ФАКУЛТЕТ ЗА ЕЛЕКТРОТЕХНИКА И ИНФОРМАЦИСКИ ТЕХНОЛОГИИ

Пријателски функции и класи и динамичка алокација на меморија

– Податочни структури и програмирање –

Пријателски функции и класи

- **Со прогласувањето** на дадена функција или класа **за пријателска и се дава** право на **пристап до приватните членови** на класата
- Пријателството **не мора да биде обострано** и **не се наследува**!
- **Не е важно дали** пријателството **ќе се дефинира во `private` или `public`** делот

```
class F1 {
   ...
   friend float soberi(const F1 &, const F2 &);
   friend class F3;
};
class F2 {
   ...
   friend float soberi(const F1 &, const F2 &);
   friend void F1::namali(const F2 &);
};
```

Пример 1

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

class F2;
class F1
{
        int i; float f;
public:
        F1(int a = 0, float b = 0.0) : i(a), f(b) {}
        void show() const { cout << i << ' ' << f << endl; }
        void namali(const F2 &x);
private:
        friend float soberi(const F1 &, const F2 &);
        friend class F3;
};
class F2
{
        int i; float a; char c;
public:
        F2(int a = 0, float b = 0.0, char cc = ' ') : i(a), a(b), c(cc) {}
        void show() const { cout << i << ' ' << a << " '" << c << "'" << endl; }
        friend float soberi(const F1 &, const F2 &);
        friend void F1::namali(const F2 &);
};
```

пријателска глобална функција

пријателска класа

пријателска функција
членка на друга класа

Пример 1 (прод.)

```
class F3
{
        int i; float f;
public:
        F3(int a = 0, float b = 0.0) : i(a), f(b) {}
        void show() const { cout << i << " " << f << endl; }
        int odzemi(const F1 &x) { f -= x.f; return(i -= x.i); }
};
void F1::namali(const F2 &x) {
        i -= x.i;
        f -= x.a;
}
float soberi(const F1 &x, const F2 &y) {
        return(x.f + y.a);
}

int main()
{
        F2 y(1, 2, 'y');
        F1 x(3, 4), z;
        F3 v(5, 4.5);
        x.namali(y); x.show();
        cout << soberi(x, y) << endl;
        v.odzemi(x); v.show();
}
```

може да пристапува до приватните членови на класата F1 бидејќи целата класа F3 е пријател на класата F1

може да пристапува до приватните членови на класата F2

пријателска е за двете класи и може да пристапува до приватните членови и на класата F1 и на класата F2

Пријателска функција

Пријателската функција е дефинирана надвор од класата, но има право да пристапува до сите нејзини членови (приватни и јавни). Иако прототипот на пријателските функции се пишуваат во дефиницијата на класата, тие не се нивни членови – методи

```
#include <iostream>
using namespace std;
class Box {
        private:
                double width;
        public:
                friend void printWidth( Box box );
                void setWidth( double wid );
};

// Дефиниција на методот
void Box::setWidth( double wid ) {
        width = wid;
}

// printWidth() не е член на класата
void printWidth( Box box ) {
      // Бидејќи printWidth() е пријателска на Box, има пристап до членовите на класата
        cout << "Width of box : " << box.width <<endl;
}
```

```
int main() {
        Box box;
        // постави ширина на box
        box.setWidth(10.0);

        // Искористи ја пријателската функција
        printWidth( box );
        return 0;
}
```

Динамичка алокација на меморија: оператори `new[]` и `delete[]` (1)

- Операторот **`new type` динамички алоцира меморија за** сместување на **еден објект од типот `type` и враќа покажувач** кон обезбедената меморија
- **Меморијата се ослободува со операторот `delete` на кој се проследува покажувачот кон меморијата** која треба да се ослободи (истиот кој бил вратен од операторот `new` при алокацијата)

```
int *ip;
ip = new int;
*ip = …
int *jp = new int(2); // go inic. na 2
...
delete ip; delete jp;
```

Динамичка алокација на меморија: оператори new[] и delete[] (2)

Меморијата алоцирана со **new type[n]** треба да се ослободи со **delete[]**!

```
#include <iostream>
using namespace std;
class X {
public:
        X() { cout << "Default constructor called" << endl; }
        X(int a) :i(a) { cout << "Constructor(" << a << ") called" << endl; }
        ~X() { cout << "Destructor for " << i << " called" << endl; }
        void show() const { cout << i << endl; }
private:  int i;
};

int main() {
        X *ip = new X;
        X *jp = new X(2);
        X *niza = new X[4];
        ip->show();
        (*jp).show();
        for (int i = 0; i < 4; i++) {
                cout << ”niza[" << i << "] = ";
                niza[i].show();
        }
        delete ip; delete jp;
        delete[] niza;
}
```

```
Default constructor called
Constructor(2) called
Default constructor called
Default constructor called
Default constructor called
Default constructor called
0
2
niza[0]=3998136
niza[1]=542393678
niza[2]=858796082
niza[3]=1836008284
Destructor for 0 called
Destructor for 2 called
Destructor for 1836008284 called
Destructor for 858796082 called
Destructor for 542393678 called
Destructor for 3998136 called
```

Пример 3

- Да се напише класа која ќе овозможи работа со низи од знаци (стрингови) при што низите ќе се сместуваат во динамички алоцирана меморија (без ефективно ограничување на нивната максимална должина)
- Во секој момент низата треба да зафаќа само онолку меморија колку што навистина е долга!

Пример 3 (прод.)

```
#include <iostream>
#include <cstring>
using namespace std;
class String
{
private:
        char * cptr;
public:
        String(const char * = "");
        ~String();
        void Show(void) const { cout << cptr; }
};
inline String::String(const char * a_cptr)
{
        cptr = new char[strlen(a_cptr) + 1];
        strcpy(cptr, a_cptr);
}
inline String::~String()
{
        delete[] cptr;
}
```

покажувач кон динамички алоцирана меморија во која ќе се чува стрингот

```
int main()
{
        const String b("World");
        String c, d("Hello");
        {
                String a(d);
                a.Show(); cout << ' ';
                b.Show(); cout << endl;
                c = a; d = b;
                c.Show(); cout << ' ';
                d.Show(); cout << endl;
        }
        c.Show(); cout << ' ';
        d.Show(); cout << endl;
        return(0);
}
```

2 проблеми

алоцирање на потребната меморија

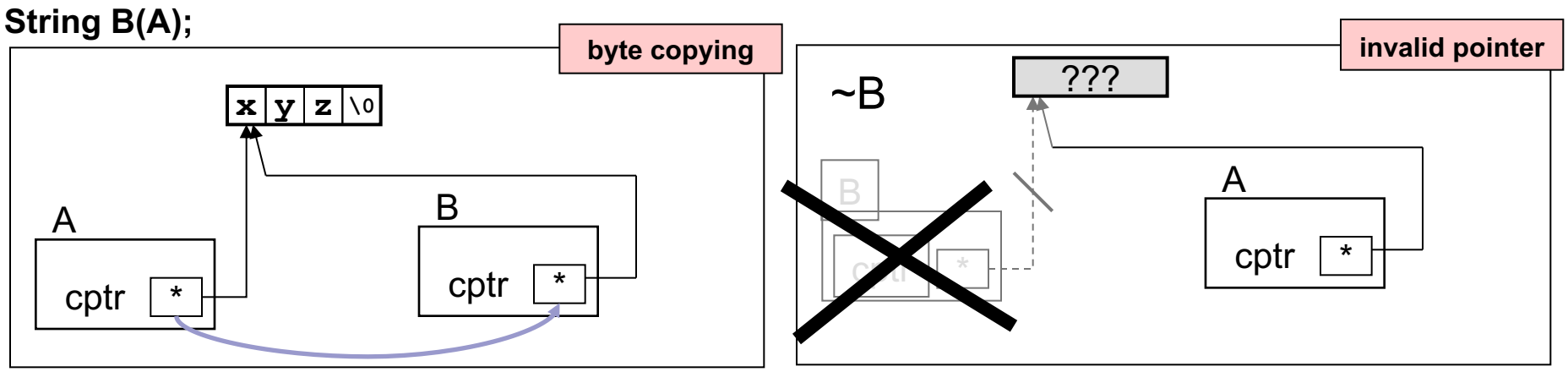
копирање на низата знаци во неа

```
Hello World
Hello World
 World
```

???

Што се случува?

```
String  A("xyz");
{
   String B(A);   //nema copy constructor
} //  Destruct B
```



РЕШЕНИЕ – нов Copy конструктор

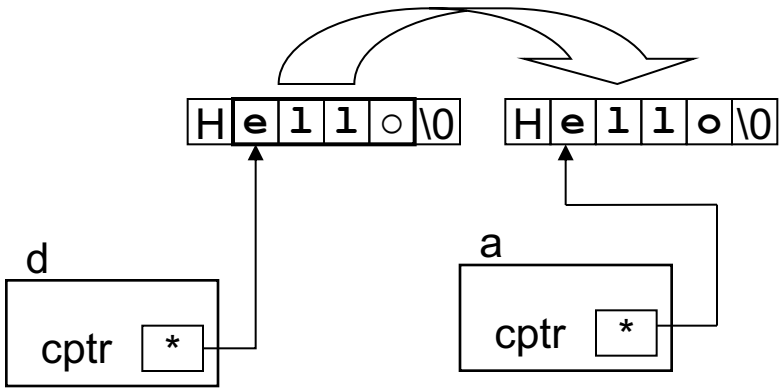
```
class String
{
public:
    . . . //isto so prethodno
    String(const String &); //copy
};

String::String(const String &from)
{
        cptr = new char[strlen(from.cptr) + 1];
        strcpy(cptr, from.cptr);
}
```

```
int main()
{
    const String b("World");
    String c, d("Hello");
    {
        String a(d);
        a.Show(); cout << ' ';
        b.Show(); cout << endl;
        c = a; d = b;
        c.Show(); cout << ' ';
        d.Show(); cout << endl;
    }
    c.Show(); cout << ' ';
    d.Show(); cout << endl;
    return(0);
}
```

нема проблем

1 проблем

се решава со преоптоварување операторски (следна лекција)



String a(d)

```
Hello World
Hello Hello
 ? Hello
```

Пример

```
#include <iostream>
#include <cstring>
using namespace std;
class String {
  private:
        char * cptr;
  public:
    String(const char * = "");
    ~String();
    void Show(void) const {
        cout << cptr; }
    void set (const char * a)
    { strcpy(cptr, a);}
};

String::String(const char * a_cptr){
    cptr = new char[strlen(a_cptr) + 1];
    strcpy(cptr, a_cptr);
}
String::~String()
{ delete[] cptr; }
```

```
int main()
{
    String b(”B");
    String c(b);

    b.Show(); cout << ' ';
    c.Show(); cout << endl;

    b.set("A");
    b.Show(); cout << ' ';
    c.Show(); cout << endl;
    return(0);
}
```

Без ***copy*** конструктор

Излез

```
B B
A A
```

Пример

```
#include <iostream>
#include <cstring>
using namespace std;
class String {
  private:
            char * cptr;
  public:
       String(const char * = "");
       ~String();
       void Show(void) const {
             cout << cptr; }
       void set (const char * a)
       { strcpy(cptr, a);}

       String(const String &); //copy constructor

};

String::String(const char * a_cptr){
       cptr = new char[strlen(a_cptr) + 1];
       strcpy(cptr, a_cptr);
}
String::~String()
{ delete[] cptr; }
```

```
String::String(const String &from)
{
    cptr = new char[strlen(from.cptr) + 1];
    strcpy(cptr, from.cptr);
}
```

```
int main()
{
    String b(”B");
    String c(b);

    b.Show(); cout << ' ';
    c.Show(); cout << endl;

    b.set("A");
    b.Show(); cout << ' ';
    c.Show(); cout << endl;
    return(0);
}
```

Со ***copy*** конструктор

Излез

```
B B
A B
```