ФАКУЛТЕТ ЗА ЕЛЕКТРОТЕХНИКА И ИНФОРМАЦИСКИ ТЕХНОЛОГИИ

Не објектно-ориентирани новини во C++

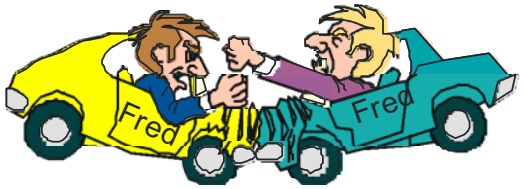
–Податочни структури и програмирање–

Новини во C++ (не OO)

- **Новини во C++** кои **не се однесуваат директно на објектно-ориентираното** програмирање:
  - нов вид на коментари // ;
  - нови вградени типови bool, wchar_t, long long, long double, (string);
  - namespaces (простори на имиња);
  - iostream библиотека;
  - декларации на променливите било каде во програмата;
  - употреба на const променливи за димензија на поле;
  - scope оператор :: ;
  - подразбирани аргументи на функциите;
  - преоптоварување на функции (и оператори);
  - inline функции;
  - референци &

Namespaces (простори на имиња)

- Сите симболи во C++ се дефинирани на ниво на поголем контекст наречен **простор на имиња** (namespace)
- Во големите проекти **колизијата на имиња** станува сериозен **проблем**. Колизијата на имиња е предизвикана од две различни глобални променливи, функции или класи во различни програмски модули кои имаат исти имиња.
- Просторите на имиња обезбедуваат поделба на програмата на делови, без колизија на имињата
- Се во стандардната библиотека е во просторот на имиња **`std`**
- Директивата **`using namespace std`** му кажува на компајлерот сите имиња да ги бара во овој простор на имиња



iostream

- од 1995 година **сите C++ компајлери** се дистрибуираат **со стандардна библиотека**
- **iostream** е стандардна библиотека за **влезно-излезни текови**

```
// Old way (deprecated)
#include <iostream.h>
void main()
{
      cout << "Hello, world " << endl;
}

// New standard
#include <iostream>
using namespace std;
void main()
{
      cout << "Hello, world " << endl;
}
#include <iostream>
void main()
{
      std::cout << "Hello, world " << std::endl;
}
```

```
#include <iostream>
using namespace std;
int main()
{
      int i; char c; float f;

      cout << "Vnesi 2 broja i eden znak: ";
      cin >> i >> f >> c;
      cout << "Vneseni bea: " << i << ' ';
      cout << f << ' ' << c << endl;
      return 0;
}
```

```
Vnesi 2 broja i eden znak: 5 6.23 x
Vneseni bea: 5 6.23 x
```

Standard end line (endl)

- The **endl** is a predefined object of **ostream** class.
- It is used to insert a new line characters and flushes the stream.

```
#include <iostream>
using namespace std;
int main( ) {
    cout << "C++ Tutorial";
    cout << " Javatpoint"<<endl;
    cout << "End of line"<<endl;
}
```

Output:

```
C++ Tutorial Javatpoint
End of line
```

Декларации

- Во C++ **променливите** може да **се декларираат на било кое место во програмата** пред нивната прва употреба

```
#include <iostream>
using namespace std;
int main()
{
	int i = 5;
	for (int lineno = 0; lineno < 3; lineno++)
	{
		cout << "This is line number " << lineno;
		int temp = 22;
		cout << " and temp is " << temp << endl;
	}
	// lineno accessible ???
	int final = 33;
	cout << " ... and final is " << final << endl;
}
```

const и enum наместо #define

- Во C++ **константните целобројни променливи** може да се употребат **за одредување на бројот на елементи при декларирање полиња**. Константите наместо со `#define` препорачано е да се декларираат со `enum`

```
#include <stdio.h>
#define SIZE 5
#define NE 0
#define DA 1
#define MOZEBI 2

int main()
{
      int cs[SIZE];
      printf("The size of cs is %d\n",
            sizeof(cs));
      cs[0] = NE;
      cs[1] = DA;
      cs[2] = MOZEBI;
      printf("%d%d%d\n", cs[0],
            cs[1], cs[2]);
      return 0;
}
```

```
#include <iostream>
using namespace std;

enum odgovor { NE, DA, MOZEBI };
int main()
{
      const int SIZE = 5;
      odgovor cs[SIZE];
      cout << "The size of cs is "
            << sizeof cs << endl;
      cs[0] = NE;
      cs[1] = DA;
      cs[2] = MOZEBI;
      cout << cs[0] << cs[1] << cs[2] << endl;
      return 0;
}
```

```
The size of cs is 20
012
```

Разлики во однос на C (1)

- `NULL` покажувач наспроти 0 покажувач
  - во C++ се третираат исто

- Стриктна проверка на типови во C++
  - мора да биде наведен **прототип на секоја функција пред таа** да може **да се повика** и повикот **мора** да одговара на прототипот
  - прототиповите `void f();` и `void f(void);` во C++ се третираат исто

Разлики во однос на C (2)

- Нов начин на кастирање
  - иако и C начинот **(typename)expression** работи, во C++ може да се употреби и **typename(expression)** што и не е вистинско кастирање
    пр. `float(5)/2` наместо `(float)5/2`

- во C++ се воведуваат **четири нови видови на кастирање** кои треба да се употребуваат наместо C стилот
  - `static_cast<type>(expression)`
  - `const_cast<type>(expression)`
  - `reinterpret_cast<type>(expression)`
  - `dynamic_cast<type>(expression)`

```
#include <iostream>
using namespace std;
int main()
{
    float f = 3.5;
    int a = f; // this is how you do in C
    int b = static_cast<int>(f);
    cout << b;
}
```

scope оператор ::

```
#include <iostream>
using namespace std;
int ccc = 987;
int main()
{
      int ccc = 123;
      cout << "main   ccc=" << ccc << endl;
      cout << "Global ccc=" << ::ccc << endl;
      {
            int ccc = 456;
            cout << "Inner  ccc=" << ccc << endl;
            cout << "Global ccc=" << ::ccc << endl;
            {
                  int ccc = 789;
                  cout << "Innest ccc=" << ccc << endl;
                  cout << "Global ccc=" << ::ccc << endl;
                  ::ccc = 321;
            }
            cout << "Inner  ccc=" << ccc << endl;
            cout << "Global ccc=" << ::ccc << endl;
      }
      cout << "main   ccc=" << ccc << endl;
      cout << "Global ccc=" << ::ccc << endl;
}
```

```
main   ccc=123
Global ccc=987
Inner  ccc=456
Global ccc=987
Innest ccc=789
Global ccc=987
Inner  ccc=456
Global ccc=321
main   ccc=123
Global ccc=321
```

Подразбирани вредности за аргументи на функција (1)

```
#include <iostream>
using namespace std;
void show(int = 1, float = 2.3, long = 4);
int main()
{
      show();               // All three arguments default
      show(5);              // Provide 1st argument
      show(6, 7.8);         // Provide 1st and 2nd
      show(9, 10.11, 12L); // Provide all three argument
      show(2.9 , 3.5, 7L); // implicit cast from float to int
      // show(  , 3.5, 7L);  ** Error: cannot omit only first argument
}
void show(int first, float second, long third)
{
      cout << "first = " << first;
      cout << ", second = " << second;
      cout << ", third = " << third;
      cout << endl;
      return;
}
```

```
first = 1, second = 2.3, third = 4
first = 5, second = 2.3, third = 4
first = 6, second = 7.8, third = 4
first = 9, second = 10.11, third = 12
first = 2, second = 3.5, third = 7
```

Подразбирани вредности за аргументи на функција (2)

```
#include <iostream>
using namespace std;

int main()
{
      const char naslov[] = " NASLOV ";
      const char podnaslov[] = "Podnaslov";
      char *tocki[] = { "Tocka eden", "Tocka dva", "Tocka tri" };
      void ruler(int n = 20, char c = '=', bool nl = true);
      ruler();
      cout << naslov << endl;
      ruler(sizeof naslov - 1);

      cout << podnaslov << endl;
      ruler(sizeof podnaslov - 1, '-');

      void list(int n, char **lt);
      list(3, tocki);
      ruler();
}
```

```
====================
 NASLOV
========
Podnaslov
---------
#1 Tocka eden
#2 Tocka dva
#3 Tocka tri
**************
====================
```

Подразбирани вредности за аргументи на функција (3)

```
void list(int n, char **lt)
{
     void ruler(int n = 1, char c = '#', bool nl = false);

     for (int i = 0; i<n; i++)
     {
          ruler();
          cout << i + 1 << ' ' << lt[i] << endl;
     }
     ruler(15, '*', true);
     return;
}

void ruler(int n, char c, bool nl)
{
     for (int i = 0; i<n; i++)
          cout << c;
     if (nl) cout << endl;
}
```

```
====================
 NASLOV
========
Podnaslov
---------
#1 Tocka eden
#2 Tocka dva
#3 Tocka tri
***************
====================
```

Преоптоварување на функции (1)

```
#include <iostream>
using namespace std;
void show(int val)
{
      cout << "Integer: " << val << endl;
}
void show(double val)
{
      cout << "Double: " << val << endl;
}
void show(char *val)
{
      cout << "String: " << val << endl;
}

int main()
{
      show(12);
      show(3.1415);
      show("Hello World!");
      return 0;
}
```

Function overloading is a feature in **C++** where two or more **functions** can have the same name but different parameters.

Не може:

```
int func(char c)
float func(char c)
```

```
Integer: 12
Double: 3.1415
String: Hello World!
```

Преоптоварување на функции (2)

```
#include <iostream>
using namespace std;
#include <string.h>

inline void string_copy(char *dest, const char *src)
{
     strcpy(dest, src);
}

inline void string_copy(char *dest, const char *src, int len)
{
     strncpy(dest, src, len);
}

static char stringa[20], stringb[20];

int main()
{
     string_copy(stringa, "That");
     string_copy(stringb, "This is a string", 4);
     cout << stringb << " and " << stringa;
}
```

```
This and That
```

Референци

```
#include <iostream>
using namespace std;

int main()
{
      int i;
      int x = 123;
      int &y = x;

      cout << "x= " << x << " y= " << y << endl;
      x = 23;
      cout << "x= " << x << " y= " << y << endl;
      x++;
      cout << "x= " << x << " y= " << y << endl;
      y++;
      cout << "x= " << x << " y= " << y << endl;

      return 0;
}
```

Грешка!

```
int &y;
y=x;
```

```
x= 123 y= 123
x= 23 y= 23
x= 24 y= 24
x= 25 y= 25
```

Функции кои враќаат референца

```
#include <iostream>
using namespace std;
int mynum = 100;
int &num()
{
      return mynum;
}
int main()
{
     int i;
     int x = 123;
     int &y = x;
     cout << "x= " << x << " y= " << y << endl;
     x = 23;
     cout << "x= " << x << " y= " << y << endl;
     x++;
     cout << "x= " << x << " y= " << y << endl;
     y++;
     cout << "x= " << x << " y= " << y << endl;
     i = num();
     cout << "i= " << i << " mynum= " << mynum << endl;
     i++;
     cout << "i= " << i << " mynum= " << mynum << endl;
     num() = 5;
     cout << "mynum= " << mynum << " num()= " << num() << endl;
     num()++;
     cout << "mynum= " << mynum << " num()= " << num() << endl;
     return 0;
}
```

```
x= 123 y= 123
x= 23 y= 23
x= 24 y= 24
x= 25 y= 25
i= 100 mynum= 100
i= 101 mynum= 100
mynum= 5 num()= 5
mynum= 6 num()= 6
```

Пренос на аргументите во функциите

```
#include <iostream>
using namespace std;

void swapByVal(int a, int b)
{
      int t = a; a = b; b = t;
}
void swapByPtr(int *a, int *b)
{
      int t = *a; *a = *b; *b = t;
}
void swapByRef(int &a, int &b)
{
      int t = a; a = b; b = t;
}
```

```
int main()
{
      int x = 1, y = 2;
      cout << "              Initial: x="
            << x << " y=" << y << endl;
      swapByVal(x, y);
      cout << "After swapByVal(x,y): x="
            << x << " y=" << y << endl << endl;

      x = 1, y = 2;
      cout << "                Initial: x="
            << x << " y=" << y << endl;
      swapByPtr(&x, &y);
      cout << "After swapByVal(&x,&y): x="
            << x << " y=" << y << endl << endl;

      x = 1, y = 2;
      cout << "              Initial: x="
            << x << " y=" << y << endl;
      swapByRef(x, y);
      cout << "After swapByRef(x,y): x="
            << x << " y=" << y << endl;
      return 0;
}
```

```
              Initial: x=1 y=2
After swapByVal(x,y): x=1 y=2

               Initial: x=1 y=2
After swapByPtr (&x,&y): x=2 y=1

              Initial: x=1 y=2
After swapByRef(x,y): x=2 y=1
```

Референци како аргументи на функција (1)

```
#include <iostream>
using namespace std;

struct bigone
{
      int key;
      char text[1000];
} bo = { 123, "This is a big structure" };

// ---- Three functions that have the structure as a parameter
void valfunc(bigone vl);
void ptrfunc(const bigone *pl);
void reffunc(const bigone &rl);

int main()
{
      valfunc(bo);    // Passing the variable itself
      ptrfunc(&bo);   // Passing the address of the variable
      reffunc(bo);    // Passing a reference to the variable
      return 0;
}
```

Референци како аргументи на функција (2)

```
// --- Pass by value
void valfunc(bigone vl)
{
     cout << endl << vl.key;
     cout << endl << vl.text;
}

// --- Pass by pointer
void ptrfunc(const bigone *pl)
{
     cout << endl << pl->key;       // Pointer notation
     cout << endl << pl->text;
}

// --- Pass by reference
void reffunc(const bigone &rl)
{
     cout << endl << rl.key;        // Reference notation
     cout << endl << rl.text;
}
```

```
123
This is a big structure
123
This is a big structure
123
This is a big structure
```

typedef и структури со функции

- typedef е **дозволено но излишно** во C++

```
struct somestruct
{
      int a;
      double d;
      char string[80];
};

somestruct what;
what.d = 3.1415;

struct point // definition of a screen
{             // dot:
      int x; // coordinates
      int y; // x/y
      void draw(void); // drawing function
};
```

```
point a;  // two points on
point b;  // the screen
a.x = 0;  // define first dot
a.y = 10; // and draw it
a.draw();
b = a;    // copy a to b
b.y = 20; // redefine y-coord
b.draw(); // and draw it
```

in C:

```
typedef struct point {
        float x;
        float y;
} tocka;
```

`tocka a;` or `struct point a;`

Namespaces (1)

`sin()` во `<cmath>` - функција која како аргумент прима агол во радијани и ја враќа вредноста на синусот од аголот.

Еден радијан е 180/PI *степени*.

```
#include <iostream>
#include <cmath>
namespace RAD
{
	double angle;
	double sinus(double x) // arg in rad
	{
		return(std::sin(x));
	}
}

namespace DEG
{
	int angle;
	double sinus(double x) // arg in deg
	{
		return(std::sin(x*M_PI / 180));
	}
}
```

Namespaces (2)

```
using namespace std;
int main()
{
	{
		using namespace RAD;
		angle = 1.05; //RAD::angle=1.05 rad
		cout << " sinus(" << angle << "rad)=" << sinus(angle) << endl;
	}
	{
		using namespace DEG;
		angle = 60; //DEG::angle=60 deg
		cout << " sinus(" << angle << "deg)=" << sinus(angle) << endl;
	}
	return 0;
}
```

```
sinus(1.05rad)=0.867423
sinus(60deg)=0.866025
```

Nested namespaces

```
#include <iostream>

namespace A
{
        int i = 0;
        namespace X
        {
                int i = 1;
                char c = 'x';
        }
        namespace Y
        {
                int i = 2;
                char c = 'y';
        }
}
namespace B
{
        int i = 3;
        int j = 7;
}
namespace C
{
        using namespace B;
        int i = 4;
        int j = 8;
}
namespace D
{
        int j = 8;
}
int i = 5;
using namespace std;

int main()
{
        cout << i << endl;              // global i
        cout << A::X::i << endl;        // A::X::i
        cout << A::Y::i << endl;        // A::Y::i
        cout << B::i << endl;           // B::i
        cout << C::i << endl;           // C::i

        using namespace A;
        //   cout << i << endl; // *** Error: ambiguous symbol global i or A::i
        cout << X::c << endl;           // A::X::c
        cout << Y::c << endl;           // A::Y::c
        {
                using namespace C;

        //      cout << j << endl; // *** Error: ambiguous symbol B::j or C::j
        }
        {
                int j;
                cin >> j;                       // local j
                if (j)
                {
                        using namespace A::X;
                        cout << c << endl;      // A::X::c
                }
                else
                {
                        using namespace A::Y;
                        cout << c << endl;      // A::Y::c
                }
        }
        using namespace D;
        cout << j << endl;              // D::j
        return 0;
}
```

```
5
1
2
3
4
x
y
0  ↵ input
y
8
```

Резервирани зборови и оператори во C++

and	do	*not*	**template**
and_eq	double	*not_eq*	**this**
asm	**dynamic_cast**	**operator**	**throw**
auto	else	*or*	**try**
bitand	enum	*or_eq*	typedef
bitor	**explicit**	**private**	**typeid**
bool	extern	**protected**	**typename**
break	float	**public**	union
case	for	register	unsigned
catch	**friend**	**reinterpret_cast**	**using**
char	goto	return	**virtual**
class	if	short	void
compl	**inline**	signed	volatile
const	int	sizeof	**wchar_t**
const_cast	long	static	while
continue	**mutable**	**static_cast**	*xor*
default	**namespace**	struct	*xor_eq*
delete	**new**	switch	

and	and_eq	bitand	bitor	compl	not	not_eq	or	or_eq	xor	xor_eq
&&	&=	&	|	~	!	!=	||	|=	^	^=

Динамичка алокација на меморија

```c
#include <stdio.h>
#include <malloc.h>
int main()
{
	int d; int *p;
	printf("dim="); scanf("%d", &d);
	p = (int *)malloc(d*sizeof(int));
	if (p == NULL)
	{
		fprintf(stderr, "Nema memorija :(");
		return(-1);
	}
	for (int i = 0; i<d; i++)  p[i] = i*i;

	for (int i = 0; i<d; i++)
		printf("p[%d]=%d\n", i, p[i]);
	free(p);
	return(0);
}
```

```c
#include <iostream>
using namespace std;

int main()
{
	int d;
	cout << "dim=";
	cin >> d;
	int *p = 0;
	p = new int[d];

	for (int i = 0; i<d; i++)
		p[i] = i*i;

	for (int i = 0; i<d; i++)
		cout << "p[" << i << "]="
		<< p[i] << endl;

	delete[] p;

	return(0);
}
```

Покажувачи vs. референци

Вредности, референци и покажувачи за **променливи**

Декларирање	Иницијализација	Декларирање + иницијализација	Пристап до вредност	(Покажувач кон) мемориска локација
`int value;`	`value = 5;`	`int value = 5;`	`value`	`&value`
`int *ptr;`	`ptr = &value;`	`int *ptr = &value;`	`*ptr`	`ptr`
`int &refer= value;`	`int &refer= value;`	`int &refer= value;`	`refer или value`	`&refer или &value`

Вредности, референци и покажувачи како **аргументи на функции**

Тип на функција	Декларирање	Употреба	Можности
Вредност како аргумент	`int funcVArg (int v)`	`int res = funcVArg(value);` **или** `int res = funcVArg(*ptr);` **или** `int res = funcVArg(refer);`	value, *ptr, refer, **може** да се користат, но **не може** да се променат во funcVArg()
Покажувач како аргумент	`int funcPArg (int *p)`	`int res = funcPArg(&value);` **или** `int res = funcPArg(ptr);` **или** `int res = funcPArg(&refer);`	value, *ptr, refer **може** да се користат и **може** да се променат во funcPArg()
Референца како аргумент	`int funcRArg (int &r)`	`int res = funcRArg(value);` **или** `int res = funcRArg(*ptr);` **или** `int res = funcRArg(refer);`	value, *ptr, refer **може** да се користат и **може** да се променат во funcRArg()