m.insert(pair<char, int> ('b',2));** | {'a':1, 'b':2} |
| set | set<int> s; s.insert(1); s.insert(1);  cout << \*s.begin(); | {1} |
| multiset | multiset<int> s; s.insert(1); s.insert(1);  cout << \*s.begin(); | {1,1} |
|  |  |  |
|  |  |  |
|  |  |  |

## Update or replace

str.replace(str.begin(), str.end(), str2, str2.begin(), str2.end()): string type. not for vector

c.swap(c2)

|  |  |  |
| --- | --- | --- |
| **action** | **code** | **Note** |
| string | string s("GAATTC"); string ss;  **s.replace(0, 3, "TCC");** | s=TCCTTC |
| string s("usr/bin/test.text");  **s.replace(s.find('.'), 5, "");** | s=usr/bin/test  remove path extension |
| string s("a,b,c,");  **s.replace(s.begin(), s.begin()+3, "#");** | s=#,c, |
| string line = "this@ is@ a test string!";  char\* str = "12345";  //char array  **line = line.replace(0, 9, str, 4);** | line= 1234 a test string!  Note: str should be char array. string type will cause compiling error. |
| string to\_do = "I'll think about that in one hour";  string schedule = "today yesterday tomorrow";  **to\_do.replace(22, 12, schedule, 16, 8);** | to\_do=I'll think about that tomorrow  compared with the above |
| string s="abc"; string s2="123";  **s.swap(s2);** cout << s << ","<< s2; | 123,abc |
|  |  |
| string a ="ABCDEF", b;  b = a.substr(1,1)+a.substr(4)+a.substr();  a = b.substr(1,1)+b.substr(4)+b.substr(); | b=B+EF+ABCDEF  a=E+BCDEF+BEFABCDEF |
|  |  |
| vector |  |  |
|  |  |  |
|  | vector<int> v {1,2}; vector<int> v2 {3};  **v.swap(v2);** cout<< v[0]; | 3 |
| linked list |  |  |
| deque |  |  |
|  |  |  |
| map |  |  |
| multi-map |  |  |
| set |  |  |
| multi-set |  |  |
|  |  |  |
| stack |  |  |
| queue |  |  |
| priority-queue |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Delete or empty

**c.erase(<iter-start>, <iter-end>)**: Iterator following the last removed element. If the iterator pos refers to the last element, the end() iterator is returned.

**c.empty()**. return bool.

|  |  |  |
| --- | --- | --- |
| **action** | **code** | **Note** |
| string | #include <string>  string s = "ABC";  s.erase(s.begin(), s.end()); | remove all characters of this string |
| s.erase(s.begin(), s.begin()+2); | remove first 2 characters |
| s.erase(s.end()-2, s.end()); | remove last 2 characters |
| s.erase(s.begin()+2); | remove the 3rd character |
| string s = ""; cout << s.empty(); | output:1  check if empty |
| vector | #include <vector> |  |
| vector<int> vec{0,1,2,3,4,5,6};  vec.erase(vec.begin()+1); | delete 2nd elements. The vector is 023456 |
| vector<int> vec{0,1,2,3,4,5,6};  vec.erase(vec.begin()+1,vec.end()-2); | delete 2nd-5nd elements. The vector is 056. |
| vector <string> s = {};  cout << s.empty(); |  |
| linked list | #include <list> |  |
| deque | #include <deque> |  |
|  |  |  |
| map | map <char, int> m = {{'a',1}, {'b',2}};  m.erase('a');  cout << m['a']; | output:0 |
| map <char, int> m = {{'a',1}, {'b',2}};  m.erase(m.find('b')); | delete element by iterrator |
| map <char, int> m = {{'a',1}, {'b',2}};      m.erase(m.begin(), m.end()); | delete all elements |
| multi-map |  |  |
| set | #include <set> |  |
| multi-set |  |  |
|  |  |  |
| stack |  |  |
| queue |  |  |
| priority-queue |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |
| --- | --- |
|  |  |
|  |  |
| vector<int> vec{0,1,0,0,5,0};  vector<int>::iterator iter;  for(iter=vec.begin(); iter!=vec.end();){  if(\*iter==0){  // iterator point to the next if succeed  iter=vec.erase(iter);  cout << \*iter<<endl;  }else{  iter++;  }  } | delete all elements with 0. the vector is 1256.  Note: after erase() used, the iterator should be redirect to the beginning or returns to iter. Otherwise iter would be "wild pointer". |
|  |  |
|  |  |
|  |  |
|  |  |

## Size or length

c.size()

c.max\_size()

str.size() and str.length(). Both string::size and string::length are synonyms and return the same value. return number of characters before the character '\0'.

sizeof(str) is equal to sizeof(string). is size of string class

str.capacity(): The capacity of a string reflects how much memory the string allocated to hold its contents.

str.max\_size(): size of string class

**sr.resize(size\_t)**: change size of string object.

|  |  |  |
| --- | --- | --- |
| **action** | **code** | **Note** |
| string | string s="ABC";  cout << s.size() << s.length(); | 33 |
|  | string s="ABC";  cout << sizeof(s) << sizeof(string); | 2424 |
|  | string s="ABC";  cout << s.capacity(); | 15 |
| vector |  |  |
| linked list |  |  |
| deque |  |  |
|  |  |  |
| map |  |  |
| multi-map |  |  |
| set | set<int> s={1,2};    cout << s.size(); | 2 |
|  | set<int> s={1,1};    cout << s.size(); | 1. unique values |
| multi-set |  |  |
|  |  |  |
| stack |  |  |
| queue |  |  |
| priority-queue |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

The sizeof keyword gives the amount of storage, in bytes, associated with a variable or a type (including aggregate types). This keyword returns a value of type size\_t.

In C++, string a is the class instance. sizeof(a) calculate memory size of this class instance, equal to sizeof(string). The output is not affected by contents of the strong. The output is determined by the compiler. could be 4 (size of pointer), or 12,or 24 or 28 or 32

In C, string a is char array. the output below is 5.

char a[]="a\0bc";

printf("%d", sizeof(a));

|  |  |
| --- | --- |
| string fun(string t, string s="x", int r=1){  while(--r)  s +=s;  t +=s;  return s;  }  int main(void){  string name ="a";  cout << fun(name) << name;  return 0;  } | output:xa |
|  |  |
| string a="A";  a.append(a).append(a).append(a);  cout << a.length() << a.size() << endl; | output: 88  AA->AAAA->AAAAAAAA  length() is equal to size() in C++ |
| string a="a\0bc";  cout<< a.length() << endl;  cout<< a.size() << endl;  cout<< a.max\_size() << endl; | output:  1  1  2,147,483,634 |
| string a="a\0bc";  cout<< sizeof(a) << endl;  cout<< sizeof("a\0bc") << endl;  cout<< a.capacity() << endl; | **output:**  **24:== sizeof(string) give the size of class instance std::string no matter what the string is.**  **5: 4 characters + '\0'**  **15** |
| #include <iostream>  #include <cmath>  #include <iomanip>  using namespace std;  string fun(string &t, string s="", int r=2){  while(--r)   s +=s;  t +=s;  return s;  }  int main(void){  string name ="x";  cout << fun(name, name) << name;  return 0;  } | output: xxxxx  pass string value, string reference to function |

## pointer

|  |  |  |
| --- | --- | --- |
| **action** | **code** | **Note** |
| string |  |  |
| vector |  |  |
| linked list |  |  |
| deque |  |  |
|  |  |  |
| map |  |  |
| multi-map |  |  |
| set |  |  |
| multi-set |  |  |
|  |  |  |
| stack |  |  |
| queue |  |  |
| priority-queue |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

### move pointer

c.data(): return pointer of the first element

|  |  |
| --- | --- |
| vector <char> t(5);  char \*p1 = t.data()+2;  char \*p2 = p1+2;  cout<< p2-t.data(); | output:4  vector.data() |
|  |  |
|  |  |
|  |  |

## pass to function

### use string in function

Some equal patterns are showed as below:

output is test

|  |  |
| --- | --- |
| void print(char c[]){cout<<c; }  int main(void) {  print("test");  return 0;  } | pass char array  c pattern |
| **void print(char \*c){cout<<c;}**  int main(void) {  print("test");  return 0;  } | pass char pointer  c pattern |
| **void print(string c){cout<<c; }**  int main(void) {  string s="test"; print(s);  return 0;  } | pass string value  c++ |
| **void print(string& c){cout<<c; }**  int main(void) {  string s="test";print(s);  return 0;  } | pass string reference  c++ |
| **void print(string \*c){cout<<\*c;}**  int main(void) {  string s="test"; string \*p=&s;print(p);  return 0;  } | pass string pointer  c++ |

|  |  |
| --- | --- |
| **void f(char \*p){ cout<< \*p;}**  int main(void) {  **string s="good"; f(&s[0]);**  return 0;  } | output: g  pass char pointer |
| **void f(char \*p){ cout<< p; }**  int main(void) {  **string s="good";f("good");**  return 0;  } | output: good  pass string value. In this case f(s); is wrong.  compare with the above |
| **void f(string \*p){ cout<< \*p; }**  int main(void) {  **string s="good";f(&s);**  return 0;  } | output: good  pass string pointer  compared with the above |
| **string func(string a){return "No." + a;}**  string n = "001"; string user = func(n);  cout << user << endl; | pass value of a string, return value of new string. |
| string f(string s){  return s.substr(0,1)+s.substr(1,1)+s.substr(1,1)+s.substr(0,1);  }  int main(void) {  string \*s = new string("ab");  cout << f("ab") << endl;  cout << f("abba") << endl;  cout << f(s) << endl;  return 0;  } | output:abba  three approaches are equal |
| **void func(string &a){ a = "No." + a; }**  int main(void){  string n = "001"; func(n);  cout << n << endl;  return 0;  } | pass reference. No return is needed.  c++ style is recommended. |
| **string func(char \*p){**  string str = p; str = "No." + str; return str;  }  int main(void){  string n = "001";string user = func(&n[0]);  cout << user << endl;  return 0;  } | pass pointer, return string  not recommended |
| char \*func(char \*p){  string str = p; str = "No." + str; return &str[0];  }  int main(void){  string n = "001"; char \*p = &n[0];  p = func(p); cout << p << endl;  return 0;  } | pass pointer, return pointer  not recommended |
|  |  |
|  |  |

### vector in function

|  |  |
| --- | --- |
|  |  |
| void swap(float \*x, float \*y){  float z = \*x;  x = y;  y = z;  }  int main(void){  vector <float> t = {3.,2.,1.};  swap(&t[0],&t[2]);  cout << t[1];  return 0;  } | pass pointer |
| void swap(float \*x, float \*y){  float \*z = x;  x = y;  y = z;  }  int main(void){  vector <float> t = {3.,2.,1.};  float \*p = &t[0];  swap(p,p+2);  cout << t[0] << t[1] << t[2];  return 0;  } | pass pointer |
| void swap(float &x, float &y){  float z = x;  x = y;  y = z;  }  int main(void){  vector <float> t = {3.,2.,1.};  swap(t[0],t[2]);  cout << t[0] << t[1] << t[2];  return 0;  } | pass reference |
|  |  |
|  |  |
|  |  |
|  |  |

# STL: iterator and algorithm

## basic

**1. declare iterator:**

**#include <iterator>**

**iterator iter;**

**declare algorithm**

**#include <algorithm>**

**2. move iterator**

iter=c.begin():

iter=c.end():

advance(iter, n): incrementally move the iterator with n position. No returns

auto new\_iter = next(iter, n): return the new iterator after advancing the iterator n positions

auto new\_iter = prev(iter, n): return the new iterator after decrementing the iterator n positions

**3. double iterators**

distance(iter\_1, iter\_2)

**4. insert iterator**

inserter(container, iter)

## for-loop or while-loop

### string

Define a string:

#include <string>

string str = "hello";

The below approaches are equal. The output is "h,e,l,l,o,"

|  |  |
| --- | --- |
| for(int i=0; i<str.length(); i++) {  cout << str[i] << ',';  } | for loop |
| char \*p = &str[0];  while(\*p!='\0') {  cout << \*p++ << ',';  } | pointer |
| **for(char& c: str) {**  cout << c << ',';  } | for loop |
| **string::iterator iter;**  **for(iter=str.begin(); iter != str.end(); ++iter) {**  cout << \*iter << ',';  } | iterator+for loop |
| **string::iterator iter=str.begin();**  **for(; iter != str.end(); ++iter) {**          cout << \*iter << ',';      } | compared with the above |
| string::iterator iter**=str.begin()**;  **for(iter; iter != str.end(); advance(iter, 1)) {**          cout << \*iter << ',';      } |  |
| string::iterator iter**=str.begin()**;  **for(iter; iter != str.end(); iter=next(iter)) {**          cout << \*iter << ',';      } |  |
| **string::iterator iter = str.begin();**  **while(iter != str.end()) {**  cout << \*iter << ',';  ++iter;  } | iterator+while loop |
|  |  |

The below approaches are equal. The output is " o,l,l,e,h,"

|  |  |
| --- | --- |
|  |  |
|  |  |
| string::iterator iter=str.end()-1;  **for(iter; iter>=str.begin(); --iter){**          cout << \*iter << ',';      } |  |
| string::iterator iter=str.end()-1;  **for(iter; iter>=str.begin(); advance(iter,-1)){**          cout << \*iter << ',';      } |  |
| string::iterator iter=str.end()-1;  **for(iter; iter>=str.begin(); iter=prev(iter,1)){**          cout << \*iter << ',';      } |  |
| string::iterator iter=str.end()-1;      do{          cout << \*iter << ',';          iter--;      }while(iter != (str.begin()-1) ); |  |
|  |  |
|  |  |

### vector

For example: vector <int> v={1,2,3};

forward loop, output: 123

|  |  |
| --- | --- |
| for(int i=0; i<v.size(); i++) {      cout<< v[i];  } |  |
| vector<int>::iterator iter;  for(iter=v.begin(); iter!=v.end(); iter++) {      cout<< \*iter;  } |  |
| vector<int>::iterator iter;  for(iter=v.begin(); iter!=v.end(); advance(iter,1)) {      cout<< \*iter;  } |  |
| vector<int>::iterator iter;  for(iter=v.begin(); iter!=v.end(); iter=next(iter)) {      cout<< \*iter;  } |  |
| for(int& i:v){      cout << i;  } |  |
| for(int i:v){      cout << i;  } |  |
|  |  |
|  |  |

backward loop, output:321

|  |  |
| --- | --- |
| for(int i=v.size()-1; i>=0; i--) {      cout<< v[i];  } |  |
| vector<int>::iterator iter=v.end()-1;  for(iter; iter>=v.begin(); iter--) {      cout<< \*iter;  } |  |
| vector<int>::iterator iter=v.end()-1;  **for(iter; iter>=v.begin(); advance(iter, -1)) {**      cout<< \*iter;  } |  |
| vector<int>::iterator iter=v.end()-1;  **for(iter; iter>=v.begin(); iter=prev(iter)) {**      cout<< \*iter;  } |  |
| **for(pair<char, int> el: m){**      cout << el.first << el.second;  } |  |
| #include <algorithm>  **for\_each(m.begin(), m.end(), [](pair<char, int> el){**      cout << el.first << el.second;  }); | 1. for\_each( iter\_start, iter\_end, func() ) defined in algorithm.h  2. lambda function |
|  |  |
|  |  |

### map

For example: map<char, int> m = {{'a',1}, {'b',2}, {'c',3}};

|  |  |
| --- | --- |
|  |  |
| **map<char, int>::iterator iter=m.begin();**  **for(iter;iter!=m.end();iter++){**      cout << iter->first << iter->second;  } | iterator |
| **for(pair<char, int> el: m) {**      cout << el.first << el.second;  } | pair defined in utility.h |
| **for\_each(m.begin(), m.end(), \**  **[](pair<char, int> el) {**       cout << el.first << el.second;  }); | for\_each defined in algrothm.h |
| map<char, int> m = {{'a',1}, {'b',2}, {'c',3}};  map<char, int> m2;  transform(m.begin(), m.end(), inserter(m2, m2.begin()), \      [](pair<char, int>& el){          cout << el.first << el.second;          return 4;  }); | ??? |
|  |  |
|  |  |
|  |  |

## lambda expression

lambda functions:

[capture list] (parameter list) -> return type { function body }

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## mathematics

### maximum and minimum

min\_element(iter.begin(), iter.end(), func())

max\_element(iter.begin(), iter.end(), func())

|  |  |  |
| --- | --- | --- |
| **action** | **code** | **Note** |
| string | string str = "hello";  cout << "max=" << \*(max\_element(str.begin(),str.end()));  cout << "min=" << \*(min\_element(str.begin(),str.end())); | output: oe |
| vector | vector<int> v={1,2,3};  cout << "max=" << \*(max\_element(v.begin(),v.end()));  cout << "min=" << \*(min\_element(v.begin(),v.end())); |  |
| linked list |  |  |
| deque |  |  |
|  |  |  |
| map | map<char, int> m = {{'a',1}, {'b',2}, {'c',3}};  typedef pair<char,int> my;  my a = \*max\_element(m.begin(), m.end(), [](const my& x, const my& y){  return x.second < y.second;});  cout << "max=" << a.second;  my b = \*min\_element(m.begin(), m.end(), [](const my& x, const my& y){   return x.second < y.second; });  cout << "min=" << b.second; | output: max=3min=1 |
| multi-map |  |  |
| set |  |  |
| multi-set |  |  |
|  |  |  |
| stack |  |  |
| queue |  |  |
| priority-queue |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

### counting

count(iter.begin(), iter.end(), <value> )

count\_if(iter.begin(), iter.end(), func() )

|  |  |  |
| --- | --- | --- |
| **action** | **code** | **Note** |
| string | int f = count(str.begin(), str.end(), 'l');  cout << f; | output:3  count times of letter 'l'. |
|  | string str = "hello world";  int f = count\_if(str.begin(), str.end(),      [x='l'](const char& c){return c>x;});  cout << f; | output: 4 (2o+1w+1r)  count times of letters greater than 'l'. |
| vector | vector<int> v {1,2,3,4,3,1,0,5};  int f = count(v.begin(), v.end(), 3);  cout << f; | output:2  counts times of integer=3; |
|  | vector<int> v {1,2,3,4,3,1,0,5};  int f = count\_if(v.begin(), v.end(),      [x=3](const int& i){ return i>=x; });  cout << f; | output:4  counts integers which is greater than or equal to 3. |
| linked list |  |  |
| deque |  |  |
|  |  |  |
| map |  |  |
| multi-map |  |  |
| set |  |  |
| multi-set |  |  |
|  |  |  |
| stack |  |  |
| queue |  |  |
| priority-queue |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

### combine/sum/product

operator+ means concatenation for string, and sum for integer or float.

**1.algorithm::accumulate()**

#include <numeric>

accumulate(c.begin(), c.end(), <initial value>).

accumulate(c.begin(), c.end(), <initial object>, <lambda function>).

function of accumulate() like sum() or join()

**2. algorithm::inner\_product()**

inner\_product(v.begin(), v.end(), v2.begin(), <initial value>)

product matching elements of two containers one on one then return sum

|  |  |  |
| --- | --- | --- |
| **action** | **code** | **Note** |
| string | string str = "abc";  string f = accumulate(str.begin(), str.end(), string(""),      [](string ss, const char& c){  return  ss=="" ? (ss+c) : (ss+','+c);});  cout << f; | output:a,b,c |
|  |  |  |
|  |  |  |
| vector | vector<int> v {1,2,3,4,3,1,0,5};  int f = accumulate(v.begin(), v.end(),0);  cout << "sum=" << f; | output: sum=19 |
| vector<int> v {1,2,3,4};  int f= accumulate(v.begin(),v.end(),1,      [](int& s, const int& i){return s\*i; });  cout << f; | output: 24  accumulative product |
| vector<string> vs {"abc", "def"};  string ss = accumulate( vs.begin(), vs.end(), string(""));  cout << ss; | output: abcdef  convert string vector to string. |
| vector<string> vs {"ab", "cd", "ef"};  string f = accumulate(vs.begin(), vs.end(), string("str:"), [](string ss, string& s) { return ss+','+s; }); cout << f; | output: str:,ab,cd,ef  convert string vector to string with separator. |
| vector<int> v {1,2,3,4}; vector<int> v2 {1,2,3,4};  int f = inner\_product(v.begin(), v.end(), v2.begin(), 0);  cout<< f; | output:30  0+1x1+2x2+3x3+4x4=  1+4+9+16=30 |
| vector<int> v {1,2,3,4};  int f = inner\_product(v.begin(), v.end(), v.rbegin(), 0);  cout<< f; | output:20  0+1x4+2x3+3x2+4x1  =4+6+6+4=20 |
| linked list |  |  |
| deque |  |  |
|  |  |  |
| map |  |  |
| multi-map |  |  |
| set |  |  |
| multi-set |  |  |
|  |  |  |
| stack |  |  |
| queue |  |  |
| priority-queue |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## merge and split

### concatenate/combine

algorithm::merge(c1.begin(), c1.end(), c2.begin(), c2.end(),c.begin())

algorithm::inplace\_merge(c1.begin(), c1.end(), c2.begin(), c2.end(),c.begin())

string

accumulate

|  |  |
| --- | --- |
| string s1("GAA"), s2("TTC");  string ss; ss.resize(s1.size()+s2.size());  merge(s1.begin(), s1.end(), s2.begin(), s2.end(), \      ss.begin());  cout << ss; | output:GAATTC |
| string s1("GAA"), s2("TTC");  string ss;  merge(s1.begin(), s1.end(), s2.begin(), s2.end(), \      back\_inserter(ss));  cout << ss; | output:GAATTC |
| string s1("GAA");  **merge(s1.begin(), s1.end(), s1.begin(), s1.end(), \**  **back\_inserter(s1));**  cout << s1; | output:GAAGAAGAA  repeat 3 times |
|  |  |
|  |  |
|  |  |

vector

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## split/slice/partition

### partition

**iter=algorithm::partition(c.begin(), c.end(), <function>)**. split container into two parts by returns of function.

vector

|  |  |
| --- | --- |
| vector<int> v {8,10,4,2,3,1};  vector<int>::iterator iter;  iter = partition(v.begin(), v.end(), \      [](const int& i){          return i%2==1;  });  vector<int> v1(distance(v.begin(), iter));  vector<int> v2(distance(iter, v.end()));  copy(v.begin(), iter, v1.begin());  for(int& i: v1) cout << i;  copy(iter, v.end(), v2.begin());  for(int& i: v2) cout << i; | v1=13, v2=2108  partition and sorting. |
| vector<int> v {8,10,4,2,3,1};  vector<int>::iterator iter;  iter = stable\_partition(v.begin(), v.end(), \      [](const int& i){          return i%2==1;  });  vector<int> v1(distance(v.begin(), iter));  vector<int> v2(distance(iter, v.end()));  copy(v.begin(), iter, v1.begin());  for(int& i: v1) cout << i;  copy(iter, v.end(), v2.begin());  for(int& i: v2) cout << i; | v1=31  v2=81042  partition into two parts and no sorting |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

### split

split string

|  |  |
| --- | --- |
| vector<string> split(const string& str, string sep=","){      static vector<string> items;  **int i = str.find(sep);**      // cout << i << " " << str << endl;      if(i>=0){          items.push\_back(str.substr(0,i));          i += sep.size();          string sub = str.substr(i);          return split(sub,sep);      }      if(!str.empty())          items.push\_back(str);      return items;  } | use methods defined in <string>  string s = "abc,123,a";  vector<string> items = split(s, ",");  for(string& i: items) cout << i << "\t"; |
| vector<string> split(string& str, string sep=","){      static vector<string> items;      string::iterator iter;  **iter = search(str.begin(), str.end(), \**  **sep.begin(), sep.end());**      items.push\_back(string(str.begin(), iter));      if(iter != str.end()){          iter += sep.size();          string sub(iter, str.end());          split(sub, sep);      }      return items;  } | use methods of <algorithm> |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

### subset

**string**

str.substr(<start position>, <length>). length could be missing. Go through '\0' or meeting the length.

|  |  |
| --- | --- |
| string a = "abcd";  cout << a.substr(3); | d |
| string a = "abcdef";  cout << a.substr(0,3); | abc |
| string a = "a\0bcdef";  cout << a.substr(0,3); | a  Note: contains '\0' |

**vector**

|  |  |
| --- | --- |
| int arr[] = {1, 2, 3, 4, 5, 6};  int n = sizeof(arr) / sizeof(arr[0]);  vector<int> v(arr+1, arr + n); | copy array of 2nd-end into a vector |

## Search

Those methods return iterator. return index: "algorithm::distance(c.begin(), iter);"

**1. find single element**

**algorithm::find(c.begin(), c.end(), <const value>)**: value of containers should support operator==. If nothing detected, iterator is NULL.

**algorithm::find\_if(c.begin(), c.end(), <function>)**:

**algorithm::find\_first\_of(c.begin(), c.end(), sub.begin(), sub.end())**: find matching element of one container in another container

**2. find two adjacent elements**

**algorithm::adjacent\_find(c.begin(), c.end(), <const value>)** work on two adjacent elements of container, find two identical elements (must support operator==)

**algorithm::adjacent\_find(c.begin(), c.end(), <function>)>)** work on two adjacent elements of container, find two associated elements

**3. find subset of containers**

**algorithm::find\_end(c.begin(), c.end(), sub.begin(), sub.end())**: find the last container B in container A

**algorithm::search(c.begin(), c.end(), sub.begin(), sub.end())**. find the first container B in container A

**algorithm::search\_n(c.begin(), c.end(), <match value>, <continuous times>)**. find the first occurence of n continuous elements in container

1. search from start. search from end.

2. exact match or partial match

3. multiple matching or regular expression

### search string

str1.find(str2, start) return index if succeeds or unpredictable value

str1.find\_first\_of(str2)

|  |  |
| --- | --- |
| string a = "abcde";  string b = "de";  **cout << a.find(b);** | 3 |
| string a = "abcdebc";  string b = "bc";  **cout << a.find(b, 4);** | 5 |
| string a = "abcdebc";  string b = "dd";  **cout << a.find(b);** | output: unpredictable value |
| string s = "ATCGAT";  **string::iterator iter;**  **iter = find(s.begin(), s.end(), 'G');**  **cout << \*iter;** | ouput:G  the 3rd argument must be same data type of elements of the container. |
| string s = "ATXGCNGTI";  string sub = "INX";  **int p = s.find\_first\_of(sub);**  cout << p; | output:2 |
| string s = "AcTCGAT";  string::iterator iter;  **iter = find\_if(s.begin(), s.end(), \**  **[](const char& c){ return c=='c' || c=='C';});**  cout << \*iter; | output:c |
| string s = "ABC3DG40";  string digits="0123456789";  string::iterator iter;  **iter = find\_if(s.begin(), s.end(), \**  **[&digits](const char& c){**  **return digits.find(c)!=string::npos; });**  cout << \*iter; | output: 3  find the digits: find characters which could be one of characters. |
| string s = "AGCCTCATTGCCGAT";  string sub = "GCC";  string::iterator iter;  **iter = find\_end(s.begin(), s.end(), \**  **sub.begin(), sub.end());**  cout << distance(s.begin(), iter); | output: 9  find matched sub string. two currency of GCC at 1 and 9. |
| string s = "AGCCTCATTGCCGAT";  string sub = "GCC";  string::iterator iter;  **iter = search(s.begin(), s.end(), \**  **sub.begin(), sub.end());**  cout << distance(s.begin(), iter); | output: 1  find matched sub string. two currency of GCC at 1 and 9. |
| string s = "ATXGCNGTI";  string sub = "INX";  string::iterator iter;  **iter = find\_first\_of(s.begin(), s.end(), \**  **sub.begin(), sub.end());**  cout << \*iter; | output:X  return iterator if any character "INX" is matched. |
| string s = "ATTTGCCGTI";  string::iterator iter;  **iter = adjacent\_find(s.begin(), s.end());**  cout << \*iter; | output:T  find two continuous characters like "TTT" or "CC" |
| string s = "ATTTGCCGTI";  string::iterator iter;  **iter = adjacent\_find(s.begin(), s.end(), \**  **[](const char& a, const char& b){**  **return a=='G'&&b=='C';**  **});**  cout << \*iter;  cout << distance(s.begin(), iter); | output: G4  find "GC" in string. |
| string s = "AGCCCTCATTGCCGAT";  string sub = "CC";  string::iterator iter;  **iter = search\_n(s.begin(), s.end(), 2, 'C');**  cout << distance(s.begin(), iter); | output:2 |
|  |  |
|  |  |

### search vector

|  |  |
| --- | --- |
| vector<int> v {1,2,3,2,4};  vector<int>::iterator iter;  iter = find(v.begin(), v.end(), 2);  cout << distance(v.begin(), iter); | output:1  2 is at 1 and 3. return the first occurrence. |
| vector<int> v {1,3,10,2,4};  vector<int>::iterator iter;  iter = find\_if(v.begin(), v.end(), \      [](const int& i){ return i%2==0;});  cout << distance(v.begin(), iter); | output:2  find first even number |
| vector<int> v {3,10,20,1,4,2};  vector<int> v2 {1,2};  vector<int>::iterator iter;  iter = find\_first\_of(v.begin(), v.end(), \  v2.begin(), v2.end());  cout << distance(v.begin(), iter); | output:3 |
| vector<int> v {3,1,2,1,4,1,2};  vector<int> v2 {1,2};  vector<int>::iterator iter;  iter = find\_end(v.begin(), v.end(), v2.begin(), v2.end());  cout << distance(v.begin(), iter); | output:5 |
| vector<int> v {3,1,2,1,4,1,2};  vector<int> v2 {1,2};  vector<int>::iterator iter;  iter = search(v.begin(), v.end(), v2.begin(), v2.end());  cout << distance(v.begin(), iter); | output:1 |
| vector<int> v {3,1,1,1,4,1,2};  vector<int>::iterator iter;  iter = search\_n(v.begin(), v.end(), 3, 1);  cout << distance(v.begin(), iter); | output:1  find {1,1,1} |
| vector<int> v {3,1,3,3,4,1,2};  vector<int>::iterator iter;  iter = adjacent\_find(v.begin(), v.end());  cout << distance(v.begin(), iter); | output:2  find first occurrence of two continuous identical elements. Here is {3,3} |
| vector<int> v {0,1,3,6,4,1,2};  vector<int>::iterator iter;  iter = adjacent\_find(v.begin(), v.end(),\      [](const int& a, const int& b){          return a!=0 && b/a==2;  });  cout << distance(v.begin(), iter); | output:2  find first occurrence of two continuous elements with specific associations. Here is {3,6} |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

### duplicates

algorithm::unique(c.begin(), c.end())

algorithm::unique\_copy(c.begin(), c.end(), back\_inserter(c2))

|  |  |
| --- | --- |
| vector<int> v {1,2,2,3,4};  vector<int>::iterator iter;  iter=unique(v.begin(), v.end());  for(int& i: v) cout << i; | output:12344 |
| vector<int> v {1,1,2,2,3,3,4,2};  vector<int>::iterator iter;  iter=unique(v.begin(), v.end());  v.erase(iter, v.end());  for(int& i: v) cout << i; | output:12342  first sort then remove duplicates |
| vector<int> v {1,1,2,2,3,3,4,2};  vector<int> v2;  sort(v.begin(), v.end());  unique\_copy(v.begin(), v.end(), back\_inserter(v2));  for(int& i: v2) cout << i; | output:1234  first sort, then remove duplicates, copy unique to new container |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

### Compare

1. overload operator==, operator<, operator>

2. object methods:

string::str.compare() for string

algorithm::max()

algorithm::min()

3.

**algorithm::equal(v1.begin(), v1.end(), v2.begin())**. compare one by one. Return true if v1 is prefix of v2 or v1==v2.

**algorithm::mismatch(v1.begin(), v1.end(), v2.begin()).** compare one by one. Return pair type of iterator which the first occurrence of mismatching.

algorithm::lexicographical\_compare(v1.begin(), v1.end(), v2.begin(), v2.end()). check if v1 is prefix of v2 and v1 is shorter than v2.

**compare string**

1. overload operator+

str1==str2, if equal

str1<str2 if less than by ASCII code

str1>str2 if greater than by ASCII code

2. object methods:

str1.compare(str2): equal return 0, less than return -1, greater than return 1

|  |  |
| --- | --- |
| string a = "ad"; string b= "bc";  if(a<=b) cout<< a; | output: ad  compare ASCII code of characters one by one. |
| string a = "ad"; string b= "bc";  if(a.compare(b)==-1) cout<< a; | str1.compare(str2)>0 str1>str2  str1.compare(str2)==0 str1==str2  str1.compare(str2)<0 str1<str2 |
| string a = "abcde"; string b = "de";  cout << a.compare(3, a.size()-1, b) << endl; | output: 0 |
| string a = "abcde"; string b = "de";  cout << a.compare(3, 10, b) << endl; | output: 0  the 2n argument the end could be longer than the length of the string |
| string s("GAATTC");  string ss("GAATTC");  bool res = equal(s.begin(), s.end(), ss.begin());  cout << res; | output:1 |
| string s("GAATT");  string ss("GAATTC");  bool res = equal(s.begin(), s.end(), ss.begin());  cout << res; | ouput:1 |
| string s("GAATTCG");  string ss("GAATTC");  bool res = equal(s.begin(), s.end(), ss.begin());  cout << res; | output:0 |
| string s("GAATTCCG");  string ss("GAATTC");  bool res = equal(s.begin(), s.end(), ss.begin());  cout << res; | ouput:0 |
| string s("GAATT");  string ss("GAATTC");  **pair<string::iterator, string::iterator> mypair;**  **mypair = mismatch(s.begin(), s.end(), ss.begin());**  cout << distance(s.begin(), mypair.first);  cout << distance(ss.begin(), mypair.second); | output:55  if s=ss="GAATTC", return 66 |
| string s("GAATTC");  string ss("GAATTC");  bool res = lexicographical\_compare( \  s.begin(), s.end(), ss.begin(), ss.end());  cout << res; | output:1 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**vector**

|  |  |
| --- | --- |
| vector<int> v {1,2};  vector<int> v2 {1,2};  bool f = equal(v.begin(), v.end(), v2.begin());  cout << f; | output: 1 |
| vector<int> v {1,2};  vector<int> v2 {1,2, 3};  bool f = equal(v.begin(), v.end(), v2.begin());  cout << f; | output:1  v is prefix of v2. |
| vector<int> v {1,2,3};  vector<int> v2 {1,2};  bool f = equal(v.begin(), v.end(), v2.begin());  cout << f; | ouput:0 |
| vector<int> v {1,2};  vector<int> v2 {1,2};  bool f = lexicographical\_compare( \      v.begin(), v.end(), v2.begin(), v2.end());  cout << f; | outpu:0 |
| vector<int> v {1,2};  vector<int> v2 {1,2,3};  bool f = lexicographical\_compare( \      v.begin(), v.end(), v2.begin(), v2.end());  cout << f; | output:1  v is shorter than v2. |
| vector<int> v {1,2,3};  vector<int> v2 {1,2};  pair<vector<int>::iterator, vector<int>::iterator> mypair;  mypair = mismatch( v.begin(), v.end(), v2.begin(), v2.end());  cout << distance(v.begin(), mypair.first);  cout << distance(v2.begin(), mypair.second); | output:22 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**integer**

|  |  |
| --- | --- |
| cout<< max(12, 3);  cout<< min(12, 3); | output:123 |
| cout<< max('A', 'Z');  cout<< min('1', '2'); | output:Z1 |

### search bound

algorithm::lower\_bound(c.begin(), c.end(), <bound value>).

algorithm::upper\_bound(c.begin(), c.end(), <bound value>).

algorithm::equal\_range(c.begin(), c.end(), <bound value>)

|  |  |
| --- | --- |
| vector<int> v {1,2,3,4,5,6,7,8,9};  vector<int>::iterator iter;  **iter = lower\_bound(v.begin(),v.end(), 6);**  cout << \*iter << distance(v.begin(), iter); | output:65  index of value=6 is 5. |
| vector<int> v {1,2,3,4,5,6,7,8,9};  vector<int>::iterator iter;  **iter = upper\_bound(v.begin(),v.end(), 6);**  cout << \*iter << distance(iter, v.end()); | output:73 |
| vector<int> v {1,2,3,4,5,6,7,8,9};  pair<vector<int>::iterator, vector<int>::iterator> iter;  **iter = equal\_range(v.begin(),v.end(), 6);**  cout << \*iter.first << \*iter.second; | output:67 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## transform

**algorithm::transform() is like apply() in R**

for the objects of string or vector

transform(v.begin(), v.end(), v.begin(), <lambda function>)

transform(v.begin(), v.end(), back\_inserter(v2), <lambda function>)

**string**

|  |  |
| --- | --- |
| string s = "abc";  transform(s.begin(), s.end(), s.begin(), ::toupper);  cout << s; | ABC  1. use ::toupper defined in cctype |
| string s = "ABC"; string ss;  transform(s.begin(), s.end(), back\_inserter(ss), ::tolower);  cout << ss; |  |
| string s = "ATGC"; string ss;  transform(s.rbegin(), s.rend(), back\_inserter(ss), ::toupper);  cout << ss; | reverse |
| map<char, char> nt {{'A','T'},{'T','A'},{'G','C'},{'C','G'}};  string s = "ATGC"; string ss;  **transform(s.rbegin(), s.rend(), back\_inserter(ss),\**  **[&nt](const char& c){return nt[c];});**  cout << ss; | GCAT  reverse\_complementary |
|  |  |
|  |  |
|  |  |
|  |  |

vector

|  |  |
| --- | --- |
| vector<int> v {1,2,3,4};  transform(v.begin(), v.end(), v.begin(),  [](const int& i){ return i\*i; });  for(int& i : v) cout << i; | output:14916 |
| vector<int> v {1,2,3,4};  vector<int> v2;  transform(v.begin(), v.end(), back\_inserter(v2),  [](const int& i){ return i\*i; });  for(int& i : v2) cout << i; | output:14916 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## random

random\_shuffle(c.begin(),c.end()). shuffling

|  |  |
| --- | --- |
| vector<int> v {1,2,3,4,5,6,7,8,9,0};  random\_shuffle(v.begin(),v.end());  for(int& i: v) cout << i; | 9203168457 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## Sort

algorithm::sort(c.rbegin(), c.rend())

algorithm::sort(c.rbegin(), c.rend())

algorithm::sort(c.begin(), c.end(), <lambda function>)

algorithm::partition

|  |  |  |
| --- | --- | --- |
| **action** | **code** | **Note** |
| string | string str = "0asdfHXhjk 24l.";  sort(str.begin(), str.end());  cout << str; | output: .024HXadfhjkls |
| vector | vector<int> v {10,-2,0,13,4};  **sort(v.begin(), v.end());**  for(int& i: v) cout << i << ','; | output: -2,0,4,10,13, |
| vector<int> v {10,-2,0,13,4};  **sort(v.rbegin(), v.rend());**  for(int& i: v) cout << i << ','; | output: 13,10,4,0,-2, |
| vector<string> vs {"0ab", " a", "zcd", "ef"};  **sort(vs.begin(), vs.end());**  for(string& i: vs) cout << i << ','; | output: a,0ab,ef,zcd, |
| vector<string> vs {"chr10", "chr3", "chr20", "chr19", \  "chr1", "chrX", "chrY"};  **sort(vs.begin(), vs.end(),**  **[](const string& a, const string& b){**      try{          int x = stoi(a.substr(3));          int y = stoi(b.substr(3));          return x < y;      }catch(...){ return a < b; }  });  for(string& i: vs) cout << i << ','; | output: chr1,chr3,chr10,chr19,  chr20,chrX,chrY,  sort chromsome |
| linked list |  |  |
| deque |  |  |
|  |  |  |
| map |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| multi-map |  |  |
|  |  |  |
| set |  |  |
| multi-set |  |  |
|  |  |  |
| stack |  |  |
| queue |  |  |
| priority-queue |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Update

### Copy

1. overload operator=

2. iterator: c(c.begin(), c.end())

3. string::str.copy(str2, <length>, <position>)

4. STL methods

algorithm::copy(c.begin(), c.end(), back\_inserter(c2))

algorithm::copy\_backward(c.begin(), c.end(), back\_inserter(c2))

algorithm::reverse\_copy(c.begin(), c.end(), back\_inserter(c2))

|  |  |  |
| --- | --- | --- |
| **action** | **code** | **Note** |
| string | string s1="A", s2=s1; cout << s1 << s2;  s2="B"; cout << s1 << s2; | output: AAAB |
| string b(a.begin(), a.end()); |  |
| string str = "abc";  char ss[4]; str.copy(ss, str.size());  cout << ss; | str.copy() works for char array rather than string type |
| string str = "abcdef";  char ss[5]; str.copy(ss, 4, 2);  cout << ss; | output: cdef |
| string str = "abc";  string ss;  **copy(str.begin(), str.end(), back\_inserter(ss));**  cout << ss; |  |
| string str = "abc"; string ss;  **for\_each(str.begin(), str.end(), \**  **[&](const char& c){ ss.push\_back(c);});**  cout << ss; |  |
|  | string s("GAATTC"); string ss;  reverse\_copy(s.begin(), s.end(), back\_inserter(ss));  cout << ss; |  |
| vector | vector <int> v={1,2,3};  vector <int> v2=v;  v[0]=4; cout << v2[0]; | v2:123, v:423 |
| vector <int> v1={1,2,3};  vector <int> v2(v1.begin(), v1.end()); |  |
| vector<int> v {1,2,3,4};  vector<int> v2;  **copy(v.begin(), v.end(), back\_inserter(v2));**  for(int& i: v2) cout << i << ','; |  |
| vector<int> v {1,2,3,4};  vector<int> v2;  **for\_each(v.begin(), v.end(),\**  **[&](const int& i){ return v2.push\_back(i);});**  for(int& i: v2) cout << i << ','; |  |
| linked list |  |  |
| deque |  |  |
|  |  |  |
| map | map <char, int> m = {{'a',1}};  map <char, int> m2 = m; |  |
| map <char, int> m = {{'a',1}};  map <char, int> m2(m.begin(), m.end()); |  |
|  |  |  |
| multi-map |  |  |
| set |  |  |
| multi-set |  |  |
|  |  |  |
| stack |  |  |
| queue |  |  |
| priority-queue |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Note: . size\_t copy(char \*p, size\_t length, size\_t pos=0). strcopy() only works for C, where string is char array.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

### Reverse

c(c.rbegin(), c.rend())

algorithm::transform(c.rbegin(), c.rend(), back\_inserter(c2), <lambda func>)

algorithm::accumulate(c.rbegin(), c.rend(), <initial value>, <lambda func>)

algorithm::for\_each(c.rbegin(), c.rend(), <lambda func>)

algorithm::copy(c.rbegin(), c.rend(), c2.begin())

algorithm::copy\_backward(c.begin(), c.end(), c2.end())

|  |  |  |
| --- | --- | --- |
| **action** | **code** | **Note** |
| string | string str = "ATCGAT";  string ss = string(str.rbegin(), str.rend()); cout << ss; | output: TAGCTA |
| string str = "ATCGAT"; string ss;  **transform(str.rbegin(), str.rend(), back\_inserter(ss),**      [](const char& c){ return c; }); cout << ss; |  |
| string str = "ATCGAT";  **string f = accumulate(str.rbegin(), str.rend(), string(""),**  **[](string f, const char& c){ return  f+c;});**  cout << f; |  |
| string str = "ATCGAT";  string ss;  **for\_each(str.rbegin(), str.rend(), \**  **[&](const char& c){ ss.push\_back(c); });**  cout << ss; |  |
|  | string str = "ATCGAT";  string ss; ss.resize(str.size());  **copy(str.rbegin(), str.rend(), ss.begin());**  cout << ss; |  |
|  | string str = "ATCGAT";  string ss;ss.resize(str.size());  **copy\_backward(str.rbegin(), str.rend(), ss.end());**  cout << ss; |  |
|  | string s("GAATTC"); string ss;  **reverse\_copy(s.begin(), s.end(), back\_inserter(ss));**  cout << ss; | reverse\_copy() |
| vector | vector<int> v {1,2,3,4};  **vector<int> v2(v.rbegin(),v.rend());**  for(int& i: v2) cout << i << ','; |  |
|  | vector<int> v {1,2,3,4};  vector<int> v2;  **transform(v.rbegin(),v.rend(), back\_inserter(v2),**  **[](const int& i){ return i; });**  for(int& i: v2) cout << i << ','; |  |
|  | vector<int> v {1,2,3,4};  vector<int> v2;  **for\_each(v.rbegin(),v.rend(), [&](int& i){ v2.push\_back(i); });**  for(int& i: v2) cout << i << ','; |  |
|  |  |  |
| linked list |  |  |
| deque |  |  |
|  |  |  |
| map |  |  |
| multi-map |  |  |
| set |  |  |
| multi-set |  |  |
|  |  |  |
| stack |  |  |
| queue |  |  |
| priority-queue |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

### swap/rotate

algorithm::swap(<index 1>, <index 2>)

algorithm::swap\_ranges(c.begin(), c.begin()+2, c2.begin())

algorithm::iter\_swap(&<index 1>, &<index 2>)

algorithm::rotate(c.begin(), c.begin()+x, c.begin()+c.size())

|  |  |  |
| --- | --- | --- |
| **action** | **code** | **Note** |
| string | string s("GAATTC");  swap(s[0],s[5]);  cout << s; | output: CAATTG |
| string s("GAATTC");  iter\_swap(&s[0], &s[5]);  cout << s; |  |
| string s("GAATTC");  swap\_ranges(s.begin(), s.begin()+3, s.begin()+3);  cout << s; | output:TTCGAA |
| string s("GAATTC");  string ss("GGG");  swap\_ranges(s.begin(), s.begin()+3, ss.begin());  cout << s; | output:GGGTTC |
| string s("GAATTC");  string ss("GGG");  swap\_ranges(s.rbegin(), s.rbegin()+3, ss.begin());  cout << s; | output:GAAGGG |
| string s("GAATTC");  rotate(s.begin(), s.begin()+3, s.begin()+6);  cout << s; | output:TTCGAA  swap GAA with TTC |
| string s("GAATTC");  rotate(s.begin(), s.begin()+2, s.begin()+6);  cout << s; | output: ATTCGA  swap GA with ATTC |
| vector | vector<int> v {1,2,3,4,5,6};  **swap(v[2], v[3]);**  for(int& i: v) cout << i; | output:124356 |
| vector<int> v {1,2,3,4,5,6};  **iter\_swap(&v[2], &v[3]);**  for(int& i: v) cout << i; | output:124356 |
| vector<int> v {1,2,3,4,5,6};  **swap\_ranges(v.begin(), v.begin()+2, v.begin()+2);**  for(int& i: v) cout << i; | 341256  swap first two with 3-4th elements |
| vector<int> v {1,2,3,4,5,6};  vector<int> v2 {7,8};  **swap\_ranges(v.begin()+1, v.begin()+3, v2.begin());**  for(int& i: v) cout << i; | 178456 |
| vector<int> v {1,2,3,4,5,6};  rotate(v.begin(), v.begin()+3, \      v.begin()+v.size());  for(int& i: v) cout << i; | 456123 |
| linked list |  |  |
| deque |  |  |
|  |  |  |
| map |  |  |
| multi-map |  |  |
| set |  |  |
| multi-set |  |  |
|  |  |  |
| stack |  |  |
| queue |  |  |
| priority-queue |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

### replace

algorithm::replace(c.being(),c.end(), <initial value>, <replace value>)

algorithm::generate(c.being(),c.end(),<function>)

string

|  |  |
| --- | --- |
| string s = "ATCGTCAT";  **replace(s.begin(),s.end(), 'T', 'C');** | ACCGCCAC |
| string s;  s.resize(5);  **fill(s.begin(), s.end(),'N');** | NNNNN |
| string s; s.resize(5);  **fill\_n(s.begin(), 5,'N');** | NNNNN |
| string s;s.resize(5);  **generate(s.begin(), s.end(), [](){return 'A';});** | AAAAA |
| string s;s.resize(5);  **generate\_n(s.begin(), 2, [](){return 'A';});**  cout << s; | AA |
|  |  |
|  |  |

vector

|  |  |
| --- | --- |
| vector<int> v {1,2,2,3,3,4};  **replace(v.begin(),v.end(),2,5);**  for(int& i: v) cout << i; | 155334 |
| vector<int> v(5);  fill(v.begin(),v.end(),2); | 22222 |
| vector<int> v(5);  fill\_n(v.begin(), 2, 5); | 55000 |
| vector<int> v(5);  generate(v.begin(), v.end(), [](){      static int i=0; return ++i;  });  for(int& i: v) cout << i; | 12345 |
| vector<int> v(5);  generate(v.begin(), v.end(), [](){      static int i=0; return i += 2;  });  for(int& i: v) cout << i; | 246810 |
| vector<int> v(5);  generate(v.begin(), v.end(), [](){return 2;});  for(int& i: v) cout << i; | 22222 |
| vector<int> v(5);  generate\_n(v.begin(), 2, [](){return 2;});  for(int& i: v) cout << i; | 22000 |
|  |  |
|  |  |

### delete

**iter = algorithm::remove(c.begin(), c.end(), <delete value>)**. don't really delete them, but move delete elements to the end of container.

**iter = algorithm::remove\_if(c.begin(), c.end(), <function>)**.

**algorithm::remove\_copy(c.begin(), c.end(), c2.begin(), <delete value>)**.

**algorithm::remove\_copy\_if(c.begin(), c.end(), c2.begin(), <function>)**.

string

|  |  |
| --- | --- |
| string s = "ATATA";  remove(s.begin(), s.end(),'A');  cout << s; | TTATA |
| string s = "ATATA";  string::iterator iter;  **iter = remove(s.begin(), s.end(), 'A');**  string ss;  ss.resize(distance(s.begin(), iter));  copy(s.begin(), iter, ss.begin());  cout << ss; | TT  remove all 'A'. Here is complete code compared with the above |
| string s = "ATGCNXCI";  string nt="ATGC";  string::iterator iter;  **iter = remove\_if(s.begin(), s.end(), \**  **[&nt](const char& c){**  **return nt.find(c)==nt.npos;**  **});**  string ss;  ss.resize(distance(s.begin(), iter));  copy(s.begin(), iter, ss.begin());  cout << ss; | ATGCC  remove characters except "ATGC" |
| string s = "ATGCNXCI";  string nt="ATGC";  string ss;  ss.resize(s.size());  **remove\_copy\_if(s.begin(), s.end(), ss.begin(),**  **[&nt](const char& c){**  **return nt.find(c)==nt.npos;**  **});**  cout << ss; | ATGCC |
| string s = "ATGTATA";  string ss; ss.resize(s.size());  **remove\_copy(s.begin(), s.end(), ss.begin(), 'A');**  cout << ss; | TGTT |
|  |  |
|  |  |
|  |  |

vector

|  |  |
| --- | --- |
| vector<int> v {1,2,3,4,1,1};  vector<int>::iterator iter;  **iter = remove(v.begin(), v.end(), 1);**  vector<int> v2(distance(v.begin(), iter));  copy(v.begin(), iter, v2.begin());  for(int& i: v2) cout << i; | 234 |
| vector<int> v {1,2,3,4,6,20};  vector<int>::iterator iter;  **iter = remove\_if(v.begin(), v.end(), \**  **[](const int i){**  **return i%2==0;**  **});**  vector<int> v2(distance(v.begin(), iter));  copy(v.begin(), iter, v2.begin());  for(int& i: v2) cout << i; | 13 |
| vector<int> v {1,2,3,4,6,20};  vector<int> v2(v.size());  **remove\_copy\_if(v.begin(), v.end(), v2.begin(),\**  **[](const int i){        return i%2==0;    });**  for(int& i: v2) cout << i; | 130000 |
| vector<int> v {1,2,3,1};  vector<int> v2(v.size());  **remove\_copy(v.begin(), v.end(), v2.begin(), 1);**  for(int& i: v2) cout << i; | 2300 |
|  |  |
|  |  |
|  |  |
|  |  |

### Convert

convert string to integer

1. string::stoi(str)

2. sstream::istringstream(str)

3. algorithm::transform()

convert string to integer

|  |  |
| --- | --- |
| cout << stoi("12");  cout << stoi("34abc12");  cout << stoi("5\n6"); | output:12345  wrong cout << stoi("");  wrong cout << stoi("abc12"); |
| int i=0;  istringstream("12")>>i; cout << i;  istringstream("34abc12")>>i;cout << i;  istringstream("5\n6")>>i;cout << i;  istringstream("")>>i;cout << i; // nothing assigned  istringstream("abc12")>>i;cout << i; // return 0 | output:1234550 |
|  |  |
|  |  |
|  |  |

convert vector

|  |  |
| --- | --- |
| int arr[] = {1, 2, 3};  int n = sizeof(arr) / sizeof(arr[0]);  **vector<int> v(arr, arr + n);** | convert array to vector |
| int arr[] = {1, 2, 3, 4, 5, 6};  int n = sizeof(arr) / sizeof(arr[0]);  **vector<int> v(arr+1, arr + n);** | copy array of 2nd-end into a vector |
| vector<int> vect1{ 10, 20, 30 };  vector<int> vect2(vect1.begin(), vect1.end()); | copy vector |
| vector<string> v {"a","b", "c"};  string ss= ""; string sep=",";  **ss=accumulate(v.begin(),v.end(), string(""),\**  **[&sep](string ss, string& s){**  **return ss=="" ? s : ss+sep+s;**  **});**  cout << ss; | output:a,b,c  join string vector to string. |
| vector<int> v {1,2,3,4};  string ss= ""; string sep=",";  **ss=accumulate(v.begin(),v.end(), string(""),\**  **[&sep](string ss, int& i){**  **string s = to\_string(i);**  **return ss=="" ? s : ss+sep+s;**  **});**  cout << ss; | output:1,2,3,4 |
|  |  |

# stream

functions

|  |  |
| --- | --- |
| string line\_of\_types;  getline(cin, line\_of\_types);  cout << line\_of\_types << endl; | getline() |
|  |  |
|  |  |

## <iostream>

### cout and cin

for cout, the stream is stack type- "last one, first out". the order of printing is left to right, but the executed order is from right to left.

|  |  |
| --- | --- |
| class A{  int \*val;  public:  **A() {val = new int; \*val=0;}**  int get(){return ++(\*val);}  };  int main(){  **A a, b=a;**  cout << a.get() << b.get();  return 0;  } | output: 21  Note: objects a and b use same constructor. the attribute the pointer sharing this same value. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## text file: <fstream>

<fstream>

|  |  |
| --- | --- |
| #include <iostream> // cerr, cout, endl, ios::in  #include <fstream> //ifstream  #include <cstdlib>  // exit  using namespace std;  int main(void) {  **ifstream infile("F:\\cpp\\read.txt", ios::in);**      if( !infile ){          cerr << "file can't be opened" << endl;          exit(1);      }      string line;      while(infile >> line) {          cout << line << endl;      }      return 0;  } | print a text file and print them |
| #include <iostream> // cerr, cout, endl, ios::out  #include <fstream> //ofstream  #include <cstdlib>  // exit  #include <vector>  #include <string>  using namespace std;  int main(void) {      ofstream outfile("F:\\cpp\\write.txt", ios::out);      if(!outfile){          cerr << "file can't be opened" << endl;          exit(1);      }      vector<string> lines {"abc", "123"};      for(string line: lines) {          outfile << line << endl;          // cout<< line;      }      return 1;  } | write strings in vector into a text file |
| //split lines into columns  void split(vector<string>& items, string& str, string sep=","){      string::iterator iter;      iter = search(str.begin(), str.end(), \          sep.begin(), sep.end());      items.push\_back(string(str.begin(), iter));      if(iter != str.end()){          iter += sep.size();          string sub(iter, str.end());          split(items, sub, sep);      }  }  int main(){      string file\_path = "F:\\cpp\\tab.txt";      string sep = "\t";      //write to text file separated by tab space      ofstream outfile; //file object      outfile.open(file\_path, ios::out);      if(!outfile){          cerr << "file can't be opened!" << endl;          exit(1);      }      map<string, int> st {{"John",34},{"Mary",12},{"Cary",77}};      outfile << "Name" << sep << "Age" << endl;      for(pair<string, int> el: st){          outfile << el.first << sep << el.second << endl;      }      outfile.close();      // read from text file      ifstream infile;      infile.open(file\_path, ios::in);      string line;      while(getline(infile, line)){          // cout << ">>" <<  line << endl;          vector<string> items;          split(items, line, sep);          cout << items[0] << "\t" << items[1] <<endl;      }      infile.close();      return 1;  } | write/read text which stores table data separated by table space |
| string ss="he";      ostringstream oss;    oss.str(ss);     oss.put(65);      ss = oss.str();    cout << ss;  oss.str("") // clear memory | output:Ae  oss.put() insert character from the front |
|  |  |
|  |  |
|  |  |
|  |  |

## <sstream>

#include <sstream>

ostringstream os;

istringstream is(str); read substring from string separated by whitespace or tab space

stringstream ss(str); bind stream with string

|  |  |
| --- | --- |
| string s = "abc 123";  **istringstream is(s);**  do{      string sub;  **is >> sub;**      cout << sub << endl;  **}while(!is.eof());** | output:  abc  123 |
| string s = "abc 123";  istringstream is(s);  string sub;  **while(is >> sub)**      cout << sub << endl; | output:  abc  123  same as the above |
| string s = "abc 123 def";  **stringstream ss(s);**  vector<string> items;  do{      string sub;      ss >> sub;      items.push\_back(sub);  }while(!ss.eof());  for(string& i: items) cout << i << "\t"; | abc 123 def  "stringstream" could split string |
| string s = "abc 123 def";  stringstream ss(s);  vector<string> items;  string sub;  while(ss >> sub)      items.push\_back(sub);  for(string& i: items) cout << i << "\t"; | abc 123 def  same as the above |
| template<class T>  string toString(const T& t){      ostringstream oss;      oss << t;      return oss.str();  }  int main(){      string s = toString(12.3f);      cout << s;      return 1;  } | convert integer, char, bool, float or double to string. |
|  |  |
|  |  |
|  |  |
|  |  |

# exceptions

logic error:

run time error:

|  |
| --- |
| #include <exception>  try{  …..  }  catch (string &problem) {  cout << problem << endl;  }  catch(…){ } |
| int problem -> cout << problem  string &problem -> cout << problem  class &exc -> cout << exc.msg |
|  |
|  |
|  |

## throw exceptions

### throw various data type:

#include <exception>

throw exception() -> catch(exception &e) -> e.what()

|  |  |
| --- | --- |
| int main(void) {  try { throw exception(); }  catch(exception &e){ cout << e.what(); }  return 0;  } | output: std::exception  Note: "throw exception("xxx");" is wrong |
| void f(void){**throw exception();** }  int main(void) {  int a=4;  **try { f(); }**  **catch (exception &e){ a--;}**  **catch(...){ a++;}**  cout << a << endl;  return 0;  } | output: 3  "throw exception();" |

throw instance of a class

throw cls() -> catch(cls &c) -> cout<< c.msg

|  |  |
| --- | --- |
| class Class {  public:string msg;  Class(string txt): msg(txt) { }  };  **void function(int i) { throw Class("object"); }**  int main(void) {  try {function(1);}  catch(Class &exc) {cout << exc.msg;}  return 0;  } | output: object  "throw Class("object");" |
| class A{  public: static int a;  A(){ a++; }  };  int A::a=0;  **void f(void){ A a; throw A(); }**  int main(void) {  A a;  **try { f(); }**  **catch(...){ }**  cout << A::a << endl;  return 0;  } | output: 3  catch(…) other exception  "throw A();" |

throw pointers

E e; -> throw &e -> catch(E\*)

|  |  |
| --- | --- |
| class E{  public: string msg;  E() {msg="bad";}  };  void f(int i){  E e;  switch(i){  case 0: throw e;  case 1: throw &e;  }  cout <<0;  };  int main(void) {  try { f(0); }  catch(E e){ cout << "msg=" << e.msg; }  return 0;  } | output: msg=bad |
| class E{  public:  string msg;  E() {msg="bad";}  };  void f(int i){  E e;  switch(i){  case 0: throw e;  case 1: throw &e;  }  cout <<0;  };  int main(void) {  try { f(1); }  catch(E\* e){ cout << "msg=" << e->msg; }  return 0;  } |  |
| class E{};  void f(int i){  E e;  switch(i){  case 0: throw e;  case 1: throw &e;  }  cout <<0;  };  int main(void) {  try { f(1); }  catch(void \*){ cout << 2; }  catch(E\*){ cout <<1; }  return 0;  } | output:2  catch(void \*) will override catch(E\*) |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

throw digits

throw 1 -> catch(int &i)

|  |  |
| --- | --- |
| int i=1;  **void f(void){ throw 1;i++; }**  void g(void){  i++;  **try{ f(); }**  **catch( int &i){ throw ++i; }**  }  int main(void) {  try { g(); i++; }  catch(...){ i++; }  cout << i << endl;  return 0;  } | output:3  throw <int>; |
| class X{  public:  X() { cout << 0; };  ~X() { cout << 2; };  };  int main(void) {  try {  X \*x = new X();  **throw true;**  delete x;  }  **catch(bool i){ cout << i; }**  return 0;  } | output: 01  1. throw bool type  2. destructor will not run |

throw string("0") -> catch(string &s)

|  |  |
| --- | --- |
| class X{  public:  X() { cout << 1; };  ~X() { cout << 2; };  };  void exec(){  **{ X x;}**  **throw string("0");**  }  int main(void) {  **try { exec();}**  **catch(string &s){ cout << s; }**  return 0;  } | output: 120  1. throw string("")  2. once throw exception, destructor will be run. |
|  |  |

### throw in function:

"double func (char param) throw (int);" This function func() return double float, only throw int type exception. try cause will not catch it if throw other type.

|  |  |
| --- | --- |
| class X{  public:  **X() throw(int) { cout << 1; };**  **~X() throw(int) { cout << 2; };**  void exec() { throw 0; }  };  void exec(X &x){ x.exec(); }  int main(void) {  **X x;**  **try { exec(x);}**  **catch(int &i){ cout << 0; }**  return 0;  } | output:102  1. define throw in constructor  2. destructor will be run even if exception is thrown when define class using "X x;" |
| class X{  public:  **X() throw(int) { cout << 1; };**  **~X() throw(int) { cout << 2; };**  void exec() { throw 0; }  };  void exec(X\* x){ x->exec(); }  int main(void) {  **X \*x = new X();**  **try { exec(x);}**  catch(int &i){ cout << 0; }  return 0;  } | output: 10  compared with the above  1. use class pointer, destructor will not run if exception is thrown |
|  |  |

## hierarchy of exceptions

|  |  |
| --- | --- |
| class Class {  public:  string msg;  Class(string txt) : msg(txt) {}  };  void function(int i) throw (string,Class) {  switch(i) {  case 0 : throw string("string");  case 1 : throw Class("object");  default: cout << "OK" << endl;  }  }  void level(int i) throw(Class) {  try {  function(i);  } catch(string &exc) {  cout << "String [" << exc << "] caught in level()" << endl;  }  }  int main(void) {  for(int i = 0; i < 2; i++) {  cout << "-------" << endl;  try {  level(i);  } catch(Class &exc) {  cout << "Object [" << exc.msg << "] caught in main()" << endl;  }  }  return 0;  } | output:  -------  String [string] caught in level()  -------  Object [object] caught in main() |
| int c=3;  try{  switch(c){  case 1: throw 20;  case 2: throw 5.2f;  case 3: throw 'a';  }  }  catch(int e){cout << "int exception";}  catch(float e){cout << "float exception";}  catch(...){cout << "other exception";} | output: other exception |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## customary exceptions

|  |  |
| --- | --- |
| class X: public logic\_error{  public: X(): logic\_error("0") {};  };  void exec(X &x) throw(X) { throw x; }  int main(void) {  X x;  try { exec(x); }  catch(X &i){ cout << i.what(); }  return 0;  } | output:0 |
| class X: public logic\_error{  public:  X(string s): logic\_error(s) {};  ~X() { cout << 2; }  };  void exec() throw(logic\_error) {  X x("0"); throw x; cout << 3;  }  int main(void) {  // X x("4");  try { exec(); }  catch(X &i){ cout << i.what(); }  return 0;  } | output: 202  ????? should be 02 ?? why destructor run twice |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## syntax errors

|  |  |
| --- | --- |
| int main(void) {  int a =8, b=0, c=6;  try{c=a/b;}  cout << c << endl;  return 0;  } | miss catch block   expected 'catch' before 'cout' |
| **void f(void){ throw exception("?");}**  int main(void) {  int a=4;  try { f(); }  catch(...){ a++; }  catch (exception &e){ a--; }  cout << a << endl;  return 0;  } | two errors:  1 should be "throw exception();" can't pass "?".  2.'...' handler must be the last handler for its try block |
| class X: public runtime\_error {  public: X(): domain\_error("0") {};  };  void z() throw(X){  X x;  throw x;  cout << 2;  }  int main(void) {  X x;  try { z();}  catch(X &i){ cout << 1;}  catch(domain\_error &i){ cout<<0;}  return 0;  } | wrong inheritance of exceptions  std:stdexcept -> logic\_error -> domain\_error |
| class X: public logic\_error{  public:  X(): logic\_error("0") {};  };  void exec() throw(X) {  throw new logic\_error("0");  }  int main(void) {  X x;  try { exec(); }  catch(X &i){ cout << i.what(); }  return 0;  } | terminate called after throwing an instance of 'std::logic\_error\*'  ??????????????? |
| int main(void) {  try { throw 3.14; }  catch(double x){ x\*=2; }  cout <<x;  return 0;  } | out of scope. x is only defined in catch(){}. |
| class X{  public:  X(void) throw(int) {cout<<1;}  ~X(void) throw(int) {cout<<2;}  **void exec() { throw string("0");** }  };  void exec(X &x) {  x.exec();  }  int main(void) {  X x;  try { exec(x); }  catch(int &i){ cout << i; }  return 1;  } | correct: catch(string &i) |
|  |  |
| int main(void) {  try { throw 2./4; }  catch(int i){ cout << i; }  return 0;  } | terminate called after throwing an instance of 'double'  correct: catch(double i) |
|  |  |
|  |  |

# function

function declaration, definition, and call

## declare, define, and call

Function declaration, is done to tell the compiler about the existence of the function. Function's return type, its name & parameter list is mentioned.

|  |  |
| --- | --- |
| int main(void){  // declare the function  int f();  // call the function  f();  return 0;  }  // define a function  int f(){  cout<<"func";  return 1;  } | 1. should declare the function before call it, because the definition of this function is behind the main().  2. put f() preceeds main(). it is ok to ignore declaration step |
| int main(void){  // declare the function  int f(double);  // call the function  float i=2.3;  f(i);  return 0;  }  // define a function  int f(double x){  cout<<"func";  return 1;  } |  |
| int main(void){  // declare the function  int f(int, int);  // call the function  int a=1,b=2;  f(a,b);  return 0;  }  // define a function  int f(int x, int y){  cout<<"func";  return 1;  } |  |
| int main(void){  // declare the function  int f(int, void \*);  // call the function  int a=1;  void \*b=nullptr;  f(a,b);  return 0;  }  // define a function  int f(int x, void \*y){  cout<<"func";  return 1;  } |  |
| int main()  int main(){  return 1;  } | No needs to declare main() in C++ |
| int main(int argc, char \*argv[]){    cout<< "number of arguments:" << argc << endl;  cout<< "file name: " << argv[0] << endl;  return 0;  } | output:  number of arguments:1  file name: F:\cpp\template.exe |

|  |  |
| --- | --- |
| #include <iostream>  using namespace std;  int main(void){  int func(void)  {return 2;}  int res = func();  cout << res << endl;  return 0;  } | compiling error  The func() can't be placed within the function main() in C++. But it is ok in C |
| #include <iostream>  using namespace std;  int func(void){  return 2;}  int main(void){  int res = func();  cout << res << endl;  return 0;  } | one approach is to move the func() out of main() |

## arguments and return

1. default arguments

|  |  |
| --- | --- |
|  |  |
| int fun(int p1 =1, int p2=1){  return p2<<p1;  }  int main(void){  cout<<fun();  cout<<fun(2);  cout<<fun(2,3);  return 0;  } | output:2412  1<<1=2  1<<2=4  3<<2=3\*4=12  default arguments |
| int fun(int p1 =1, int p2=1){  return p2<<p1;  }  int main(void){  cout<<fun(fun(),fun(2));  return 0;  } | output 16 |
| bool f(bool x, bool y){return x<y;}  int main(void) {  cout << f(true,false);  return 0;  } | output: 0  bool func() return 0 or 1 |
|  |  |

2. return statements

|  |  |
| --- | --- |
| int f(void){  int a=3;  }  int main(){  cout<< f();  return 0;  } | output: 1  Note: return statement is missing. return 1 or unpredictable value |
| int f(void){  int a;  return a=4;  }  int main(){  cout<< f();  return 0;  } | output: 4  Note: return a; |
| char f1(char c){return c=='z' ? 'a': c+1;}  char f2(char &c){c=f1(c);return c;}  int main(void){  char x='x';  cout<<f2(x);  cout<<f2(x);  cout<<f2(x)<<endl;  return 0;  } | output: yza  In f2(char &c), the variable c and the arguments x share the same address. |

## inline function

If the compiler performs function inlining, it replaces a call of that function with its body, avoiding the overhead of a function call (placing data on stack and retrieving the result), which may result in a larger executable as the code for the function has to be repeated multiple times. The result is similar to [function-like macros](https://en.cppreference.com/w/c/preprocessor/replace), except that identifiers and macros used in the function refer to the definitions visible at the point of definition, not at the point of call.

inline function should be quickly executable. inline function could work like MACRO but address more functions.

|  |  |
| --- | --- |
| #include <iostream>  #include <cmath>  #include <iomanip>  using namespace std;  inline int func(void)  {  return 2;  }  int main(void){  int res = func();  cout << res << endl;  return 0;  } |  |
| using namespace std;  **inline int**  **function (int parameter) {**  **return parameter \* 2;**  **}**  int main(void) {  int  var = 1;  var = function (var);  var = function (var);  var = function (var);  cout << var << endl;  return 0;  } | split them into line with "inline int" and function()  Or "int inline func()" |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## pass pointer or reference

A pointer is a variable that hold memory address of another variable

A reference is same object, just with a different name and reference must refer to an object. Since references can’t be NULL, they are safer to use. It is useful to change values of variables within functions.

1. A pointer can be re-assigned while reference cannot, and must be assigned at initialization only.
2. Pointer can be assigned NULL directly, whereas reference cannot.
3. Pointers can iterate over an array, we can use increment/decrement operators to go to the next/previous item that a pointer is pointing to.
4. A pointer is a variable that holds a memory address. A reference has the same memory address as the item it references.
5. A pointer to a class/struct uses ‘->'(arrow operator) to access it’s members whereas a reference uses a ‘.'(dot operator)
6. A pointer needs to be dereferenced with \* to access the memory location it points to, whereas a reference can be used directly.

pass variable by reference

|  |  |
| --- | --- |
| int f(int &a){  return ++a;  }  int main(void){  int t=3;  cout<< f(t) << endl;  return 0;  } | output:4  reference &a of variable t. a is alias name of t. |
| int \*f(int &a){  return &++a;  }  int main(void){  int t=3;  cout<< \*f(t) << endl;  return 0;  } | output: 4;  return pointer of reference, then dereference the pointer in main() |
|  |  |
|  |  |
|  |  |

array to reference

Pass array to function in C is to pass pointer to function. It is not recommended to pass array to reference. The reference approach is clumsy compared with pointer approach.

|  |  |
| --- | --- |
| int f(int (&a)[4]){  return \*a;  }  int main(void){  int t[]={1,2,3,4};  cout<< f(t) << endl;  return 0;  } | output:1  Note pass array by reference, the size should be explicitly declared |
| int f(int (&a)[4]){  return \*a++;  } | wrong: a is reference, can't move or initialized as pointer |
| int f(int (&a)[]){  return \*a;  } | wrong: size of array to reference should be declared. |
| int \*f(int &a){  //move pointer  **return &a+1;**  }  int main(void){  int t[]={1,2,3,4};  cout<< \*f(\*t) << endl;  return 0;  } | output: 2  pass reference of 1st value to function, and reference the reference as pointer before moving and returns |
| int f(int &a){  //move pointer and return value  return \*(&a+1);  }  int main(void){  int t[]={1,2,3,4};  cout<< f(\*t) << endl;  return 0;  } | The pattern is equal to the above. |

|  |  |
| --- | --- |
| using namespace std;  int f1(int p){  ++p;  return p++;  }  **int f2(int &p){**  **++p;**  **return p++;**  **}**  int main(void){  int a=1, b, c;  b=f1(a);  c=f2(b);  cout << a+b+c << endl;  return 0;  } | output: 8  pass value: b=f1(1)=2  pass pointer: c=f2(b=2)=3, b=4  a=1, b=4, c=3  define f2(int &p), and reference: f2(b)  so pass address of b into function, p and b share the same address, or p and b are the same. |
| // pass value: copy value, different address  char f1(char c){  return c++;  }  // pass reference: same value, same address  char f2(char &c){  c--;  return c++;  }  int main(void){  // reference  char x='x';  cout<<f1(x);  cout<<f2(x);  cout<<x<<endl;  return 0;  } | output:xwx |
| // pass pointer itself has its own address  // point a memory address,  //which may be allocated to a variable  char f3(char \*c){  (\*c)--;  // return \*c firstly  // then move pointer c rather than x;  return \*c++;  }  char f4(char \*c){  // return \*c firstly, then add value  return (\*c)++;  }  int main(void){  x='x';  cout<<f3(&x);  cout<<f4(&x);  cout<<x<<endl;  return 0;  } | output:wwx |
| int f1(int \*a){  return \*a+1;  }  int \*f2(int \*a){  return a+1;  }  int \*f3(int &a){  return &a+1;  }  int main(void){  int t[]={0,1,2,3};  cout<< f1(f3(\*f2(t)));  return 0;  } | output:3 |
| string fun(string &t, string s="", int r=2){  while(--r)  s +=s;  t +=s;  return s;  }  int main(void){  string name ="x";  cout << fun(name, name);  cout<<name;  return 0;  } | output: xxxxx  while loop execute one time:  s=s+s="xx"=>fun()="xx"  t=t+s="xxx" => name="xxx" |
|  |  |

## overloading

functions are denoted as the same name, but have different arguments. Overloading is kind of polymorphism of OOP

Here are overloading rules:

1. Prerequisite: Function name must be identical.

2.. There is exact match: Type and number of parameters match with those of arguments.

3. If a not exact match is found: Char, Unsigned char, and short are promoted to an int. Or Float is promoted to double

4. If still no match is found: C++ tries to find a match through the standard conversion.

5. compiling error: could not convert ... from ... to 'std::\_\_cxx11::...'

|  |  |
| --- | --- |
|  |  |
| **float func(int a){ return ++a;}**  int main(void) {  double i=20.5;cout << func(i);  return 0;  } | output: int:21  type\_casting:  pass arguments: convert double to int  return: convert int to float |
| int func(int a){return ++a; }  int main(void) {  long i=1234567890; cout << func(i);  return 0;  } | output: 1234567891 |
| int func(int a){ return ++a; }  int main(void) {  long i=12345678900;cout << func(i);  return 0;  } | output: -539222987 depending platform  long int is bigger than int |
|  |  |

## syntax errors

|  |  |
| --- | --- |
| **int func(int a){**  cout<<"int a"<<endl;  return ++a;  }  **int func(int &a){**  cout<<"reference a"<<endl;  return a++;  }  int main(void) {  int i=2;**func(i);**  return 0;  } | error: overloaded is ambiguous  calling func() with value or reference is unknown. |
| void f(string s){}  int main(void) {  int i=2; f(i);  return 0;  } | error: could not convert 'i' from 'int' to 'std::\_\_cxx11::string  {aka std::\_\_cxx11::basic\_string<char>}' |

# C-like data type

## digits and character

c++ don't support "unsigned double float".

### digits

|  |  |
| --- | --- |
| 0x10 | output: 16  correct: 0X16 |
| x10 | invalid |
| int i=1,j=2;  if(i>j&&j>i) i++;  if(i>j||j>1) j++;  if(i|j>1) i++;  if(i&j) j++;  cout << i\*j << endl; | output:8 |
| int i=2;  float a=3;  bool f1=a/i>i/a;  float f2=a/i>i/a;  int f3=a/i>i/a;  cout << f1 << f2 << f3 << endl; | output:111 |
| #include <iostream>  using namespace std;  int main(void) {  cout << "This computing environment uses:" << endl;  cout << sizeof(char) << " bytes for chars" << endl;  cout << sizeof(short int) << " bytes for shorts" << endl;  cout << sizeof(int) << " bytes for ints" << endl;  cout << sizeof(long int) << " bytes for longs" << endl;  cout << sizeof(float) << " bytes for floats" << endl;  cout << sizeof(double) << " bytes for doubles" << endl;  cout << sizeof(bool) << " byte for bools" << endl;  cout << sizeof(int \* ) << " bytes for pointers" << endl;  return 0;  } | 1 bytes for chars  2 bytes for shorts  4 bytes for ints  8 bytes for longs  4 bytes for floats  8 bytes for doubles  1 byte for bools  **8 bytes for pointers ??** |
| int a, b;  a=b=1; | it is ok to combining assignment |
| int a=b=1; | compiling error |
| float a(3); | It is C++ style. Being equal to "float a= 3;" in C |
| int i=3;  **i=(i--);**  cout << i; | output:3  The execution order: decrement i=3 to i=2 firstly, then return previous value i=3 to i, so i=3. |
| const a =3.14;  const b= a\*a;  cout << b; |  |

### type conversion

short int (2 bytes, (2^4)^2=2^8=256) , long int(8 bytes, (2^4)^8=42^32=,297,967,296), int(4bytes, (2^4)^4)=2^16=65536)

float, double float

|  |  |
| --- | --- |
| int a = 1; float i = 2;   cout << a/i; | output: 0.5  **Note: implicit convert to float** |
| int a=1; float i=2;  cout<< (float) a/i; | the same as the above. That is explicit conversion. |
| short s=1;int i=2;  float f=4.4;double d=6.6;  cout<< s/float(i) + int(f)/i + long(d)/s; | output:8.5  s/float(i) = 1/2.0=0.5  int(f)/i =4/2=2  long(d)/s=6/1=6 |
| short s=1;int i=2;  float f=4.4;double d=6.6;  cout<< s/i + f/i + d/s; | output:8.8  0+2.2+6.6 |
| short s=1;int i=2;  float f=4.4;double d=6.6;  int x = s/i + f/i + d/s;  cout<< x; | output:8  0+2+6 |
| float f = 3.21; double d = 1.23;  int k = int(f) + (int) d; cout << k << endl; | output: 4 |
| float f = 1234.5678;  double d = f;  if (d == f)cout << "equal" << endl;  elsecout << "not equal" << endl; | output: equal  Note: double is 8 bytes, and float is 4 bytes. double has wider range than float. |
| **double d = 123456.789012; float f = d;**  **if (d == f) cout << "equal" << endl;**  **else cout << "not equal" << endl;** | **output: not equal** |
| int i=10;  float f=3.5;  cout << float(i)/int(f); | **output: 3.33333** |
| int \*i = new int;  \*i=1.0/2\*2/1\*2/4\*4;  cout << \*i; | **output: 2**  1.0/2\*2/1\*2/4\*4 = 0.5\*2/1\*2/4\*4 =1.0/1\*2/4\*4=1.0\*2/4\*4=2.0/4\*4=0.5\*4=2.0->2 |
| int a=2; float b=1;  cout << (float) a % b; | compiling error  Note: % is not working for float type |
| float func(double x){ return x/(.5\*x); }  int main(void){  int i=2;  int v = func(i);  cout << v;  return 0;  } | though argument is double type, pass int is ok. |

explicit conversion

|  |  |
| --- | --- |
| double scores = 95.5;  int n = static\_cast<int> (scores); | n=95  static\_cast convert data type |
| float a=1.2; a = (int) a;  cout<<a; | output: 1 |
| double i = -5.55; cout << (int) i; | output:-5 |
|  |  |
|  |  |

## enum

|  |  |
| --- | --- |
| enum e {a, b,c,d};  cout << a << e(2) << e(a); | output: 020  #define a 0 |
| enum e {a, b,c,d};  cout << e(b+1); | output:2  #define b 1 |
| enum e {a, b,c,d};  e f = e(a+c);  cout <<f <<endl; | output:2  e f = e(0+2) =e(2)=2 |
| enum e {a, b,c,d=-1};  cout <<e(b) <<endl; | output:2  a=0,b=1,c=2,d=-1 |
| enum e {a=1, b,c,d};  **e f = c;**  cout << f <<endl; | output:3 |
| enum e {a=1, b,c,d};  **e f = e(a+1);**  cout << f <<endl; | output:2 |
| enum e {a=1, b,c,d};  e& operator++(e &x){x=a; return x; }  int main(void) {  e f = c; cout << int(++f) <<endl;  return 0;  } | output:1  f=3 -> ++f -> x=1  Note: x=e(a+1) is right. x=++a is wrong. a is enum |
| enum answer {yes, no, na};  enum answer a[3];  a[0] = no; a[1] = yes; a[2] =na;  for(int i = 0; i < 3; i++)  cout << a[i]; | output:102  instantiate enum type as a |
| enum state{ok, error, warning};  enum state s1, s2, s3;  s1=ok; s2=warning; s3=error;  cout << s1 << s2 << s3; | output:021 |
|  |  |
|  |  |
|  |  |
|  |  |

### syntax errors

|  |  |
| --- | --- |
| enum e {a=1, b,c,d};  e f = a+1;  cout <<f <<endl; | compiling error: invalid conversion from int to main()::e  correct: e f = e(a+1)  Note: use overloading operator+ |
| class Int{  public: T v;  Int(T a){ v = a; }  Int& operator++(){ v += 2; return \*this;}  };  ostream& operator<<(ostream &o, Int &a){  ++a; return o << a.v;  }  int main(void) {  Int i = B;cout << i;  return 0;  } | compiling error  v+=2 is wrong. v is enum type  correct v= T(v+2) |
| enum e {a=1, b,c,d};  e d = e(a+1);  cout << d <<endl; | compiling error: previous declaration 'main()::e d'  can't update d because d is declared. |

## array

integer, float, char, bool array

C aproach

|  |  |
| --- | --- |
| int arr[3]={1,2,3}; |  |
| float arr[]={.4,.7,10.3}; |  |
| bool arr[]={1,1,}; |  |
| char arr[]={'A', 'B', '\0'}; |  |
|  |  |
|  |  |

C++ approach

|  |  |
| --- | --- |
|  |  |
| int row=2, col=3;  **int \*\*p = new int \*[row];**  for(int i = 0; i < row; i++){  **p[i] = new int[col];**  for(int j = 0; j < col; j++){  **p[i][j] = i+j;**  cout << p[i][j];  }  cout << endl;  } | 2D array: 012, 123 |
| int \*b=NULL;  b = new int [1];  b[0]=3;  cout << \*b << endl; |  |
|  |  |
|  |  |
|  |  |
|  |  |

New methods in c++ compared c.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| size of array | int a[10] = {1,2};  cout << "size of array" << end(a)-begin(a) <<endl; | end(arr), begin(arr) return address of start and end pointer. |
| sizeof() | int a[10] = {1,2};  cout << sizeof(a) << endl; | output:40 |
|  |  |  |

## struct

### basic

|  |  |
| --- | --- |
| struct str{int t[3];char s[3];};  str a = {1,2,3, 'a','b','c'};  str b = {5,6,7,'x','y','z'};  cout << char(b.s[0]+a.t[0]) << int(a.s[2] - a.s[0]) << int(b.s[2]-b.s[1]); | output: y21  char(b.s[0]+a.t[0])=char('x'+1)='y'  int(a.s[2] - a.s[0])=int(3-1)=2  int(b.s[2]-b.s[1])=int('z'-'y')=1 |
| struct sct{int t[2];};  struct str {sct t[2];};  str t[2] = {{0,2,4,6},{1,3,5,7}};  cout << t[1].t[0].t[1] << t[0].t[1].t[0]; | output: 34  nested struct array |
| struct T{  int v;  struct T \*next;  };  struct T t1, t2;  t2.v=2, t2.next = NULL;  t1.v=1, t1.next = &t2;  cout << t1.v << t1.next->v; | output:12 |
|  |  |
|  |  |

### syntax error

|  |  |
| --- | --- |
| struct T{  int v;  struct T next;  }; | error: field 'next' has incomplete type 'main()::T'  correct: struct T{ int v; struct T \*next;}; |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# memory

## memory leak

## The new and delete keyword

the new keyword create a new block address for variable and create object. But the variable should be initialized in the next.

The new will return a pointer. Here is the usage:

new <data-type>

In C, they could be malloc() and free() only ask for the memory. the variable should be created in advance. It is ok to use them in c++ but not suggested.

|  |  |
| --- | --- |
|  | output |
| int \*ptr = new int[10];  cout << sizeof(ptr) << endl;  cout << ptr << endl; | 4  0xf27f48 |
| float \*ptr = new float[10];  cout << sizeof(ptr) << endl;  cout << ptr << endl; |  |
|  |  |
|  |  |
|  |  |
|  |  |

the delete word

|  |  |
| --- | --- |
| int \*ptr = new int;  cout << sizeof(ptr) << endl;  cout << ptr << endl;  delete ptr; |  |
| char \*ptr = new char[10];  cout << sizeof(ptr) << endl;  cout << ptr << endl;  **delete [] ptr;** | "delete [] ptr;" will delete the array of pointers. That is entire array  "delete ptr;" will only delete the first cell.  "delete ptr[1]" will delete the 2nd object allocations |
| // Creating int pointer  int\* ptr1 = new int;    // Initializing pointer with value 20  int\* ptr2 = new int(20);    cout << "Value of ptr1 = " << \*ptr1 << "\n";  cout << "Value of ptr2 = " << \*ptr2 << "\n";    delete ptr1; // Destroying ptr1  delete ptr2; // Detroying ptr2 | delete variable no matter whether that is initialized or not. |
| void \*ptr;  delete ptr; | delete void pointer |
| int \*ptr = (int \*)malloc(10\*sizeof(int))  delete [] ptr; | delete array allocated by malloc() |
| int x;  delete x; | compiling error.  can't delete nonpointer |
| int i=3;  int \*ptr=&i;  cout << i << endl;  delete ptr;  cout << \*ptr << endl; | **though no compiling error, the delete will not work. delete only delete dynamically allocated memory, but the variable is static variable.** |