Anotace

- Generika,
- přetěžování operátorů,
- výjimky,
- automaty (začátek).

Generika I

- v C++ fungují obecnější šablony,

Generika I

- v C++ fungují obecnější šablony,
- využijeme jich, pokud chceme vytvořit šablonu nějaké třídy, tedy

Generika I

- v C++ fungují obecnější šablony,
- využijeme jich, pokud chceme vytvořit šablonu nějaké třídy, tedy
- chceme vytvořit více totožných tříd, které se budou lišit datovým typem.

Generika I

- v C++ fungují obecnější šablony,
- využijeme jich, pokud chceme vytvořit šablonu nějaké třídy, tedy
- chceme vytvořit více totožných tříd, které se budou lišit datovým typem.
- Příklad využití byl minule v podobě generické třídy `List`.

Generika I

- v C++ fungují obecnější šablony,
- využijeme jich, pokud chceme vytvořit šablonu nějaké třídy, tedy
- chceme vytvořit více totožných tříd, které se budou lišit datovým typem.
- Příklad využití byl minule v podobě generické třídy `List`.
- Jde o jistou náhražku maker preprocesoru známých z jazyka C.

Generika II

- Při použití generika jsme postupovali stejně jako u obyčejného datového typu, jen jsme na správné místo dali parametr do špičatých závorek.

Generika II

- Při použití generika jsme postupovali stejně jako u obyčejného datového typu, jen jsme na správné místo dali parametr do špičatých závorek.
- Při definici postupujeme podobně a tímto parametrem bude datový typ se všemi důsledky, které z toho plynou,

Generika II

- Při použití generika jsme postupovali stejně jako u obyčejného datového typu, jen jsme na správné místo dali parametr do špičatých závorek.
- Při definici postupujeme podobně a tímto parametrem bude datový typ se všemi důsledky, které z toho plynou,
- tedy můžeme dělat proměnné dotyčného (parametrického) typu.

Generika II

- Při použití generika jsme postupovali stejně jako u obyčejného datového typu, jen jsme na správné místo dali parametr do špičatých závorek.
- Při definici postupujeme podobně a tímto parametrem bude datový typ se všemi důsledky, které z toho plynou,
- tedy můžeme dělat proměnné dotyčného (parametrického) typu.
- `class jmeno <parametry,oddelene,carkami>`
  `{ definice třídy }`

Generika II

- Při použití generika jsme postupovali stejně jako u obyčejného datového typu, jen jsme na správné místo dali parametr do špičatých závorek.
- Při definici postupujeme podobně a tímto parametrem bude datový typ se všemi důsledky, které z toho plynou,
- tedy můžeme dělat proměnné dotyčného (parametrického) typu.
- `class jmeno <parametry,oddelene,carkami> { definice třídy }`
- Příklad: `public class genericka <T> {public T promenna;}`

Generika příklad

```
public class seznam <T>
{     public T data;
      public seznam<T> next;
}
...
      seznam<int> x=new seznam<int>();
      x.data=10;
      x.next=new seznam<int>();
      // Tohle by bylo spatne:
      // x.next=new seznam<double>();
```

Komplexní čísla

a operace s nimi, aneb přetížené operátory

- Čísla jsou různá: Přirozená, celá, racionální, reálná, komplexní...

Komplexní čísla

a operace s nimi, aneb přetížené operátory

- Čísla jsou různá: Přirozená, celá, racionální, reálná, komplexní...
- Na všech ale má smysl definovat sčítání, násobení, odčítání a dělení.

Komplexní čísla

a operace s nimi, aneb přetížené operátory

- Čísla jsou různá: Přirozená, celá, racionální, reálná, komplexní...
- Na všech ale má smysl definovat sčítání, násobení, odčítání a dělení.
- V počítači máme jen čísla celá a necelá. Co s tím?

Komplexní čísla

a operace s nimi, aneb přetížené operátory

- Čísla jsou různá: Přirozená, celá, racionální, reálná, komplexní...
- Na všech ale má smysl definovat sčítání, násobení, odčítání a dělení.
- V počítači máme jen čísla celá a necelá. Co s tím?
- Definujeme si vlastní třídu. Tu ale nepůjde sčítat a násobit. Anebo - ze by?

Komplexní čísla

a operace s nimi, aneb přetížené operátory

- Čísla jsou různá: Přirozená, celá, racionální, reálná, komplexní...
- Na všech ale má smysl definovat sčítání, násobení, odčítání a dělení.
- V počítači máme jen čísla celá a necelá. Co s tím?
- Definujeme si vlastní třídu. Tu ale nepůjde sčítat a násobit. Anebo - ze by?
- Ano: Dotyčné třídě přetížíme operátory (přesněji dodefinujeme je).

Komplexní čísla

a operace s nimi, aneb přetížené operátory

- Čísla jsou různá: Přirozená, celá, racionální, reálná, komplexní...
- Na všech ale má smysl definovat sčítání, násobení, odčítání a dělení.
- V počítači máme jen čísla celá a necelá. Co s tím?
- Definujeme si vlastní třídu. Tu ale nepůjde sčítat a násobit. Anebo - ze by?
- Ano: Dotyčné třídě přetížíme operátory (přesněji dodefinujeme je).
- Syntakticky se tváříme, jako bychom definovali běžnou statickou metodu, tato funkce se ale bude divně jmenovat.

Příklad

Opět Gaussova celá čísla

```
class kompl
{    public int re,im;
     public kompl(int re,int im)
     {    this.re=re; this.im=im;}
     public static kompl operator + (kompl a,kompl b)
     {    return new kompl(a.re+b.re,a.im+b.im);}
     public static kompl operator * (kompl a,kompl b)
     {    return new kompl(a.re*b.im-a.im*b.im,
               a.re*b.im+a.im*b.re);
     }
}
```

Příklad pokračování

Abychom mohli třídu `kompl` demonstrovat, předefinujeme jí virtuální metodu `ToString` jako minule:

```
public override string ToString()
{    return ""+re+"+ "+im+"i";}
```

A jedeme:

```
kompl a=new kompl(1,0), b=new kompl(0,1),c;
c=a+b;
Console.WriteLine(c);
Console.WriteLine(a*b);
```

Přetížitelné operátory

Přetížit lze mnoho operátorů, konkrétně:
unární `+`, `-`, `!`, `~`, `++`, `--`
a binární `+`, `-`, `*`, `/`, `%`, `&`, `|`, `^`, `<<`, `>>`
NELZE například $\&\&, ||, [], (typ)x, +=, -=...$

Když je problém můžeme...

- ukončit program,

Když je problém můžeme...

- ukončit program,
- na všech možných místech myslet na všechno možné,

Když je problém můžeme...

- ukončit program,
- na všech možných místech myslet na všechno možné,
- nehasit, co nás nepálí.

Když je problém můžeme...

- ukončit program,
- na všech možných místech myslet na všechno možné,
- nehasit, co nás nepálí.
- V Pascalu byly k dispozici první dvě možnosti, tedy buď to pštrosí algoritmus, nebo se starat.

Když je problém můžeme...

- ukončit program,
- na všech možných místech myslet na všechno možné,
- nehasit, co nás nepálí.
- V Pascalu byly k dispozici první dvě možnosti, tedy buďto pštrosí algoritmus, nebo se starat.
- C# umožňuje všechny tři možnosti, my zatím umíme tu první.

Výjimky II

- Výjimky už známe, objevovaly se, když jsme šlápli vedle a program tím skončil.

Výjimky II

- Výjimky už známe, objevovaly se, když jsme šlápli vedle a program tím skončil.
- My ovšem můžeme výjimky odchytávat,

Výjimky II

- Výjimky už známe, objevovaly se, když jsme šlápli vedle a program tím skončil.
- My ovšem můžeme výjimky odchytávat,
- anebo dokonce házet a posílat tím zprávu, že se něco nepovedlo.

Výjimky II

- Výjimky už známe, objevovaly se, když jsme šlápli vedle a program tím skončil.
- My ovšem můžeme výjimky odchytávat,
- anebo dokonce házet a posílat tím zprávu, že se něco nepovedlo.
- Výjimka postupně propadá programem a ukončuje funkce, které ji nečekaly,

Výjimky II

- Výjimky už známe, objevovaly se, když jsme šlápli vedle a program tím skončil.
- My ovšem můžeme výjimky odchytávat,
- anebo dokonce házet a posílat tím zprávu, že se něco nepovedlo.
- Výjimka postupně propadá programem a ukončuje funkce, které ji nečekaly,
- dokud nevypadneme z programu, nebo nenarazíme na blok, který ji očekával.

Výjimky III

- Syntax a sémantika:

Výjimky III

- Syntax a sémantika:
- `try` uvádí blok, ve kterém může nastat výjimka.

Výjimky III

- Syntax a sémantika:
- `try` uvádí blok, ve kterém může nastat výjimka.
- `catch` uvádí ovladač události za `try` blokem.

Výjimky III

- Syntax a sémantika:
- `try` uvádí blok, ve kterém může nastat výjimka.
- `catch` uvádí ovladač události za `try` blokem.
- `catch` bloků může být více, protože výjimek je mnoho typů (a každou můžeme ošetřovat zvlášť, přesto jsou všechny výjimky potomkem třídy `System.Exception`).

Výjimky III

- Syntax a sémantika:
- `try` uvádí blok, ve kterém může nastat výjimka.
- `catch` uvádí ovladač události za `try` blokem.
- `catch` bloků může být více, protože výjimek je mnoho typů (a každou můžeme ošetřovat zvlášť, přesto jsou všechny výjimky potomkem třídy `System.Exception`).
- `finally` uvádí blok, který se má provést v každém případě (ať výjimka přijde nebo ne a ať je výjimka jakákoliv, tedy včetně nečekané).

Výjimky III

- Syntax a sémantika:
- `try` uvádí blok, ve kterém může nastat výjimka.
- `catch` uvádí ovladač události za `try` blokem.
- `catch` bloků může být více, protože výjimek je mnoho typů (a každou můžeme ošetřovat zvlášť, přesto jsou všechny výjimky potomkem třídy `System.Exception`).
- `finally` uvádí blok, který se má provést v každém případě (ať výjimka přijde nebo ne a ať je výjimka jakákoliv, tedy včetně nečekané).
- `throw` hodí výjimku (ovládá se podobně jako return).

Výjimky příklad

```
void bezpecnedeleni(int a, int b)
{    try{
          Console.WriteLine(a / b);
     }
     catch(System.DivideByZeroException e)
     {    Console.WriteLine("NELZE");}
}
```

Vlastní výjimka

```
class me:System.Exception{}
...
void bezpecnedeleni(int a, int b) {    try{
          if(y==0) throw new me();
          return (x/y);
     }
     catch (System.Exception e)
     {    Console.WriteLine("Prisla vyjimka!"); }
     finally
     {    Console.WriteLine("V kazdem pripade...");}
}
```

Výjimky – poznámky I

- Bloků `catch` může být více za sebou.

Výjimky – poznámky I

- Bloků `catch` může být více za sebou.
- Vykoná se první blok, který popisuje dotyčnou výjimku.

Výjimky – poznámky I

- Bloků `catch` může být více za sebou.
- Vykoná se první blok, který popisuje dotyčnou výjimku.
- V C# je třeba definovat ovladače synovské výjimky před rodičovskými:

Výjimky – poznámky I

- Bloků `catch` může být více za sebou.
- Vykoná se první blok, který popisuje dotyčnou výjimku.
- V C# je třeba definovat ovladače synovské výjimky před rodičovskými:
- `catch(System.Exception e){...}`
  `catch(System.DivideByZeroException e){...}`
  ... tohle ani nezkompilujeme.

Výjimky – poznámky II

- Jak neprogramovat:

```
bool uz=false;
while(!uz)
{    try{ volani_divne_funkce();uz=true;}
     catch(System.Exception e)
     {    Console.WriteLine("Tak znova...");}
}
```

Výjimky – poznámky II

- Jak neprogramovat:

```
bool uz=false;
while(!uz)
{    try{ volani_divne_funkce();uz=true;}
     catch(System.Exception e)
     {    Console.WriteLine("Tak znova...");}
}
```

- Výjimky jsou dobrý sluha, ale špatný pán!

Opakování z prváku

- V prváku byly na cvičeních naznačeny tzv. automaty. Využívaly se ke hledání v textu.

Opakování z prváku

- V prváku byly na cvičeních naznačeny tzv. automaty. Využívaly se ke hledání v textu.
- Konkrétně byl některý z těchto: Mealyho, Moorův, (Aho a Corasicková).

Opakování z prváku

- V prváku byly na cvičeních naznačeny tzv. automaty. Využívaly se ke hledání v textu.
- Konkrétně byl některý z těchto: Mealyho, Moorův, (Aho a Corasicková).
- Ve skutečnosti jde o mnohem výkonnější aparát s aplikacemi:

Opakování z prváku

- V prváku byly na cvičeních naznačeny tzv. automaty. Využívaly se ke hledání v textu.
- Konkrétně byl některý z těchto: Mealyho, Moorův, (Aho a Corasicková).
- Ve skutečnosti jde o mnohem výkonnější aparát s aplikacemi:
- od návrhu překladačů (interpretů a parseru konfiguračních souborů)

Opakování z prváku

- V prváku byly na cvičeních naznačeny tzv. automaty. Využívaly se ke hledání v textu.
- Konkrétně byl některý z těchto: Mealyho, Moorův, (Aho a Corasicková).
- Ve skutečnosti jde o mnohem výkonnější aparát s aplikacemi:
- od návrhu překladačů (interpretů a parseru konfiguračních souborů)
- až po teorii složitosti (potažmo těžkosti).

Opakování z prváku

- V prváku byly na cvičeních naznačeny tzv. automaty. Využívaly se ke hledání v textu.
- Konkrétně byl některý z těchto: Mealyho, Moorův, (Aho a Corasicková).
- Ve skutečnosti jde o mnohem výkonnější aparát s aplikacemi:
- od návrhu překladačů (interpretů a parseru konfiguračních souborů)
- až po teorii složitosti (potažmo těžkosti).
- Kleenova věta ukazuje vztah k tzv. gramatikám.

Opakování vyhledávání v textu

- Automat byl vybaven tzv. stavy, mezi kterými přecházel.

Opakování vyhledávání v textu

- Automat byl vybaven tzv. stavy, mezi kterými přecházel.
- Mezi stavy přecházel jen po přečtení vstupu.

Opakování vyhledávání v textu

- Automat byl vybaven tzv. stavy, mezi kterými přecházel.
- Mezi stavy přecházel jen po přečtení vstupu.
- Došel-li automat do správného stavu, oznámil, které slovo našel.

Opakování vyhledávání v textu

- Automat byl vybaven tzv. stavy, mezi kterými přecházel.
- Mezi stavy přecházel jen po přečtení vstupu.
- Došel-li automat do správného stavu, oznámil, které slovo našel.
- My to na pohled zjednodušíme.

- Automat má pouze rozhodnout, zda vstupní slovo patří do zadaného jazyka nebo ne.

- Automat má pouze rozhodnout, zda vstupní slovo patří do zadaného jazyka nebo ne.
- Automat rozhoduje jen o přijetí, nic nevypisuje (tzv. akceptor, ten, který vypisuje, se označuje transducer).

- Automat má pouze rozhodnout, zda vstupní slovo patří do zadaného jazyka nebo ne.
- Automat rozhoduje jen o přijetí, nic nevypisuje (tzv. akceptor, ten, který vypisuje, se označuje transducer).
- Za to začneme měnit parametry automatu. Vždy mu zůstanou stavy, přechodová funkce a pásky (vstupní/výstupní), ze kterých se bude číst pomocí tzv. hlav.

- Automat má pouze rozhodnout, zda vstupní slovo patří do zadaného jazyka nebo ne.
- Automat rozhoduje jen o přijetí, nic nevypisuje (tzv. akceptor, ten, který vypisuje, se označuje transducer).
- Za to začneme měnit parametry automatu. Vždy mu zůstanou stavy, přechodová funkce a pásky (vstupní/výstupní), ze kterých se bude číst pomocí tzv. hlav.

- Automat má pouze rozhodnout, zda vstupní slovo patří do zadaného jazyka nebo ne.
- Automat rozhoduje jen o přijetí, nic nevypisuje (tzv. akceptor, ten, který vypisuje, se označuje transducer).
- Za to začneme měnit parametry automatu. Vždy mu zůstanou stavy, přechodová funkce a pásky (vstupní/výstupní), ze kterých se bude číst pomocí tzv. hlav.

Definition

Automatem nazveme strukturu (S, X, F, z, K), kde

- Automat má pouze rozhodnout, zda vstupní slovo patří do zadaného jazyka nebo ne.
- Automat rozhoduje jen o přijetí, nic nevypisuje (tzv. akceptor, ten, který vypisuje, se označuje transducer).
- Za to začneme měnit parametry automatu. Vždy mu zůstanou stavy, přechodová funkce a pásky (vstupní/výstupní), ze kterých se bude číst pomocí tzv. hlav.

Definition

Automatem nazveme strukturu (S, X, F, z, K), kde

- S označuje množinu stavů,

- Automat má pouze rozhodnout, zda vstupní slovo patří do zadaného jazyka nebo ne.
- Automat rozhoduje jen o přijetí, nic nevypisuje (tzv. akceptor, ten, který vypisuje, se označuje transducer).
- Za to začneme měnit parametry automatu. Vždy mu zůstanou stavy, přechodová funkce a pásky (vstupní/výstupní), ze kterých se bude číst pomocí tzv. hlav.

Definition

Automatem nazveme strukturu (S, X, F, z, K), kde

- S označuje množinu stavů,
- X označuje páskovou abecedu,

- Automat má pouze rozhodnout, zda vstupní slovo patří do zadaného jazyka nebo ne.
- Automat rozhoduje jen o přijetí, nic nevypisuje (tzv. akceptor, ten, který vypisuje, se označuje transducer).
- Za to začneme měnit parametry automatu. Vždy mu zůstanou stavy, přechodová funkce a pásky (vstupní/výstupní), ze kterých se bude číst pomocí tzv. hlav.

Definition

Automatem nazveme strukturu (S, X, F, z, K), kde

- S označuje množinu stavů,
- X označuje páskovou abecedu,
- F je přechodová funkce,

- Automat má pouze rozhodnout, zda vstupní slovo patří do zadaného jazyka nebo ne.
- Automat rozhoduje jen o přijetí, nic nevypisuje (tzv. akceptor, ten, který vypisuje, se označuje transducer).
- Za to začneme měnit parametry automatu. Vždy mu zůstanou stavy, přechodová funkce a pásky (vstupní/výstupní), ze kterých se bude číst pomocí tzv. hlav.

Definition

Automatem nazveme strukturu (S, X, F, z, K), kde

- S označuje množinu stavů,
- X označuje páskovou abecedu,
- F je přechodová funkce,
- z je začáteční stav,

- Automat má pouze rozhodnout, zda vstupní slovo patří do zadaného jazyka nebo ne.
- Automat rozhoduje jen o přijetí, nic nevypisuje (tzv. akceptor, ten, který vypisuje, se označuje transducer).
- Za to začneme měnit parametry automatu. Vždy mu zůstanou stavy, přechodová funkce a pásky (vstupní/výstupní), ze kterých se bude číst pomocí tzv. hlav.

Definition

Automatem nazveme strukturu (S, X, F, z, K), kde

- S označuje množinu stavů,
- X označuje páskovou abecedu,
- F je přechodová funkce,
- z je začáteční stav,
- K je množina přijímacích konečných stavů.

Konečné automaty I

- Konečným automatem nazveme automat vybavený jednou vstupní páskou, na kterou nelze zapisovat s přechodovou funkcí $F : S \times X \to S$, který po přečtení každého znaku posune hlavu na pásce o políčko doprava. Vstup je přijat, pokud je automat v některém z přijímacích stavů v okamžiku, kdy hlava vyjede za konec vstupu.

Konečné automaty I

- Konečným automatem nazveme automat vybavený jednou vstupní páskou, na kterou nelze zapisovat s přechodovou funkcí $F : S \times X \to S$, který po přečtení každého znaku posune hlavu na pásce o políčko doprava. Vstup je přijat, pokud je automat v některém z přijímacích stavů v okamžiku, kdy hlava vyjede za konec vstupu.
- Příklad: Automat zjišťující, zda vstup obsahuje sudý počet jedniček, nebo automat zjišťující, zda vstup je dělitelný 2 nebo 3.

Konečné automaty II

- Konečné automaty se velmi snadno programují: Stavy indexujeme přirozenými čísly, vstupní znaky též, celý automat tak reprezentujeme dvourozměrným polem.
- Jeden rozměr určuje stav automatu s, druhý rozměr i určí vstupní znak a na souřadnici (s, i) najdeme číslo stavu, do kterého máme přejít.

Zásobníkový automat

- Konečný automat je velmi jednoduchý, nedokáže rozpoznat například jazyk $a^n b^n$, tedy jazyk sestávající ze slov obsahující napřed několik znaků a a pak stejný počet znaků b. Mohl by ale rozpoznat jazyk $a* b*$.

Zásobníkový automat

- Konečný automat je velmi jednoduchý, nedokáže rozpoznat například jazyk $a^n b^n$, tedy jazyk sestávající ze slov obsahující napřed několik znaků a a pak stejný počet znaků b. Mohl by ale rozpoznat jazyk $a* b*$.
- Proto budeme automatům přidávat různé pomůcky, například zásobník. Tyto pomůcky mohou obecně pracovat nad stejnou páskovou abecedou jako vstupní páska, nebo také nemusejí.

Zásobníkový automat

- Konečný automat je velmi jednoduchý, nedokáže rozpoznat například jazyk $a^n b^n$, tedy jazyk sestávající ze slov obsahující napřed několik znaků a a pak stejný počet znaků b. Mohl by ale rozpoznat jazyk $a* b*$.
- Proto budeme automatům přidávat různé pomůcky, například zásobník. Tyto pomůcky mohou obecně pracovat nad stejnou páskovou abecedou jako vstupní páska, nebo také nemusejí.
- Pro prázdný znak (pokud nechceme číst znak) přidáme symbol λ.

Zásobníkový automat – definice

- Zásobníkovým automatem nazveme strukturu (S, A, B, F, s, P), kde:
- S je množina stavů, s je počáteční stav, P je množina přijímacích stavů,
- A je pásková abeceda, B je zásobníková abeceda,
- F je přechodová funkce: $S \times (A \cup \lambda) \times B \rightarrow S \times B*$.
- Přičemž zásobníkový automat v každém kroku může přečíst znak na vstupu, v každém případě přečte jeden znak z vrcholu zásobníku, zjistí vlastní stav a podle toho přejde do jiného stavu a vypíše několik znaků na zásobník. Pokud automat přečte znak, posune hlavu o 1 pozici doprava.

Přechodová funkce

nemusí být jednoznačná

- Ačkoliv přechodové funkci říkáme funkce, ne vždy musí být jednoznačná.
- Pokud přechodová funkce není nutně jednoznačná, mluvíme o nedeterministických automatech. Pokud předhodová funkce jednoznačná je, nazýváme automat deterministickým.
- Pro nedeterministický automat je přípustných více výpočtů, proto definujeme, že nedeterministický automat přijme kdykoliv existuje přijímající výpočet.

Nedeterministické a deterministické automaty

- Povšimněte si, že libovůli nedeterministického automatu ponecháváme pouze volbu výpočtu, automat je ale vždy povinný vybrat takový výpočet, který vede do přijímacího stavu (pokud takový výpočet existuje).

Nedeterministické a deterministické automaty

- Povšimněte si, že libovůli nedeterministického automatu ponecháváme pouze volbu výpočtu, automat je ale vždy povinný vybrat takový výpočet, který vede do přijímacího stavu (pokud takový výpočet existuje).
- O konečných automatech se ví, že deterministický KA rozpozná přesně ty samé jazyky, jako nedeterministický.

Příklady na zásobníkový automat

- Rozpoznání jazyka $a^n b^n$ (deterministický),

Příklady na zásobníkový automat

- Rozpoznání jazyka $a^n b^n$ (deterministický),
- palindrom (nedeterministický).

Turingův stroj

... bude poslední popsaný automat, ale ne dnes.