```
++ Opšte ++
const int a;      // mora se odmah inicijalizovati
std::cin >> a;    // pre učitavanja, prazni cout
using std::cout   // cout umesto std::cout
"aaa" "bbb" <=> "aaabbb"
cin >> mid >> fin; <=> cin >> mid; cin >> fin;
* Preciznost ispisa
  <ios> <iomanip>
  streamsize prec = cout.precision();
  cout << setprecision(3) << mid << setprecision(prec) << endl;
* Uslov
  if(objekat) ... // koristi konverziju u bool ili poželjnije u void*


++ Promenljive ++
* Automatske (lokalne)
* Statičke (globalne i ključna reč static)
  int *p() {
    static int x;
    return &x;
  } // x je jedinstveno pri svakom poyicu i pravi se samo jednom
* Dinamičke (new/delete)
  nizovi:
    p* = new T[n]; // za n=0, ne vraća pokazivač
     ...           // na prvi već na jedan iza prvog
    delete [] p;   // uklanja ceo niz, a ne samo prvi
```

Ako se ne inicijalizuju pri definiciji, automatskim promenljivama se dodeljuje podrazumevana vrednost što je za ugrađene tipove nedefinisana vrednost, a za klasne tipove zavisi od konstruktora bez parametara. Podrazumevani konstruktor na isti način inicijalizuje osobine klase.

Konstantne reference i pokazivači ne mogu biti korišćeni za promenu memorije na koju pokazuju. (T i; const T& i2 = i; // i2 nije lvalue). Zato const T& kao parametar f-je kaže: "ne kopiraj, a ja neću promeniti vrednost". Argumenti koji odgovaraju nekonstantnim referencama moraju biti lvalues.

```
++ Klase ++
* struct - sve javno, class - sve privatno
* Funkcije članice definisane u telu klase iniciraju inline oznaku
* ::grade(); u telu metoda poziva globalnu funkciju grade() van klase
* Lista inicijalizacije
  Konstruktor() : var(0), var2(1) { }
  Inicijalizaciju se po redu navođenja u deklaraciji klase.
  Dodela u telu konstruktora poziva operator dodele.
* Uslov trojke
  Ako je potreban destruktor, potreban je i konstruktor za kopiranje i
  operator dodele ali mogu biti privatni ili deklarisani, a nedefinisani
* Ako se definiše konstruktor sa parametrima, mora se definisati eksplicitno i
konstruktor bez parametara
* Ako je potrebno definisati "unakrsne" klase neke se mogu deklarisati sa class
X; uz naknadnu definiciju
* static metode pripadaju imenskom prostoru klase
```

* Postoje ugneždene klase
* Privatnost članova je na nivou klase, a ne njene instance

++ Operatori ++
```
istream& operator >> (istream& i, T& x);
ostream& operator << (ostream& o, const T& x);
class T {
  friend istream& operator >> (istream&, T&);
  T& operator += (const T& x) {
     ...
     return *this;
  }
  double operator double() const { ... }

  // Ako postoji prefiksni može se iskoristiti
  // i kao postfiksni. Obrnuto ne važi.
  T& operator ++ () {  // prefiksni - ++A
     ...
     return *this;
  }
  T operator ++ (int) { // postfiksni - A++ - nije referenca!
     T r = *this; ... ; return r;
  }
};
T operator + (const T& x, const T& y) {
    T z = x; // kopija
    z += y;  // operator +=
    return z;
}
// Binarni + definišemo van klase kako ne bi poremetili
// simetriju pri konverziji operanada.
```

Operator [] (const i obični koji se ne preklapaju iako imaju iste parametre jer svaka funkcija članica ima i operator this koji je const ili ne) i operator dodele moraju biti članice klase!

++ Konstruktori i operatori dodele ++
```
string a = "aaa";    // Inicijalizacija, poziva string
                     // konstruktor sa parametrom const char *
string b;            // Podrazumevani konstruktor
b = a;               // Operator dodele: uništava b i inicijalizuje novi sa a
T v;
v = f("aaa");        // Parametar f se inicijalizuje sa "aaa",
                     // a zatim se povratna vrednost inicijalizuje
                     // i na kraju dodeljuje operatorom objektu v
```

* Operator dodele
Dodela kada uklanjamo prethodni, inače kopiranje inicijalizacijom.
```
T& operator = (const T& y) {
   if(&y != this) {
      destroy, copy
   }
   return *this;
```

```
}
```

```
* Konstruktor kopije
T(const T& x) ... // const jer ne menja parametar
```
Za šablonske konstruktore po nekom T2 šablonske klase po T1 mora eksplicitno da se definiše sa parametrom T<T1>.

```
* Eksplicitni i implicitni poziv konstruktora
T vi(100);      // eksplicitni poziv
T vi = 100;     // implicitni poziv
vi + 100;       // <=> T temp; vi + temp(100);

class T {
  explicit T(int n) ...   // dozvoljava samo eksplicitni poziv konstruktora
}
```

```
++ Nasleđivanje ++
```
Nasleđuje se sve osim konstruktora, destruktora i operatora dodele

```
class New : public Core { ... };  // public - sve ostaje isto
                                  // private - sve private
                                  // protected - public postaje protected
                                  //             private kao da ne postoji
```
protected zaštićenost članica klase - pun pristup samo izvedenim klasama, private za ostale

Redosled: Rezervišemo prostor za ceo izvedeni objekat, Konstruktor osnovne (osnovni, bez parametara), Inicijalizaciju izvedene prema listi inicijalizacije, Konstruktor izvedene

Broj parametara konstruktora osnovne i izvedene ne mora biti isti.

Objekat izvedene klase ima pristup proteced članovima samo svog podobjekta svoje bazne klase, a ne svih istanci.

```
* Dinamičko povezivanje
virtual double f(); // pamti stvarni tip objekta uz referencu ili pokazivač.
                    // oznaka je nasledna
Izvedena c1; Bazna &c2 = c1; c2.f(); // zove f iz izvedene, a ne iz bazne!
```

Konstruktor, operator new i static metode ne mogu biti virtuelne.

```
* Virtuelni destruktor
class Bazna {
   public:
     virtual ~Bazna() {}
   private:
      double x;
};
```
Obezbeđuje dinamičko povezivanje. Ostali članovi bivaju uništeni inače pa je destruktor prazan. Izvedene klase bivaju korektno uništene ako se uništavaju njihove reference ili pokazivači.

```
* Apstraktne klase
class Node {
   virtual void f(int, double) = 0;
}


++ Šabloni ++
template <class T>
T F(vector<T> v) {
   typedef typename vector<T>::sizetype vec_sz;
           \_____ ovo je ime tipa ____/
   ...
};
char F(vector<char> v) {
   ...
}

F<int>(...); // obavezno ako nema parametara ili nije moguće utvrditi
F(...);

template <class vector<T> > // razmak da se razlikuje od operatora >>

template <class T, int n = 2>
class Klasa {
 ...
};


++ STL ++
* Kontejneri
K<T>::iterator
    ::const_iterator
    ::size_type

c.begin()           iterator na prvi
 .end()             iterator na mesto iza prvog
 .size()            broj elemenata - veličina
 .empty()           da li je prazan?

c.insert(d, b, e)   posle iteratora d ubacuje kopije vrednosti iteratora [b,e)
 .erase(i)          uklanja i ili [b,e). vraća iterator posle sekvence.
 .erase(b,e)        neki iteratori mogu postati nevažeći.

c.push_back(t)      dodaje t na kraj

k<T> c;             prazan
     c2(c);         kopija c
     c3(n);         n elemenata vrednosno inicijalizovano (0 ili konstruktor)
     c4(n, t);      n elemenata koje su kopije t
     c5(b, e);      elementi na koje pokazuju iteratori [b,e)

* Iteratori
*it, (*it).x, it->x, ++it, it++,...
```

```
* Vektor
v.reserve(n)    rezerviše dodatnih n mesta
v.resize(n)     menja veličinu na n, odseca višak
v.capacity()
v.back()        vrednost poslednjeg
v.pop_back()    ... pri tome ga i uklanja
v[n]            ne proverava da li n postoji
                n je tipa typename vector<T>::size_type

* Lista
l.sort(), l.sort(cmp)     // neopadajući redosled ili po cmp prediktu
l.push_front()

* Mapa
map<string, int> c; map<string, vector<int> >
++c["string"], c["novo"] vrednosno inicijalizovano (konstruktor ili na 0)
Iterator mape je pair<K, V>. it.first, it.second. Ključ je const.

m.begin(), m.end()
m.find(k) // vraća iterator na s sa ključem k ili .end ako k ne postoji

* Skup
set<char> c;
c.insert(char b)
c.count(char b)   // 1 ako je tu, inače 0
c.erase(char b)
c.size();

* String
string s = "AAAA";
string s1(n, "*");
s.size(), s.lenght()  // dužina
s.c_str()             // const char*
s[n], s+t, s<t, s!=t,...
s.swap(string t)      // zamenjuje s i t
s.find("AAA")         // pozicija prvog pojavljivanja ili vrednost string::npos

* #include<iterator>
back_inserter(c)  // iterator koji proširuje kontejner
front_inserter(c)
inserter(c, it)   // ispred it

* #include <algorithm>
sort(vec.begin(), vec.end())        // neopadajući redosled
sort(vec.begin(), vec.end(), cmp)   // ... ili po cmp prediktu
max(e1, e2)                         // e1, e2 istog tipa, vraća vrednost većeg
find(b, e, t), find_if(b, e, pred)  // u [b,e) traži t ili predikat pred
search(b, e, b2, e2)                // ... ili sekvencu [b2, e2)
```