Типови/Класе

Тип - Подсећање

- Неки основни појмови:
  - **Објекат** је парче меморије у којем је смештена вредност неког типа
  - **Променљива** је објекат који има име (и којем се, због тога, можемо обраћати директно)
  - **Тип** одређује **скуп вредности** које објекат може да има и **скуп операција** које се над тим вредностима могу извршавати

Тип - Подсећање

- Скуп вредности

- Главни поступак је композиција других типова.
- На почетку имамо уграђене, тј. основне типове.
- Кључне речи **struct** или **class**

```
class Token {
public:
    char kind;
    double value;
};
```

- Али, не морају увек све комбинације свих вредности елементарних типова бити ваљане вредности корисничког типа.

Тип - Подсећање

- Скуп операција

- Главни поступак је увођење функција.
- Али, које операције нам требају?
  - Зависи шта желимо да радимо.

```
class Token {
public:
    char kind;
    double value;
};
```

Тип - Подсећање

- Скуп операција

- Али, које операције нам требају?
  - Зависи шта желимо да радимо.
- Нпр.:

```
void foo() {
    MyType x;
    MyType y;
    add(x, y);
}
```

Тип - Подсећање

- Скуп операција

- Али, које операције нам требају?
  - Зависи шта желимо да радимо.
- Нпр.:

```
void foo() {
    int x;
}
```

```
void foo() {
    Token x;
}
```

Тип - Подсећање

- Скуп операција

- Али, које операције нам требају?
  - Зависи шта желимо да радимо.
- Нпр.:

```
void foo() {
    int x;
}
```

```
void foo() {
    int x = 5;
}
```

```
void foo() {
    Token x;
}
```

Тип - Подсећање

- Скуп операција
- Које операције нам требају?
- Операција стварања променљиве нам је увек потребна. (конструктор)

```
void foo() {
    Token x;
}
```

```
class Token {
public:
    Token() {}
    char kind;
    double value;
};
```

Конструктор је функција (чланица).
Специјална функција, али и даље функција.

Тип - Подсећање

- Скуп операција
- Које операције нам требају?
- Операција стварања променљиве нам је увек потребна. (конструктор)

```
void foo() {
    Token x;
}
```

```
class Token {
public:
    Token() : kind('x'), value(0.0) {}
    char kind;
    double value;
};
```

Када тело конструктора крене да се извршава подаци чланови су већ конструисани.

```
class Token {
public:
    Token() { kind = 'x'; value = 0.0; }
    char kind;
    double value;
};
```

Тип - Подсећање

- Скуп операција
- Које операције нам требају?
- Операција стварања променљиве нам је увек потребна. (конструктор)

```
void foo() {
    MyClass x;
}
```

```
class MyClass {
public:
    MyClass() : x(1, 5, "xyz"), y(0.0, 88) {}
    SCC x;
    OtherClass y;
};
```

Када тело конструктора крене да се извршава подаци чланови су већ конструисани.

```
class MyClass {
public:
    MyClass() { x = SCC(1, 5, "xyz"); ... }
    SCC x;
    OtherClass y;
};
```

Тип

- Скуп операција
- Које операције нам требају?
- Операција стварања променљиве нам је увек потребна. (конструктор)
- ?

```
Token foo(Token x) {
    return x;
}
foo(a);
```

Тип

- Скуп операција
- Које операције нам требају?
- Операција стварања променљиве нам је увек потребна. (конструктор)
- Операција стварања копије нам је врло често потребна. (конст. копије)

```
Token foo(Token x) {
    return x;
}
foo(a);
```

```
class Token {
public:
    Token() : kind('x'), value(0) {}
    Token(const Token& x) : kind(x.kind), value(x.value) {}
    char kind;
    double value;
};
```

Тип

- Скуп операција
- Које операције нам требају?
- Операција стварања променљиве нам је увек потребна. (конструктор)
- Операција стварања копије нам је врло често потребна. (конст. копије)
- ?

```
void foo() {
    int x;
    x = 5;
    x = y;
}
```

```
void foo() {
    Token x;
    x = y;
}
```

Тип

- Скуп операција
- Које операције нам требају?
- Операција стварања променљиве нам је увек потребна. (конструктор)
- Операција стварања копије нам је врло често потребна. (конст. копије)
- Операција доделе вредности нам је, такође, изузетно често потребна, али о томе мало касније. (оператор доделе)

```
void foo() {
    int x;
    x = 5;
    x = y;
}
void foo() {
    Token x;
    x = y;
}
```

Тип

- Скуп операција
- Које операције нам требају?
- Операција стварања променљиве нам је увек потребна. (конструктор)
- Операција стварања копије нам је врло често потребна. (конст. копије)
- Операција доделе вредности нам је, такође, изузетно често потребна, али о томе мало касније. (оператор доделе)
- ?

```
void foo() {
    int x;
    x = 5;
}
```

<=>

```
void foo() {
    int x = 5;
}
```

```
void foo() {
    int x(5);
}
```

```
void foo() {
    int x{5};
}
```

```
void foo() {
  Token x('8');
}
```

```
void foo() {
  Token x{'8'};
}
```

Тип

- Скуп операција
- Које операције нам требају?
- Операција стварања променљиве нам је увек потребна. (конструктор)
- Операција стварања копије нам је врло често потребна. (конст. копије)
- Операција доделе вредности нам је, такође, изузетно често потребна, али о томе мало касније. (оператор доделе)
- Операције иницијализације у разноразним облицима су исто врло корисне. (разне врсте конструктора)

```
void foo() {
    int x;
    x = 5;
}
```

<=>

```
void foo() {
    int x = 5;
}
```

```
void foo() {
    int x(5);
}
```

```
void foo() {
    int x{5};
}
```

```
void foo() {
  Token x('8');
}
```

```
void foo() {
  Token x{'8'};
}
```

Тип

- Скуп операција
- Које операције нам требају?
- Операција стварања променљиве нам је увек потребна. (конструктор)
- Операција стварања копије нам је врло често потребна. (конст. копије)
- Операција доделе вредности нам је, такође, изузетно често потребна, али о томе мало касније. (оператор доделе)
- Операције иницијализације у разноразним облицима су исто врло корисне. (разне врсте конструктора)

```
class Token {
public:
    Token(char ch, double val) : kind(ch), value(val) {}
    Token(char ch) : kind(ch), value(0.0) {}
    Token() : kind('x'), value(0.0) {}
    char kind;
    double value;
};
```

Тип

- Скуп операција
- Које операције нам требају?
- Операција стварања променљиве нам је увек потребна. (конструктор)
- Операција стварања копије нам је врло често потребна. (конст. копије)
- Операција доделе вредности нам је, такође, изузетно често потребна, али о томе мало касније. (оператор доделе)
- Операције иницијализације у разноразним облицима су исто врло корисне. (разне врсте конструктора)

```
class Token { // а може и овако
public:
    Token(char ch, double val) : kind(ch), value(val) {}
    Token(char ch) : kind(ch) {}
    Token() {}
    char kind = 'x';
    double value = 0.0;
};
```

Тип

- Обратити пажњу!!!!

```
void foo() {
    int x;
    x = 5;
}
```

<=>

```
void foo() {
    int x = 5;
}
```

- Али:

```
void foo() {
    int x = 5;
}
```

<=>

```
void foo() {
    int x(5);
}
```

<=>

```
void foo() {
    int x{5};
}
```

Постоје ситне разлике између последњег и прва два случаја, али нећемо се на то освртати на овом предмету.

Тип – Специјалне функције

- Операција стварања променљиве нам је увек потребна. (конструктор)
- Операција стварања копије нам је врло често потребна. (конст. копије)
- Операције иницијализације у разноразним облицима су исто врло корисне. (разне врсте конструктора)
- Операција доделе вредности нам је, такође, изузетно често потребна. (оператор доделе копије)
- Операција уништења променљиве нам је увек потребна (деструктор)

```
class Token {
public:
    Token(char ch, double val) : kind(ch), value(val) {}
    Token(char ch) : kind(ch) {}
    Token() {}
    Token(const Token& x) : kind(x.kind), value(x.value) {}
    ~Token() {}
    char kind = 'x';
    double value = 0.0;
};
```

Тип – Специјалне функције

- Операција стварања променљиве нам је увек потребна. (конструктор)
- Операција стварања копије нам је врло често потребна. (конст. копије)
- Операције иницијализације у разноразним облицима су исто врло корисне. (разне врсте конструктора)
- Операција доделе вредности нам је, такође, изузетно често потребна. (оператор доделе копије)
- Операција уништења променљиве нам је увек потребна (деструктор)

- Ове операције су толико често потребне да се стварају њихове подразумеване дефиниције и без нашег петљања.

```
class Token {
public:
    Token() {}
    Token(const Token& x) : kind(x.kind), value(x.value) {}
    ~Token() {}
    char kind = 'x';
    double value = 0.0;
};
```

Класе

- Класа је кориснички дефинисан тип.
- Класа је сложен тип, у смислу да се састоји од мањих јединица које се зову „чланови“

```
class X {
public:
  // јавни чланови, представљају спрегу класе са спољним светом
  // сви им могу приступити
  // функције, типови, променљиве...
private:
  // приватни чланови, тичу се само њене унутрашње организације
  // може им се приступити само из класе
  // функције, типови, променљиве...
};
```

Класе

- Чланови класе су подразумевано приватни:

```
class X {
      int mf();
      // …
};
```

- ... је исто што и:

```
class X {
private:
      int mf();
      // …
};

X x; // променљива x типа X
int y = x.mf(); // грешка: mf је приватно
```

Структуре

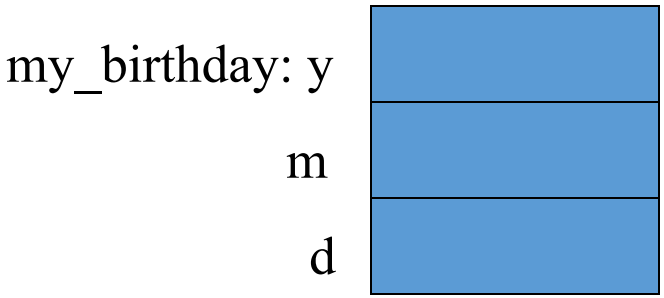
- Структура је класа код које су чланови подразумевано јавни:

```
struct X {
      int m;
      // …
};
```

- … је исто што и:

```
class X {
public:
      int m;
      // …
};
```

Тип Date

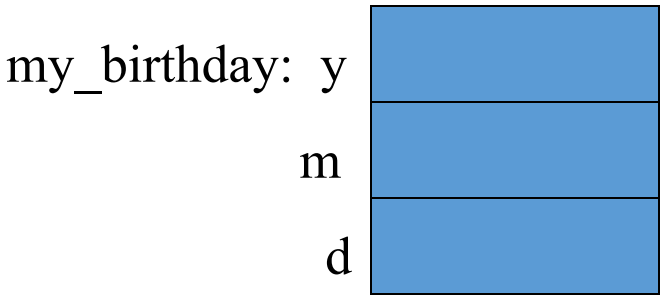



```
struct Date {
  int y, m, d;    // year, month, day
};

Date my_birthday;      // променљива типа Date (“објекат”)

my_birthday.y = 12;
my_birthday.m = 30;
my_birthday.d = 1950;  // 1950. дан?!
```

Тип Date




```
struct Date {
  int y, m, d;
};

Date my_birthday;

// помоћне функције:

void init_day(Date& dd, int y, int m, int d);
  // провери конзистентност датума и иницијализуј променљиву dd

void add_day(Date& dd, int n);
  // одређује који је датум за n дана

init_day(my_birthday, 12, 30, 1950);
  // грешка у извршавању: датум није ваљан
```

Шта све типови у Це++-у имају

Це++ језик тежи да омогући кориснику да направи своје типове који су исте изражајности као и уграђени типови.

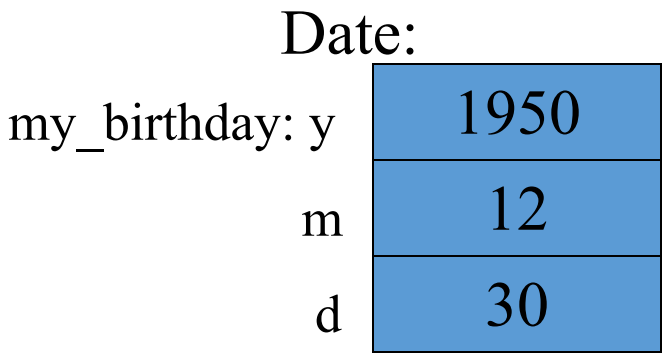
```
int x = 5; // иницијализација
x = 4; // додела вредности

const int t = 5; // може иницијализација
t = 6; // не може додела;

void foo(int y);
int x;
foo(x);

int x = 5; // иницијализација
int x(5); // иницијализација
int x{5}; // иницијализација – C++11
```

Тип Date




```
struct Date {
  int y, m, d;
  Date(int y, int m, int d);
    // Конструктор: проверава ваљаност датума и иницијализује
  void add_day(int n); // одређује датум за n дана
  // функције које су чланови класе зовемо по некад методе
};

// …
Date my_birthday; // грешка у превођењу: нема празне иницијализације
Date my_birthday(12, 30, 1950); // грешка у извршавању
Date my_day(1950, 12, 30);  // ОК
my_day.add_day(2);          // 1. 1. 1951.
```

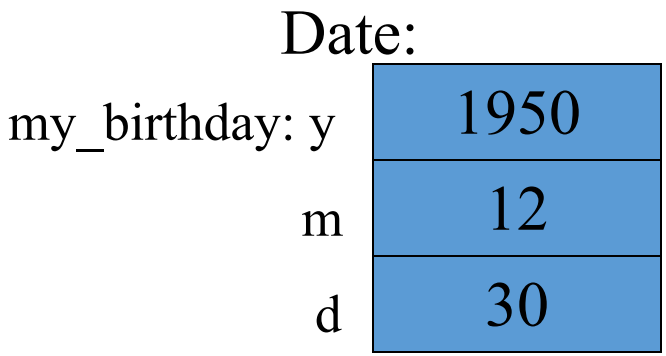
Функције и методе (Слободне функције и функције чланице)

```
struct Date {
  int y, m, d;
  void add_day(int n) {
    ...
    m = n;
    ...
  }
};

void add_day(Date& dd, int n) {
  ...
  dd.m = n;
  ...
}
```

```
Date x;
x.add_day(5);
add_day(x, 5);
```

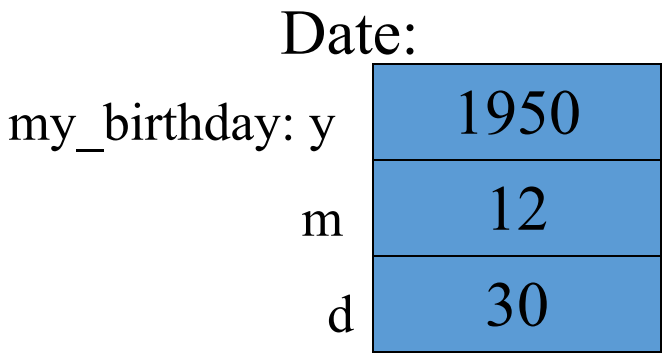
Тип Date




```
struct Date {
  int y, m, d;
  Date(int y, int m, int d);
    // Конструктор: проверава ваљаност датума и иницијализује
  void add_day(int n); // одређује датум за n дана
  // функције које су чланови класе зовемо методе
};

// …
Date my_birthday; // грешка у превођењу: нема празне иницијализације
Date my_birthday(12, 30, 1950); // грешка у извршавању
Date my_day(1950, 12, 30);  // ОК
my_day.add_day(2);          // 1. 1. 1951.
my_day.m = 14;              // упс! датум опет није ваљан
```

Тип Date




```
class Date {
  int y, m, d;
public:
  Date(int y, int m, int d);
  void add_day(int n);

  int month() { return m; }
  int day() { return d; }
  int year() { return y; }
  // обично се за овакве методе каже да су гет методе
};

// …
Date my_birthday(1950, 12, 30);
cout << my_birthday.month() << endl;
my_birthday.m = 14; // грешка: Date::m је приватан члан
```

Функције и методе

```
class Date {
  int y, m, d;
public:
  void add_day(int n) {
    ...
    m = 5;
    ...
  }
};

void add_day(Date& dd, int n) {
  ...
  dd.m = 5; // ово више не пролази!
  ...
}
```

```
Date x;
x.add_day(5);
add_day(x, 5);
```

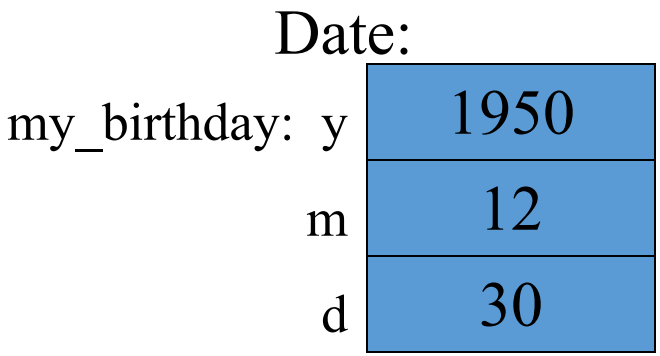
Функције и методе

```
class Date {
  int y, m, d;
public:
  friend void add_day(Date& dd, int n);
  void add_day(int n) {
    ...
    m = 5;
    ...
  }
};

void add_day(Date& dd, int n) {
  ...
  dd.m = 5; // опет пролази
  ...
}
```

```
Date x;
x.add_day(5);
add_day(x, 5);
```

Тип Date

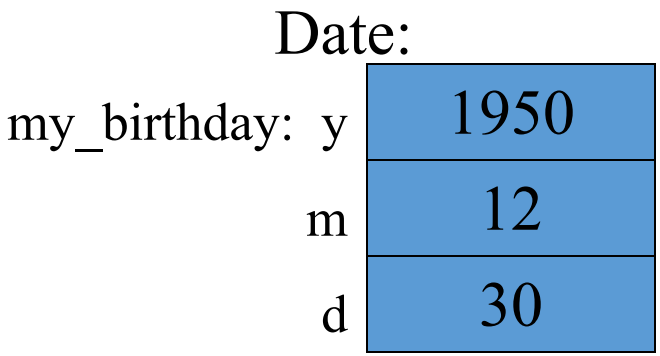



```
class Date {
public:
  // Често се јавни чланови пишу прво, да спрега буде видљивија
  Date(int yy, int mm, int dd);
  void add_day(int n);
  int month();
  // …
private:
  int y, m, d;
};

Date::Date(int yy, int mm, int dd) // дефиниција методе
  : y(yy), m(mm), d(dd) { /* … */ };

void Date::add_day(int n) { /* … */ }; // дефиниција методе
```

Тип Date




```
class Date {
public:
  Date(int yy, int mm, int dd);
  void add_day(int n);
  int month() { return m; }
  // …
private:
  int y, m, d;
};

int month() { return m; } // коју грешку ће пријавити компајлер?
int Date::season() { /* … */ } // грешка: season не постоји у Date
```

Тип Date

```
class Date {
public:
  class Invalid { };
  Date(int y, int m, int d);
  // …
private:
  int y, m, d;
  bool check(int y, int m, int d);
};

Date::Date(int yy, int mm, int dd) : y(yy), m(mm), d(dd)
{
  if (!check(y, m, d)) throw Invalid();
}
```

Набројиви типови - енумерације

- Енумерација је врло једноставан кориснички тип који је одређен скупом својих вредности.
- То је други начин да се дефинишу вредности које објекат неког типа може да има (први начин је композиција)

- Пример:

```
enum Month {
  jan=1, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec
};

Month m = feb;
m = 7;                  // грешка, не може int у Month
int n = m;              // обрнуто може
Month mm = Month(7);    // и ово може
```

Набројиви типови - енумерације

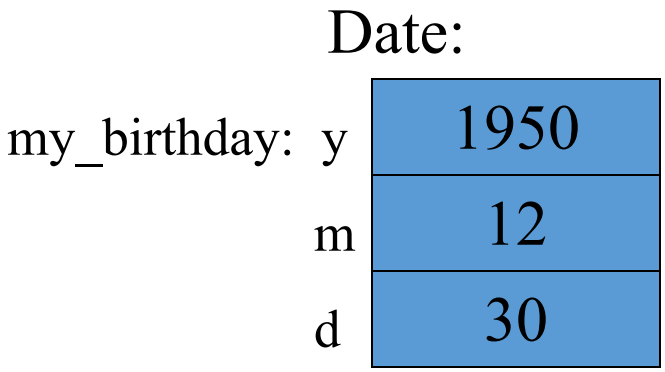
- У основи, сваком симболу из енумерације се придружује неки цео број (ког типа?)
- Подразумевано:

```
// први симбол има вредност 0,
// следећи симбол је вредност претходног + 1
enum { konj, svinja, pile }; // konj==0, svinja==1, pile==2
```

- Вредност се може експлицитно придружити:

```
enum { jan=1, feb, march /* … */ }; // feb==2, march==3
enum stream_state { good=1, fail=2, bad=4, eof=8 };
int flags = fail + eof; // flags==10
stream_state s = flags; // грешка
stream_state s2 = stream_state(flags);
```

Тип Date




```
class Date {
public:
  enum Month {
       jan=1, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec
  };
  Date(int y, Month m, int d);
  // …
private:
  int y;
  Month m;
  int d;
};

Date my_birthday(1950, 30, Date::dec); // грешка, током превођења
Date my_birthday(1950, Date::dec, 30); // ОК
```

Const

```
const int x = 5;
x = 6; // грешка у превођењу
increment(x); // грешка???
// зависи од тога шта ће increment да ради са x, тј. да ли га
  прима по вредности или референци или const референци
```

```
class Date {
public:
        // …
        int day(){ return d; }
};
```

```
Date d(2000, Date::jan, 20);
const Date cd(2001, Date::feb, 21);
```

```
cout << d.day() << " - " << cd.day() /*грешка!*/ << endl;
d.add_day(1);
cd.add_day(1); // грешка
```

Функције и методе

```
class Date {
  int y, m, d;
public:
  friend int day(Date& dd);
  int day() { return d; }
};

int day(Date& dd) {
  return dd.d;
}
```

```
const Date x;
x.day();
day(x);
```

Функције и методе

```
class Date {
  int y, m, d;
public:
  friend int day(const Date& dd);
  int day() const { return d; }
};

int day(const Date& dd) {
  return dd.d;
}
```

```
const Date x;
x.day();
day(x);
```

Функције и методе

```
class Date {
  int y, m, d;
public:
  friend int day(const Date& dd);
  friend int day(Date& dd);
  int day() const;
  int day();
};
```

```
const Date x;
x.day();
day(x);
```

```
Date x;
x.day();
day(x);
```

- const је део потписа.

Класе

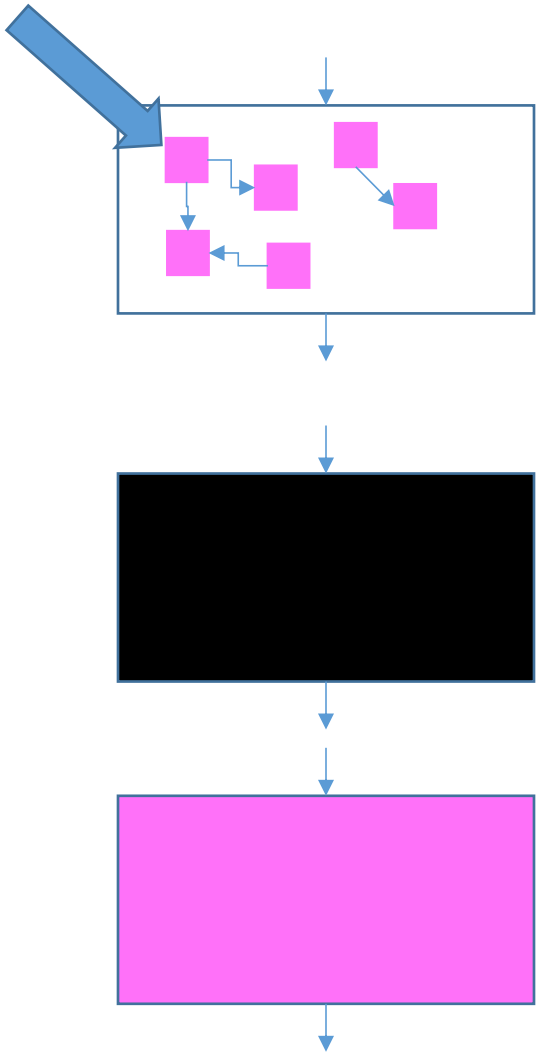
- Пројектовање класе је пројектовање новог типа
- Пожељне особине: природна синтакса у коришћењу, интуитивна семантика и ефикасна имплементација (једна или више).
- Шта чини добру спрегу?
  - Минимална
    - Што мања то боља...
  - Потпуна
    - ... али не премала
  - Типски безбедна
  - Ваљана са становишта константности

Класе

- Кључне операције:
  - Подразумевани конструктор (своди се на празан код)
    - Поништава се ако се декларише било који други конструктор
  - Конструктор копије (подразумевано се своди на копирање података)
  - Додела копије (подразумевано се своди на копирање података)
  - Деструктор (подразумевано се своди на празан код)

Једна подела функција

Основни елемент језика



Корисничка функција

Изражава се у Це++-у.
Свако може да је пише и свако често то ради.

Библиотечка функција

Изражена је директно у машинском коду (асемблеру).
Пишу их творци тих библиотека, писци системског софтвера и сл.

Уграђена функција (intrinsic, builtin)

Изражена је на начин својствен компајлеру.
Пишу их радници на компајлеру.

Математичке нотације

- Префиксна (Пољска)

```
- + a b c
-(+(a, b), c)
minus(plus(a, b), c)
```

- Инфиксна

```
a + b – c
a plus b minus c
```

- Постфиксна (Обрнута пољска)

```
a b + c –
```

Помоћне функције

```
bool dates_equ(const Date& a, const Date& b) {
  return a.year() == b.year()
       && a.month() == b.month()
       && a.day() == b.day();
}

if (dates_equ(date1, date2)) ...

if (date1 dates_equ date2) ...

if (date1 == date2) ...

bool operator==(const Date& a, const Date& b) {
  return a.year() == b.year()
       && a.month() == b.month()
       && a.day() == b.day();
}
```

Операцијске функције – „преклапање“ операција

```
bool operator==(const Date& a, const Date& b) {
  return a.year() == b.year()
    && a.month() == b.month()
    && a.day() == b.day();
}

enum Month {
 jan=1, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec
};

Month operator++(Month& m) {
 m = (m == dec) ? jan : Month(m+1);
 return m;
}

Month m = nov;
++m;  // децембар
++m;  // јануар
```

Операцијске функције – „преклапање“ операција

- Могу се користити само постојеће операције
  - *Нпр.:* **+ - * / % [] () ^ ! & < <= > >=**
- Број операнада над којима та операција ради не може се мењати
  - рецимо, нема унарног **<=**, ни бинарног **!**
- Тип бар једног параметра мора бити кориснички дефинисан тип
  - `int operator+(int,int); // грешка`
  - `Vector operator+(const Vector&, const Vector &);`
- Савети:
  - Операције дефинишите само у складу са њиховим очекиваним значењем. **+** да буде сабирање, или унија итд., ***** множење, или пресек...
  - Не дефинишите операције осим ако немате јасну потребу.

Још неке ствари које могу бити дефинисане за кориснички тип

- Кориснички дефинисани литерали. Нпр.:
  - Време: **2h+10m+12s+123ms+3456ns**
  - Комплексни бројеви: **2+4i**
  - Знаковни низови (стрингови): **“pera”s**