S1 process: fast, unconscious, effortless, and inflexible
S2 process: slow, conscious, effortful, and flexible

Typedefs	Pointer Code	Works	Does not work
T* pointer	pointer p = &i const pointer& = p	++*r i == 3 p = 0 r == 0	r = 0
const T* p_t_c	p_t_c p = &i p_t_c& r = p	r = 0 p == 0	++*r ++*p
	p_t_c p = &i const p_t_c& r = p	p = 0 r == 0	++*r ++*p r = 0
T* const c_point	c_point p = &i c_point& r = p	++*r i == 3	r = 0 p = 0
const T* const c_p_t_c	c_p_t_c p = &i c_p_t_c& r = p		++*r ++*p r = 0 p = 0
T* pointer const T* p_t_c	pointer p = &i p_t_c pc = p		p_t_c& r = p pointer q = pc pointer& s = pc
T* const c_p const T* const c_p_t_p	c_p cp = &i c_p_t_c cpc = cp c_p_t_c& r = cp		c_p cq = cpc c_p& s = cpc

What function is available	Which line calls this function
f(int)	int i = 2; const int ci = 3; f(i) f(ci) f(4)
g(int&)	g(i)
g(cons tint&)	g(ci) g(4)
h(int*)	h(&i) h(static_cast<int*>(0))
h(const tint*)	h(&ci) h(static_cast<const int*>(0))
x(int)	x(2)
x(long)	x(3L)
struct A{ A (int n){}}; x(A)	x(A(2))
struct B : A{ B(int n){} };	x(B(3))

```
template <typename T>
struct A {
   static int si;};
template <typename T>
int A<T>::si;
int main () {
   cout << ( A<int>::si ==  A<double>::si) << endl; // true
   cout << (&A<int>::si == &A<double>::si) << endl;} // false
-----------------------------------------------------
template <typename T>
struct A {
   static T v0;
//  static T v1 = v0 + 1; //error: forbids in-class initialization of non-const static
   static T v1;
//  static const T cv0 = 0; // error: forbids initialization of member constant "A<double>::cv1" of non-integral type "const double"
   static const T cv0;
   static const int cv1 = cv0 + 1;
   enum {ev = 3};};
template <typename T>
T A<T>::v0;
template <typename T>
T A<T>::v1 = v0 + 1;
//template <typename T> // error: uninitialized const "A<T>::cv0"
//const T A<T>::cv0;
template <typename T>
const T A<T>::cv0 = T();
template <typename T>
const int A<T>::cv1;
int main () {
   assert(A<int>::v0    == 0);
   assert(A<double>::v0 == 0);
//  assert(&A<int>::v0   != &A<double>::v0);  // error: comparison between distinct pointer types "int*" and "double*" lacks a cast
   assert(A<int>::v1    == 1);
   assert(A<double>::v1 == 1);
//  assert(&A<int>::v1   != &A<double>::v1);  // error: comparison between distinct pointer types "int*" and "double*" lacks a cast
   assert(A<int>::cv0    == 0);
   assert(A<double>::cv0 == 0);
//  assert(&A<int>::cv0   != &A<double>::cv0); // error: comparison between distinct pointer types "int*" and "double*" lacks a cast
   assert(A<int>::cv1    == 1);
   assert(A<double>::cv1 == 1);
   assert(&A<int>::cv1   != &A<double>::cv1);
   assert(A<int>::ev    == 3);
   assert(A<double>::ev == 3);
   A<int> x;
   A<int> y;
   assert(&x.cv1 == &y.cv1);
```

```
class A {
    A (int v, int* p, int& r) :_r (r) {"A(int, int*, int&)"
    A (const A& that) {"A(const A&)"}
    ~A () {"~A()"
struct B {
    A _x;
    B (int v, int* p, int& r) :  _x (v, p, r) {"B(int, int*, int&)"
    B (const A& x) : _x (x) {"B(const A&)"}
    ~B () { "~B()"}
int main () {
   int  v = 2;    int* p = &v ;    int& r = v;
   A x(v, p, r);
   {    B y(v, p, r);   }
   {   B z(x)    }
```

```
A(int, int*, int&)

A(int, int*, int&)
B(int, int*, int&)
~B()
~A()

A(const A&)
B(const A&)
~B()
~A()

~A()
Done.
*/
```

```
class A {
  private:
    static int _cv;
        int _iv;
    mutable int _iw;
  public:
    A () :
        _iv (0),
        _iw (1) {
      cm();
      im();
      cim();
      A* p = this;}
    static void cm () {
      ++_cv;
//      ++_iv;     // error: invalid use of member 'A::_iv' in
static member function
//      ++_iw;     // error: invalid use of member 'A::_iv' in
static member function
//      im();      // error: cannot call member function 'void
A::im()' without object
//      cim();     // error: cannot call member function 'void
A::cim() const' without object
//      A* p = this; // error: 'this' is unavailable for static
member functions
      }
    void im () {
      ++_cv;
      ++_iv;
      ++_iw;
      cm();
      cim();
        A* p = this;
      const A* q = this;}
    void cim () const {
      ++_cv;
//      ++_iv;          // error: increment of data-member
'A::_iv' in read-only structure
      ++_iw;
      cm();
//      im();           // error: no matching function for call
to 'A::im() const'
//          A* p = this;   // error: invalid conversion from
'const A* const' to 'A*'
      const A* q = this;}};
int A::_cv;
int main () {
  using namespace std;
  cout << "Methods.c++" << endl;
  A::cm();
// A::im();  // error: cannot call member function 'void
A::im()' without object
// A::cim(); // error: cannot call member function 'void
A::cim() const' without object
  {
  A x;
  x.cm();
  x.im();
  x.cim();
  }
  {
  const A x;
  x.cm();
// x.im();   // error: no matching function for call to 'A::im()
const'
  x.cim();
  }
```

---

1. In C++ there are five causes for the default constructor to not be automatically generated. List any two.
defining any constructor
containing a const
containing a reference
containing a user-defined type with no public default constructor
having a superclass with no public default constructor
2. In C++ there are four automatically generated methods that exhibit refinement overriding. List any two.
default constructor
copy constructor
copy assignment operator
destructor

In java, finalize will use replacement.

Keyword static for local variable Changes: how many there are, when it's allocated, when it's initialized, the lifetime, but doesn't change the scope.
For the global variable, it only changes the scope.
The struct class doesn't give the default constructor or copy assignment operator without explicityly declaring it.

Class (static) variables can only be init in static init blocks or at the line of declaration.