

# Objektorientētā programmēšana

## 1. laboratorijas darbs

# Abstrakcija un iekapsulēšana

Dr. sc. ing. Pāvels Rusakovs

Dr. sc. ing. Vitālijs Zabiņako

Mg. sc. ing. Andrejs Jeršovs

Mg. sc. ing. Pāvels Semenčuks

Mg. sc. ing. Vladislavs Nazaruks

*RTU 2020*

# Abstrakcija un iekapsulēšana

Klases fragments: privātie atribūti un pārlādētie konstruktori

```
class CoordPoint {  
    private:  
        int X;  
        int Y;  
    public:  
        CoordPoint();           // konstruktors  
        CoordPoint(int, int);  // konstruktors  
        ...  
};
```

## Statiskie un dinamiskie objekti

```
CoordPoint CP1, CP2(1, 2), CP3 = CoordPoint(3, 4),  
    *CP4 = new CoordPoint(5, 6), *CP5;  
...  
CP5 = new CoordPoint(7, 8);  
// objekts CP5 izveidots pēc deklarēšanas  
CP1, CP2, CP3 – statiskie objekti; CP4, CP5 – dinamiskie objekti.
```

### Dinamisku objektu iznīcināšana

```
delete CP4; // destruktora izsaukums  
delete CP5; // destruktora izsaukums
```

---

### Destruktors. Objektorientētā izvade

```
~CoordPoint() {  
    cout << "Message from the \"CoordPoint\" - destroyed!" <<  
        endl;    // 'endl' C++ valodā ir '\\n' analogs C valodā  
}
```

---

### Metožu izsaukumi statiskos un dinamiskos objektos

```
CP1.Print();           // izsaukums no statiska objekta  
(*CP4).Print();       // izsaukums no dinamiska objekta  
CP4->Print();          // izsaukums no dinamiska objekta
```

---

### Metožu izsaukumi (paplašinātā sintakse)

```
CP1.CoordPoint::Print(); // <objekts>.<klase>::<metode>  
(*CP4).CoordPoint::Print(); // <objekts>.<klase>::<metode>  
CP5->CoordPoint::Print(); // <objekts>.<klase>::<metode>
```

### Iespējamās **problēmas** darbā ar *dinamiskiem* objektiem

1. Objekts **netika izveidots** pirms tā izmantošanas.

```
CoordPoint *CP;
```

```
...
```

```
// objekts nav izveidots ar new palīdzību
```

```
CP->Print();    // vēl netika izpildīts konstruktors
```

Dažos kompilatoros var būt *brīdinājums*:

**Warning LAB\_1.CPP 47: Possible use of 'CP' before definition in function main()**

*Iespējamais rezultāts:*

```
X = 21829, Y = -4981
```

---

2. **Neizveidotā objekta dzēšana.**

```
CoordPoint *CP;
```

```
...
```

```
// objekts nav izveidots ar new palīdzību
```

```
delete CP;
```

*Iespējamais rezultāts:*

**General Protection Exception  
0x1BDF:0x3309**

**LAB\_1(1) 0x1BDF:0x3309 Processor Fault**

3. Izveidotais objekts tika **iznīcināts divas reizes**.

```
CoordPoint *CP = new CoordPoint();
```

```
...
```

```
delete CP;
```

```
...
```

```
delete CP;
```

*Iespējamais rezultāts: lietotne var “uzkārties”.*

---

4. Izveidotais objekts **netika iznīcināts**.

```
{
```

```
    CoordPoint *CP = new CoordPoint();
```

```
    ...
```

```
    // nav "delete CP;"
```

```
}
```

Rezultāts: *atmiņas noplūde*.

---

5. Adresēts **jau izdzēstais** objekts.

```
delete CP;
```

```
CP->Print();
```

Rezultāts: *neprognozējams*.

## Inicializatori

```
CoordPoint::CoordPoint() : X(0), Y(0) { // X=0, Y=0  
}
```

## Rādītājs uz esošu objektu

```
void SetX(int X) {  
    this->X = X; // atribūta un parametra vārds sakrīt  
}
```

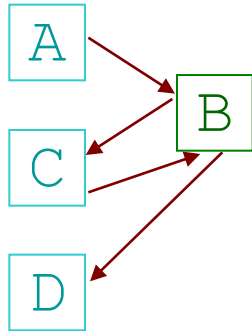
## Iegultās (**inline**) funkcijas: divi veidošanas paņēmieni

```
class CoordPoint {  
    ...  
    void SetX(int X) { // Funkcija-metode definēta  
        this->X = X; // klases CoordPoint iekšā.  
    } // Paņēmiens der tikai metodēm.  
};  
  
inline void CoordPoint::SetY(int Y) { // "inline"  
    this->Y = Y; // definīcijā  
}
```

## Iegulto funkciju efekts

Lai **A**, **B**, **C**, **D** ir koda bloki. Ir darbību secība: **A**, **B**, **C**, **B**, **D**.

B *nav* iegultā funkcija



B *ir* iegultā funkcija



---

## Atribūtu aizsardzība no izmaiņām

```
int GetX() const {  
    Y = 2; // Kompilācijas kļūda ! Y ir atribūts.  
    return X;  
}
```

Funkcija **netiks** noformētā kā iegultā, ja...

1. Funkcijā ir *cikli*.

```
inline void ClearBuffer() {  
    while (kbhit())  
        getch();  
}
```

**Warning TEST.CPP 43: Functions containing while are not expanded inline**

---

2. Funkcijā ir operators **goto**.

```
inline void GetChar() {  
    char c=getch();  
    if (c=='n')  
        goto n;  
    ...  
}
```

**Warning TEST.CPP 45: Functions containing goto are not expanded inline**

---

3. Dažos citos gadījumos.



### Darbs ar rakstzīmju virknēm (C++ 4.5)

```
#include <string.h>
#include <cstring.h>
...
char* S1;
char S2[20];
string S3;    // klase no cstring.h

...
S1 = "C++";
strcpy(S2, "C++");
S3 = "C++";

...
cout << "TEXT:    " << S1 << " " << S2 << " " <<
    S3 << endl;
cout << "LENGTH: " << strlen(S1) << " " <<
    strlen(S2) << " " << S3.length() << endl;
...
```

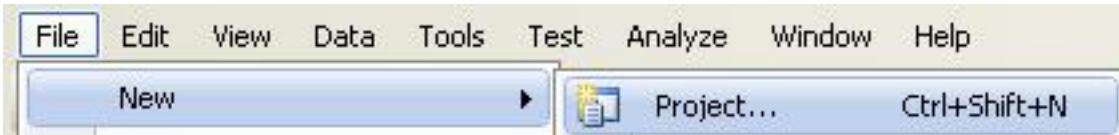
### Darbs ar rakstzīmju virknēm (C++ 5.x un jaunākās versijas)

```
#include <string>           // bez ".h"
...
using namespace std;       // nosaukumvieta

void main(void) {
    ...
    string S3;
    ...
    S3 = "C++";
    ...
    cout << "TEXT:   " << S3 << endl;
    cout << "LENGTH: " << S3.length() << endl;
}
```

## Darbs ar Microsoft Visual Studio (MVS)

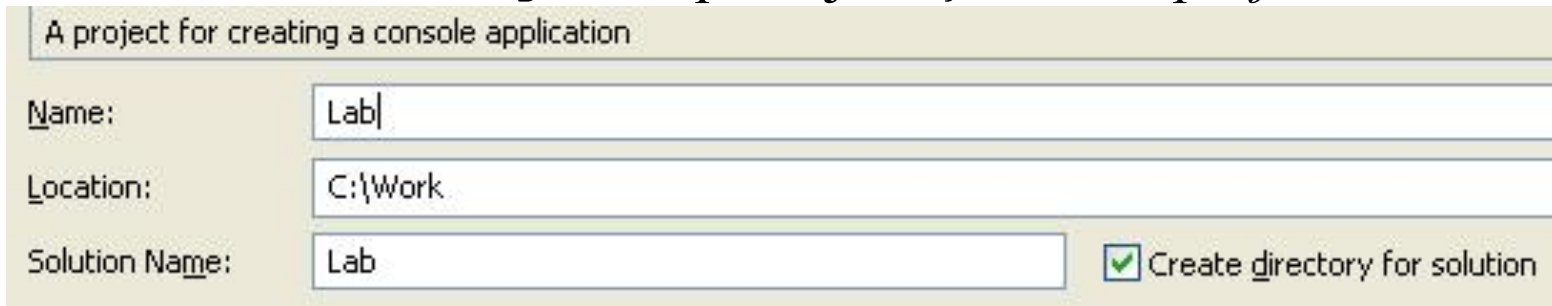
1. File → New → Project



2. Visual C++ → CLR Console Application



3. Formas New Project apakšējā daļā norādīt *projekta vārdu*.

A screenshot of the 'New Project' dialog box. The 'Name' field is set to 'Lab', the 'Location' is 'C:\Work', and the 'Solution Name' is 'Lab'. The 'Create directory for solution' checkbox is checked.

4. Rezultāts: mape C : \Work\Lab ar vairākiem failiem.

### Automātiski izveidotās programmas Lab.cpp teksts

```
// Lab.cpp : main project file.  
  
#include "stdafx.h"  
  
using namespace System;  
  
int main(array<System::String ^> ^args)  
{  
    Console::WriteLine(L"Hello World");  
    return 0;  
}
```

*Apturēt* programmas izpildi var:

```
Console::ReadLine();
```

### Citi MVS programmas varianti (tikai *jaunas* konstrukcijas)

```
#include <iostream>
```

```
using namespace std;
```

```
int main(array<System::String ^> ^args)
{
    cout << "Hello World";
    ...
}
```

---

```
#include <iostream>
```

```
int main(array<System::String ^> ^args)
{
    std::cout << "Hello World";
    ...
}
```