STU FEI

Programovacie techniky

5. nullptr, trieda, objekt, konštruktor, deštruktor

nullptr

nullptr = nulový smerník

```
int f(int);
int f(int*);


f(NULL); //ktorá funkcia bude zavolaná?

f(0);        //int f(int);
f(nullptr); //int f(int*)

#define NULL 0 //NULL je 0
```

Od štruktúry k triede

```
struct Token {
    int value;
};


int main() {
   Token obj;
   obj.value = 1;

   return 0;
}
```

```
class Token {
   public:
       int value;
};

int main() {
   Token obj;
   obj.value = 1;

   return 0;
}
```

V C++ je struct trieda, preto môžeme písať Token obj, namiesto struct Token obj

Špecifikátory prístupu

public – k členom triedy je možné pristupovať zvonku

private - členy su viditeľné iba zvnútra triedy

protected (viacej pri dedení tried)

V triede (class) sú členy defaultne private
V štruktúre (struct) sú členy defaultne public

Vznik objektu

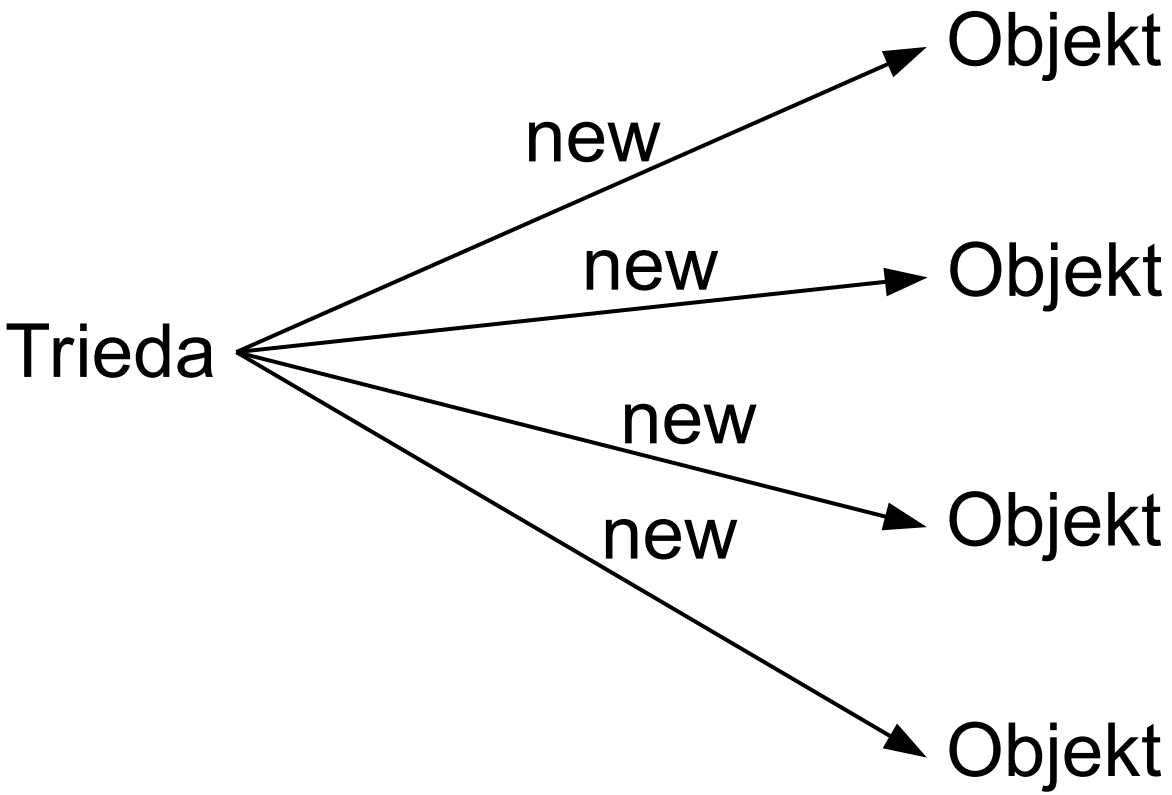
```
class Token {
    public:
        int value;
};

int main() {
    Token obj;
    obj.value = 1;


    return 0;
}
```

Statický vznik objektu

```
class Token {
    public:
        int value;
};

int main() {
    Token* obj = new Token;
    obj->value = 1;

    delete obj;
    return 0;
}
```

Dynamický vznik objektu pomocou new

Trieda, objekt



Objekt je inštancia triedy. Trieda je „návod“ pre vytvorenie objektu. Trieda definuje nový type.

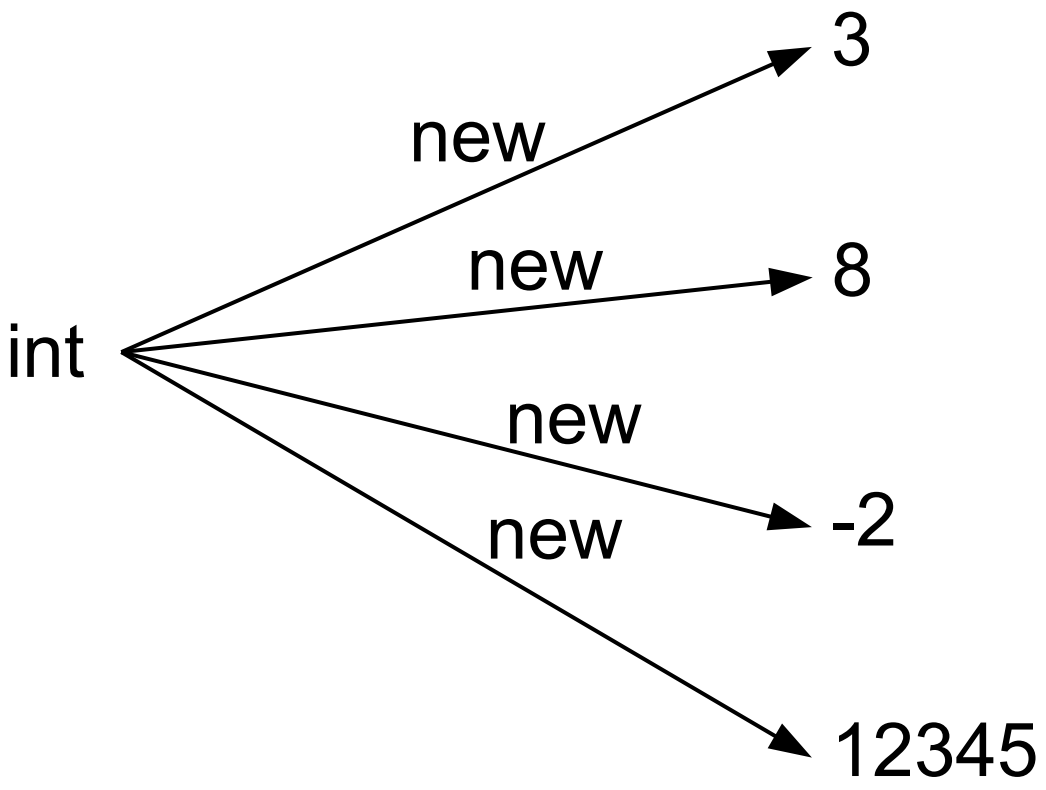
Vznik objektu (základné typy)

```
double *pvalue = new double;
double *pole = new double[5];
char     *str = new char[20];
```

Uvoľnenie pamäte:

```
delete pvalue;
delete[] pole;        //zátvorky [] sú nutné!!!
delete[] str;
```

Trieda, objekt



Objekt existuje v pamäti.

Vznik objektu (základné typy)

Dynamický vznik:

```
double *pvalue = new double; //heap
delete pvalue;
```

Dynamický vznik s vynulovaním:

```
double *pvalue = new double(); //heap
delete pvalue;
```

Statický vznik:

```
double value; //stack
```

Vznik objektu

Dynamický vznik:

```
Token *obj = new Token; //heap
delete obj;
```

Dynamický vznik s vynulovaním (len s generovaným default konštruktorom):

```
Token *obj = new Token(); //heap
delete obj;
```

Statický vznik:

```
Token obj; //stack
```

Dynamická alokácia pamäte

Alokácia dvojrozmerného poľa 2x4

```
int **dvojrozmerne_pole;

dvojrozmerne_pole = new int*[2];

dvojrozmerne_pole[0] = new int[4];

dvojrozmerne_pole[1] = new int[4];
```

Implicitné hodnoty

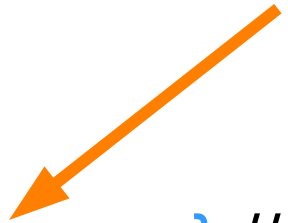
```
#include <iostream>

void print(int n = 1) {
    std::cout << n;
}

int main() {
    print(3); //3
    print();   //1

    return 0;
}
```

public, private: príklad

```
class Rectangle {
  public:
    int w, h;
};

int main () {
  Rectangle r;
  r.w = 10;
  r.h = 12;

  return 0;
}
```

public, private: príklad

```
class Rectangle {
  private:
    int w, h;
  public:
    void setW(int w) {w = w;} //metóda, setter pre w
    void setH(int h)  {h = h;}  //metóda, setter pre h
};

int main () {
  Rectangle r;
  r.setW(3);
  r.setH(25);

  return 0;
}
```



this

```
class Rectangle {
  private:
    int w, h;
  public:
    void setW(int w) {this->w = w;} //setter pre w
    void setH(int h)  {this->h = h;}  //setter pre h
};

int main () {
  Rectangle r;
  r.setW(3);
  r.setH(25);

  return 0;
}
```

this

this = smerník na seba t.j. na objekt, ktorého premenné alebo metódy práve používame.

this rieši prekrytie atribútu triedy lokálnou premennou!!

public, private: príklad

```
class Rectangle {
  private:
    int w = 0, h = 0;
  public:
    int getW() {return w;} //getter
    int getH()  {return h;} //getter
};

int main () {
  Rectangle r;
  int w0 = r.getW(); //w0 = 0

  return 0;
}
```

Príklad

```
class Point {
    public:
        int x,y;
        void  posun(int a, int b) {  //metóda  triedy Point
                x = x+a;
                y = y+b;
            }
};

int main() {
    Point bod;
    bod.x = 1;
    bod.y = 2;
    bod.posun(3,4); //čo sa stane??

    return 0; }
```

Class definuje namespace

```
class Point {  //definuje tiež menný priestor Point
public:
    int x;
    int y;
    void posun(int a, int b);  //deklarácia metódy
};

void Point::posun(int a, int b) { //definícia metódy
     x= x + a;
     y = y + b;
}
```

Definícia metódy posun je mimo triedu

Private vs public

```
class Point {
private:
    int x,  int y;
public:
    void nastav (int a, int b) {
        x = a;   y = b;
    }
};

int main() {
    Point bod;
    bod.x = 1;                  //compile error, x je private
    bod.nastav(3,4);            // OK

     return 0; }
```

Konštruktor

```
class Token {
public:
    int x;

    Token(int x) {
        this->x = x;
    }
}; //bodkočiarka
```

```
int main() {
    Token t(3);

    return 0;
}
```

Konštruktor sa zavolá,
keď sa vytvára objekt

Konštruktor

```
class Token {
public:
    int x;

    Token(int x = 1) {
        this->x = x;
    }
};
```

```
int main() {
   Token t(3); //OK
   Token tt;    //OK

   return 0;
}
```

Skompiluje, pretože existuje konštruktor, ktorý sa dá zavolať bez parametrov (s implicitnou hodnotou 1).

Konštruktor

```
class Token {
public:
    int x;

    Token(int x) {
        this->x = x;
    }
};
```

```
int main() {
   Token t(3); //OK
   Token tt;    //chyba

   return 0;
}
```

Neskompiluje, pretože neexistuje defaultný konštruktor, ktorý sa dá zavolať bez parametrov.

Konštruktor

```
class Token {
public:
    int x;

    Token(int x0): x(x0) {
       //nejaký kód
    }
};
```

```
int main() {
   Token t(3); //OK


   return 0;
}
```

Inicializácia premennej x na hodnotu x0 nastane pred spustením tela konštruktora

Defaultný konštruktor

```
class Token {
public:
    int x;

    Token() {
        x = 1; //lepšie s this
        //this->x = 1;
}

    Token(int x) {
        this->x = x;
    }
};
```

```
int main() {
    Token t(3); //OK
    Token tt;  //OK

    return 0;
}
```

Konštruktor (= default)

```
class Token {
public:
    int x;

    Token() = default;

    Token(int x) {
        this->x = x;
    }
};
```

```
int main() {
    Token t(3); //OK
    Token tt;  //OK

    return 0;
}
```

=default pri vzniku objektu sa vygeneruje aj defaultný konštruktor

Dynamický vznik objektu

```
Token *t, *t1, *t2, *t3;

t = new Token;            //zavolá sa defaultný konštruktor
t1 = new Token(3);        //zavolá sa konštruktor

t2 = new Token[10];       //10x sa zavolá defaultný konštruktor

t3 = new Token(3)[5];     //chyba, neskompiluje
```

Kopírovanie, priraďovanie

Token t0;

Kopírovanie (copy):

Token t = t0; //t ešte neexistuje

Priraďovanie (assignment =):

Token t1;
t1 = t0; //t1 už existuje, nezavolá sa konštruktor (ani kopírovací)

Kopírovací (copy) konštruktor

```
class Token {
private:
    int x = 0;
public:
    Token () = default;

    Token(const Token& t) { //povinná referencia
        this->x = t.x;
    }
};

int main() {
   Token t0;      //zavolá sa defaultný konštruktor
   Token t = t0; //zavolá sa copy konštruktor
   return 0; }
```

Deštruktor

```
class Line {
public:
     double *vector = nullptr;
      int dim = 0;
      Line(int);
      ~Line();
};

Line::Line(int dim = 1) {
    this->dim = dim;
    vector = new double[dim];
}

Line::~Line() {
    delete[] vector;
}
```

```
int main() {

Line *line = new Line(10);

delete line;

return 0;
}
```

Deštruktor

```
Line::~Line() {
    delete[] vector;
}
```

delete[] vector; je potrebné pre uvoľnenie pamäte, ktorú zaberá vector.

C++ za vás nespraví nič, kompilátor nevie aké máte plány s vector-om, či ste jeho adresu medzitým niekde uložili (pre neskoršie spracovanie), ale nie.

Deštruktor

Dynamický vznik objektu:

Deštruktor za zavolá po delete

Statický vznik objektu:

Deštruktor sa zavolá, keď premenná je out-of-scope {}

```
void foo() {

   Token t;

} //zavolá sa deštruktor pre t, premenná je odstránená zo stacku, uvoľní sa pamäť
```

Deštruktor (základné typy)

```
void foo() {

   double a;

} //premenná a zanikne, je odstránená zo stacku,
uvoľní sa pamäť
```

Automatické generovanie

```
class Token {
public:
   int a = 1;
};

int main () {
   Token t0;      //OK, defaultný konštruktor existuje
   Token t = t0; //OK, copy konštruktor existuje
   t = t0;            //OK, viem priraďovať
}
```

Automatické generovanie

```
class Token {
public:
   int a = 1;
};

int main () {
   Token* t0 = new Token;  //OK, defaultný konštruktor
existuje

   delete t0; //OK, deštruktor existuje
}
```

Automatické generovanie

Vždy sa vygeneruje:

- Defaultný konštruktor
- Copy konštruktor
- Operátor priradenia
- Deštruktor

Ak pridáme ľubovolný konštruktor, defaultný konštruktor sa už nevygeneruje.

Ale môžeme ho pridať napr. pomocou =default.

Kopírovací konštruktor

```
class Token {
public:
  int a = 1;
};

int main () {
   Token t0;
   t0.a = 5;

   Token t = t0; //t.a sa rovná 5

   return 0;
}
```

Kopírovací konštruktor

```
class Token {
public:
  int a = 1;
};

void foo(Token t) {  //kopírovanie, zavolá sa
  t.a = 5;           copy konštruktor
}                    //lokálna premenná t zanikne

int main () {
   Token t0;
   foo(t0); //t0.a sa rovná 1

   return 0;
}
```

Kopírovací konštruktor

```
class Token {
public:
  int a = 1;
};

void foo(Token& t) { //bez kopírovania, nezavolá sa
  t.a = 5;                copy konštruktor, t je referencia
}

int main () {
   Token t0;
   foo(t0); //t0.a sa rovná 5

   return 0;
}
```

Terminológia

Naučili sme sa, čo je:

- Trieda
- Objekt
- Premenná triedy
- Metóda triedy
- Getter, setter (špecializované metódy)
- Konštruktor
- Defaultný (prednastavený) konštruktor
- Kopírovací (copy) konštruktor
- Deštruktor
- public, private
- new, delete

Úloha

Pre triedu Line napíš copy konštruktor, ktorý prekopíruje všetky elementy poľa vector (a nie len smerník na toto pole).

Takéto kopírovanie sa nazýva hlboká kópia.

Bez prekopírovania obsahu vektora vznikne „plytká“ kópia.