C++

Robotex 2014

TTÜ Robotiklubi

Tallinn University of Technology Robotics Club

Konstandid

Konstantide märkimiseks on

const

```
const int number= 5;
```

static on mitme tähendusega

static

Luues funktsiooni sees kohaliku staatilise muutuja jätab see enda väärtuse meelda (erinevalt teistest kohalikest muutujatest), järgmine kord funktsiooni välja kutsudes on selle väärtus sama mis eelmine kord.

```
int liidaYks(void)
{
  static int n = 1;
  return ++n;
}
```

Lihtsam viis luua ise objekte

struct

```
struct fail {
    int lines;
    int word_count;
    map<string, int> words;
};

struct fail faili_andmed;

faili_andmed.lines = 10;
```

union

Nagu struct aga hoiab korraga ainult ühe muutuja väärtust.

```
union Hind {
    float eur;
    float usd;
    float gbp;
} Hind;
```

union Hind raamatuHind;

raamatuHind.eur = 10;

Võidaldab defineerida nummerdatud loendeid, on int tüüpi enum sisemiselt.

```
enum Laevad {Reisilaev = 2, Tanker, Jaht}

enum Laevad laevaLiik;

laevaLiik = Jaht;
```

Uue tüübi loomine.

typedef

Kui templatedega objektide tüübid liiga tüütult pikaks

lähevad siis hea koodi loetavamaks teha

```
typedef map<string, int> Sonad;

Sonad sonad;

for (Sonad::iterator it = sonad.begin(); it != sonad.end(); it++) {

...

}
```

Lühidalt öeldes - funktsioonid mis kutsuvad iseend välja.

rekursioon

```
int fact(int n)
{
    if (n == 1)
        return 1;
    else
        return n*fact(n-1);
}
```

Peab olema tingimus mil rekursioon lõppeb (n == 1).

Klassid

Programmeerija poolt loodud uued andmetüübid - klassid.

Võimaldavad hoida andmeid, luua funktsioone mis nendega tegelevad ja piirata ligipääsu andmetele

```
class Vektor
{
public:
    float algusX;
    float algusY;
    float loppX;
    float loppY;
    float pikkus();
}
```

Liikmetele juurdepääs punktiga

```
Vektor v;
v.algstX = 0;
v.algusY = 0;
v.loppX = 1;
v.loppY = 1;
```

Välja arvatud siis kui tegu viidaga.

```
Vektor *v;
v = new Vektor();

v->algusX = 0;
v->algusY = 0;
```

Klassid saavad sisaldada teisi klasse, või ükskõik mis tüüpe

```
class Punkt
{
public:
    float x;
    float y;
};

class Vektor
{
public:

    Point algus;
    Point lopp;
};
```

Koodi struktureerimine.

Klassid

Klasside nimed Suure algustähega.

Klasside puhul pannakse deklaratsioon header faili,

definitsioon sama nimega .cpp faili.

Meil oleks siis fail Vektor.h ja Vektor.cpp

Vektor.cpp alguses on #include "Vektor.h"

Funktsioonide argumentidena kasutamine.

```
liidaYks(Punkt p)
{
    p.x += 1;
    p.y += 1;

}
```

Ei tööta

```
liidaYks(Punkt &p)
{
    p.x += 1;
    p.y += 1;
}
```

Töötab

.h fail

```
class Punkt
{
public:
    float x;
    float y;
};

class Vektor
{
public:

    Punkt algus;
    Punkt lopp;
    float pikkus();
};
```

Klassid

Meetodid

.cpp

Klassid

Meetodid

```
#include <cmath>

#include "Vektor.h"

float Vektor::pikkus()
{
   return sqrt(pow(lopp.x - algus.x, 2) + pow(lopp.y-lopp.x, 2));
}
```

Klassid

Konstruktor

Käsitsi klassi liikmete väärtustamine (initsialiseerimine) on tüütu. Kuna seda on vaja naguinii iga kord teha siis.. on olemas igal klassil meetod mida nimetatakse konstruktoriks, sel on alati sama nimi mis klassil endal ja ei tagasta midagi, isegi mitte void.

```
class Punkt
{
    Punkt();
    ...
}

Punkt::Punkt()
{
    x = 0;
    y = 0;
}
```

Klassid

Koopia konstruktor

Eraldi konstruktori liik - copy constructor.

Kasutatakse objektide omistamisel.

```
class Tudeng
{
public:
    float kkh;
    char *nimi;
}
int main()
{
   Tudeng t1;
   t1.nimi = "tudeng1";
   t1.kkh = 4.0;
   Tudeng t2 = t1;
   cout << t1.nimi << endl;
   cout << t2.nimi << endl;
   t2.nimi = "teine";
   cout << t1.nimi << endl;
}
```

Klassid

Koopia konstruktor

Eraldi konstruktori liik - copy constructor.

Kasutatakse objektide omistamisel.

```
class Tudeng
{
public:
    float kkh;
    char *nimi;
    Tudeng (Tudeng &t)
    {
        kkh = t.kkh;
        nimi = strdup(t.nimi);
    }
}
```

Klassid

Konstruktor

Overload.

C++ keeles saab kirjutada funktsioone ja klasside meetodeid mille argumendid on erinevad aga nimed samad.

```
class Punkt
{
    Punkt();
    Punkt(float x, float y);
    ...
}

Punkt::Punkt(float x_, float y_)
{
    x = x_;
    y = y_;
}
```

Saab ka lühemalt.

Kui on vaja ainult muutujaid omistada saab kasutada initializer listi.

```
Punkt::Punkt(float x_, float y_) : x(x_), y(y_)
{
    // Muu kood siia
}
```

Objektorienteeritud programeerimine

OOP

Andmete kapseldamine (encapsulation)

public, protected, private

Pärimine (inheritance)

Polümorfism (polymorphism)

Pärimine

Alamat objekti saab kasutada alati seal, kus oodatakse ülemat.

```
class Masin
{
    public:
    float mass, kiirus;
}
class CV90 : public Masin
{
    public:
    int meeskonnaSuurus;
}
void teeMidagi(Masin m);

CV90 m;

teeMidagi(m);
```

Meetodite ülekirjutamine, **virtual**

Ülemklass saab lubada oma meetodide üle kirjutamise alamklasside poolt, selleks tuleb lisada meetodi deklaratsiooni juurde märksõna **virtual**

OOP

virtual

```
class CV90 : public Masin
{
    string kirjeldus();
}

string CV90::kirjeldus()
{
    string s;
    s += "mass " + mass + ", kiirus "+ kiirus + ", meeskond " + meeskond;
    return s;
}
```

OOP

abstraktsed klassid

Võib teha ka klasse, milles osa meetodeid on jäetud implementeerimata. Sellise klassi objekte ei ole võimalik luua, kasutamiseks peab neist pärima ning lisama puudu oleva funktsionaalsuse alamklassis.

```
class Masin
{
    float mass, kiirus;
    virtual string kirjeldus() = 0;
}

int main()
{
    Masin m; // VIGA
}
```

Tõnis Märtmaa

e-mail: robotiklubi@robotiklubi.ee

tel. +372 53408660