Anotace

- Variant record (záznam s variantami),
- objektové programování.

Struktury

- Struktury slouží k uchovávání spolu souvisejících dat ne nutně stejného typu.

Struktury

- Struktury slouží k uchovávání spolu souvisejících dat ne nutně stejného typu.
- Příklad: Knihovna, telefonní seznam, účetní záznamy...

Struktury

- Struktury slouží k uchovávání spolu souvisejících dat ne nutně stejného typu.
- Příklad: Knihovna, telefonní seznam, účetní záznamy...
- Někdy ale chceme pro různé položky různé údaje.

Struktury

- Struktury slouží k uchovávání spolu souvisejících dat ne nutně stejného typu.
- Příklad: Knihovna, telefonní seznam, účetní záznamy...
- Někdy ale chceme pro různé položky různé údaje.
- Příklad: Časopis nemá autora, kniha nemá redakční radu...

Struktury

- Struktury slouží k uchovávání spolu souvisejících dat ne nutně stejného typu.
- Příklad: Knihovna, telefonní seznam, účetní záznamy...
- Někdy ale chceme pro různé položky různé údaje.
- Příklad: Časopis nemá autora, kniha nemá redakční radu...
- Jak zařídit, abychom si evidovali ta správná data ke správným položkám?

Struktury

- Struktury slouží k uchovávání spolu souvisejících dat ne nutně stejného typu.
- Příklad: Knihovna, telefonní seznam, účetní záznamy...
- Někdy ale chceme pro různé položky různé údaje.
- Příklad: Časopis nemá autora, kniha nemá redakční radu...
- Jak zařídit, abychom si evidovali ta správná data ke správným položkám?
- Variantním recordem, alias záznamem s variantami.

Struktury

- Struktury slouží k uchovávání spolu souvisejících dat ne nutně stejného typu.
- Příklad: Knihovna, telefonní seznam, účetní záznamy...
- Někdy ale chceme pro různé položky různé údaje.
- Příklad: Časopis nemá autora, kniha nemá redakční radu...
- Jak zařídit, abychom si evidovali ta správná data ke správným položkám?
- Variantním recordem, alias záznamem s variantami.
- Řekneme, co má být přítomno pro jednotlivé hodnoty indikátorové proměnné.

Syntax

- Napřed uděláme indikátorový typ (zpravidla výčtový):
  `type typknihy=(kniha,casopis,noviny);`

Syntax

- Napřed uděláme indikátorový typ (zpravidla výčtový):
  `type typknihy=(kniha,casopis,noviny);`
- Následně vyrobíme strukturu, ve které je tento typ obsažený a obalíme ho case-klauzulí:

```
type tkniha=record
    nazev:string;
    pocetstran:integer;
    case typ:typknihy of
        kniha: (autor:string);
        casopis: (sefredaktorc:string;
            barevnost:boolean);
        noviny: (sefredaktorn:string;
            objem_reklamy:real;);
    end;
```

Příklad

Knihovna

```
var knihovna:array[1..100] of tkniha;
begin
      knihovna[1].nazev:='Algoritmy a programovaci
techniky';
      knihovna[1].pocetstran:=300;
      knihovna[1].typ:=kniha;
      knihovna[1].autor:='Pavel Topfer';

      knihovna[2].nazev:='40Hex';
      knihovna[2].pocetstran:=30;{odhaduju...}
      knihovna[2].typ:=casopis;
      knihovna[2].sefredaktorc:='Darkangel';
      .....
```

Poznámky

- Využití je asi jasné.

Poznámky

- Využití je asi jasné.
- Data ve variantní části jsou uložena v tzv. *unii*, tedy jsou ložena přes sebe! V modernějších (neobjektových) jazycích (např. C) tomu říkáme **unie.**

Poznámky

- Využití je asi jasné.
- Data ve variantní části jsou uložena v tzv. *unii*, tedy jsou ložena přes sebe! V modernějších (neobjektových) jazycích (např. C) tomu říkáme **unie.**
- Říkáme si o nich proto, že jsou dávnou (neobjektovou) implementací něčeho na způsob polymorfismu a dědičnosti, o čemž si řekneme později v objektovém programování...

Poznámky

- Využití je asi jasné.
- Data ve variantní části jsou uložena v tzv. *unii*, tedy jsou ložena přes sebe! V modernějších (neobjektových) jazycích (např. C) tomu říkáme **unie.**
- Říkáme si o nich proto, že jsou dávnou (neobjektovou) implementací něčeho na způsob polymorfismu a dědičnosti, o čemž si řekneme později v objektovém programování...
- ... a nyní – to objektové programování...

Třídy a objekty

- Co je objekt?

Třídy a objekty

- Co je objekt?
- Co je třída?

Třídy a objekty

- Co je objekt?
- Co je třída?
- Téměř všechno kolem nás je objekt.

Třídy a objekty

- Co je objekt?
- Co je třída?
- Téměř všechno kolem nás je objekt.
- Třída je vlastně typem jednotlivých objektů.

Třídy a objekty

- Co je objekt?
- Co je třída?
- Téměř všechno kolem nás je objekt.
- Třída je vlastně typem jednotlivých objektů.
- Objekty mají různé vlastnosti (atributy) a schopnosti něco dělat (metody).

Definice tříd

- Jelikož objekty jsou instance jednotlivých tříd, nemá smysl definovat jednotlivé objekty, ale definujeme třídy.

Definice tříd

- Jelikož objekty jsou instance jednotlivých tříd, nemá smysl definovat jednotlivé objekty, ale definujeme třídy.
- Řekneme, že prvek má být typu `object`.

Definice tříd

- Jelikož objekty jsou instance jednotlivých tříd, nemá smysl definovat jednotlivé objekty, ale definujeme třídy.
- Řekneme, že prvek má být typu `object`.
- Následně postupujeme jako při definici struktury (record).

Definice tříd

- Jelikož objekty jsou instance jednotlivých tříd, nemá smysl definovat jednotlivé objekty, ale definujeme třídy.
- Řekneme, že prvek má být typu `object`.
- Následně postupujeme jako při definici struktury (record).
- Jenže máme mnohem více možností.

Příklad

Popíšeme, co má v příslušné třídě být:

```
type autom=object
              nosnost,nalozeno:integer;

              procedure naloz(kolik:integer);
              procedure vyloz(kolik:integer);
              procedure dojed_k_hromade;
              procedure dojed_ke_KS;
              procedure stav;
       end;
```

Implementace metod

Atributy jsou v pořádku, ale co metody? Ty musíme definovat:

```
procedure autom.naloz(kolik:integer);
begin if(kolik+nalozeno>nosnost) then
            nalozeno:=nosnost
      else  nalozeno:=nalozeno+kolik;
      writeln('Nakladam, mam nalozeno ',nalozeno);
end;

procedure autom.vyloz(kolik:integer);
begin if(kolik>nalozeno) then
            nalozeno:=0
      else  nalozeno:=nalozeno-kolik;
      writeln('Vykladam, mam nalozeno ',nalozeno);
end;
```

Pokračování

Další metody

```
procedure autom.dojed_k_hromade;
begin
end;
procedure autom.dojed_ke_KS;
begin
end;
procedure autom.stav;
begin
writeln('Mam nosnost ',nosnost,' a nalozeno
',nalozeno,' tun...');
end;
```

Tvorba instance (tedy objektu)

Konec příkladu

```
var liaz:autom;
begin
      liaz.nosnost:=10;
      liaz.nalozeno:=0;
      liaz.naloz(5);
      liaz.stav;
end.
```

Rané poznámky k objektům

- Vypadají na pohled podobně jako struktury (recordy).

Rané poznámky k objektům

- Vypadají na pohled podobně jako struktury (recordy).
- Umí ovšem přiřadit strukturám "funkce", což může přispět k zamezení nekompetentní práce se "strukturou" .

Rané poznámky k objektům

- Vypadají na pohled podobně jako struktury (recordy).
- Umí ovšem přiřadit strukturám "funkce", což může přispět k zamezení nekompetentní práce se "strukturou" .
- Zatím to ovšem byla jen nevinná dětská hra, objekty toho umějí mnohem víc...

Privátní a veřejné prvky

- Chceme-li řídit přístup k vybraným prvkům, použijeme klíčová slova `public`, `private` a `protected`.

Privátní a veřejné prvky

- Chceme-li řídit přístup k vybraným prvkům, použijeme klíčová slova `public`, `private` a `protected`.
- `public` – modifikátor říká, že prvek je veřejný, smí k němu přistupovat každý (jako v příkladu).

Privátní a veřejné prvky

- Chceme-li řídit přístup k vybraným prvkům, použijeme klíčová slova `public`, `private` a `protected`.
- `public` – modifikátor říká, že prvek je veřejný, smí k němu přistupovat každý (jako v příkladu).
- `private` – označuje sekci neveřejných prvků. K těm se smí jen zevnitř třídy.

Privátní a veřejné prvky

- Chceme-li řídit přístup k vybraným prvkům, použijeme klíčová slova `public`, `private` a `protected`.
- `public` – modifikátor říká, že prvek je veřejný, smí k němu přistupovat každý (jako v příkladu).
- `private` – označuje sekci neveřejných prvků. K těm se smí jen zevnitř třídy.
- `protected` – pochopíme později, nyní stručně: Z třídy lze odvodit "potomka", tedy třídu specifičtější. Do položek `protected` se smí jen ze současné třídy, nebo z potomka.

Privátní a veřejné prvky

- Chceme-li řídit přístup k vybraným prvkům, použijeme klíčová slova `public`, `private` a `protected`.
- `public` – modifikátor říká, že prvek je veřejný, smí k němu přistupovat každý (jako v příkladu).
- `private` – označuje sekci neveřejných prvků. K těm se smí jen zevnitř třídy.
- `protected` – pochopíme později, nyní stručně: Z třídy lze odvodit "potomka", tedy třídu specifičtější. Do položek `protected` se smí jen ze současné třídy, nebo z potomka.
- Toto je způsob, jak zabránit nekompetentnímu přístupu do objektů. Nepovolujte zasahovat tam, kam to není nutné.

Privátní a veřejné prvky

- Chceme-li řídit přístup k vybraným prvkům, použijeme klíčová slova `public`, `private` a `protected`.
- `public` – modifikátor říká, že prvek je veřejný, smí k němu přistupovat každý (jako v příkladu).
- `private` – označuje sekci neveřejných prvků. K těm se smí jen zevnitř třídy.
- `protected` – pochopíme později, nyní stručně: Z třídy lze odvodit "potomka", tedy třídu specifičtější. Do položek `protected` se smí jen ze současné třídy, nebo z potomka.
- Toto je způsob, jak zabránit nekompetentnímu přístupu do objektů. Nepovolujte zasahovat tam, kam to není nutné.
- Typicky se povoluje přístup k metodám a ne k atributům, pro atributy se v krajních případech zavádějí funkce `get` a `set`.

Příklad

Jak jsme říkali, atributy neveřejné, metody veřejné: `type`

```
autom=object
      private nosnost,nalozeno:integer;
      public procedure naloz(kolik:integer);
            procedure vyloz(kolik:integer);
            procedure stav;
      end;
```

... jenže ouha! Kód:

```
var liaz:autom;
begin liaz.nosnost:=10;
      liaz.nalozeno:=0;
      liaz.naloz(5);
      liaz.stav;
end.
```

... nebude fungovat!

Opravička problému

Zachráníme to funkcí `init`:

```
type autom=object
      private nosnost,nalozeno:integer;
      public procedure naloz(kolik:integer);
             procedure vyloz(kolik:integer);
             procedure stav;
             procedure init(nosn:integer);
      end;
procedure autom.init(nosn:integer);
begin nosnost:=nosn; nalozeno:=0;
end;
var   liaz:autom;
begin liaz.init(10);
      ...
end.
```

Přirozené zobecnění

Bez dynamické alokace to nebylo ono ani bez objektů...

- V programu nám typicky nepostačí předem daný (pevný) počet objektů.

Přirozené zobecnění

Bez dynamické alokace to nebylo ono ani bez objektů...

- V programu nám typicky nepostačí předem daný (pevný) počet objektů.
- Zpravidla na ně chceme dělat pointery (jako když jsme se učili o spojových seznamech).

Přirozené zobecnění

Bez dynamické alokace to nebylo ono ani bez objektů...

- V programu nám typicky nepostačí předem daný (pevný) počet objektů.
- Zpravidla na ně chceme dělat pointery (jako když jsme se učili o spojových seznamech).
- Stačí, aby auto dojelo do fronty u hromady s pískem...

Přirozené zobecnění

Bez dynamické alokace to nebylo ono ani bez objektů...

- V programu nám typicky nepostačí předem daný (pevný) počet objektů.
- Zpravidla na ně chceme dělat pointery (jako když jsme se učili o spojových seznamech).
- Stačí, aby auto dojelo do fronty u hromady s pískem...
- Postupujeme úplně stejně, jako když jsme se učili o pointerech (potažmo spojových seznamech):

Přirozené zobecnění

Bez dynamické alokace to nebylo ono ani bez objektů...

- V programu nám typicky nepostačí předem daný (pevný) počet objektů.
- Zpravidla na ně chceme dělat pointery (jako když jsme se učili o spojových seznamech).
- Stačí, aby auto dojelo do fronty u hromady s pískem...
- Postupujeme úplně stejně, jako když jsme se učili o pointerech (potažmo spojových seznamech):
-
```
type automobil=^autom;
     autom=object;
     ...
     end;
var liaz:automobil;
...
```

Dynamicky alokované objekty

Pracuje se s nimi úplně přirozeně:

```
var liaz:automobil;
begin
      liaz:=new(automobil);
      liaz^.init;
      liaz^.naloz(5);
      ...
      dispose(liaz);
end.
```

Jenže typicky při naalokování (nebo odalokování) chceme udělat plno věcí (zinicializovat nebo uklidit). K tomu byl navržen tzv. *konstruktor*, resp. *destruktor.*

Konstruktory a destruktory

- Definujeme je podobně jako metody, ale uvedeme je klíčovým slovem `constructor` resp. `destructor`.

Konstruktory a destruktory

- Definujeme je podobně jako metody, ale uvedeme je klíčovým slovem `constructor` resp. `destructor`.
- K jejich volání pak dochází při volání `new` resp. `dispose`.

Konstruktory a destruktory

- Definujeme je podobně jako metody, ale uvedeme je klíčovým slovem `constructor` resp. `destructor`.
- K jejich volání pak dochází při volání `new` resp. `dispose`.
- Těmto funkcím jako druhý parametr předáme explicitní volání konstruktoru (resp. destruktoru).

Konstruktory a destruktory

- Definujeme je podobně jako metody, ale uvedeme je klíčovým slovem `constructor` resp. `destructor`.
- K jejich volání pak dochází při volání `new` resp. `dispose`.
- Těmto funkcím jako druhý parametr předáme explicitní volání konstruktoru (resp. destruktoru).
- Konstruktorů i destruktorů může být kolik chce a mohou se jmenovat vpodstatě jak chtějí.

Příklad konstruktory a destruktory

Stále auta

```
type automobil=^autom;
      autom=object;
      ...
      constructor init;
      constructor init(nosn:integer);
      destructor done;
      end;
constructor init;
begin nosnost:=10;
      nalozeno:=0;
      writeln('Zavolan konstruktor bez parametru!');
end;
```

Příklad – pokračování

Konstruktory a destruktory

```
constructor autom.init(nosn:integer);
begin
      nosnost:=nosn;
      nalozeno:=0;
      writeln('Vytvoreno auto s nosnosti ',nosnost);
end;
destructor autom.done;
begin
      writeln('Auto jede do srotu!');
end;
```

Konstruktory a destruktory

Použití – tedy volání

```
var liaz,tatra:automobil;
begin
      liaz:=new(automobil,init);
      liaz^.naloz(5);
      liaz^.stav;
      dispose(liaz,done);
      tatra:=new(automobil,init(15));
      dispose(tatra,done);
      {Tatra nezná bratra!}
end.
```

- Objektů by zjevně bylo možno použít k implementaci spojového seznamu.
- Funkce tento seznam obhospodařující by bylo možno udržovat jako metody.
- S konstruktory odpadá nutnost vyplňovat údaje ve struktuře vždycky jeden po druhé.
- Abychom mohli udělat spojový seznam samostatně implementovaný (bez globálních funkcí), můžeme ho udělat s hlavou (a volat metody nějakého reprezentanta tohoto spojového seznamu).
- Jak to udělat? Dnes cvičení, ukážeme si příště (nebo mezi nedodělky).

Dědičnost

Změna příkladu!

- Vraťme se k příkladu s knihovnou. Máme tiskoviny různých typů.

Dědičnost

Změna příkladu!

- Vraťme se k příkladu s knihovnou. Máme tiskoviny různých typů.
- Společné mají jen to, že se dávají do knihovny.

Dědičnost

Změna příkladu!

- Vraťme se k příkladu s knihovnou. Máme tiskoviny různých typů.
- Společné mají jen to, že se dávají do knihovny.
- Může se jednat o knihu, časopis nebo noviny.

Dědičnost

Změna příkladu!

- Vraťme se k příkladu s knihovnou. Máme tiskoviny různých typů.
- Společné mají jen to, že se dávají do knihovny.
- Může se jednat o knihu, časopis nebo noviny.
- Jednotlivé typy definujeme jako potomka typu `tiskovina`.

Dědičnost

Změna příkladu!

- Vraťme se k příkladu s knihovnou. Máme tiskoviny různých typů.
- Společné mají jen to, že se dávají do knihovny.
- Může se jednat o knihu, časopis nebo noviny.
- Jednotlivé typy definujeme jako potomka typu `tiskovina`.
- Syntakticky: Za klíčové slovo `object` do závorky uvedeme rodiče.

Dědičnost

Změna příkladu!

- Vraťme se k příkladu s knihovnou. Máme tiskoviny různých typů.
- Společné mají jen to, že se dávají do knihovny.
- Může se jednat o knihu, časopis nebo noviny.
- Jednotlivé typy definujeme jako potomka typu `tiskovina`.
- Syntakticky: Za klíčové slovo `object` do závorky uvedeme rodiče.
- Semanticky: Dojde ke zdědění všeho, čím rodič disponoval.

Příklad

```
type ptisk=^tiskovina;
tiskovina=object
      nazev:string;
      pocet_stran:integer;
      procedure zasun_do_knihovny;
      procedure vytahni_z_knihovny;
      ptisk next;
end;
kniha=object(tiskovina)
      autor:string;
end;
casopis=object(tiskovina)
      sefredaktor:string;
      barevnost:boolean;
end;
```

Příklad – pokračování

...

```
noviny=object(tiskovina)
       sefredaktor:string;
       objem_reklamy:real;
end;
```

- Třídy `kniha`, `casopis` i noviny zdědí společné údaje,

Příklad – pokračování

...

```
noviny=object(tiskovina)
       sefredaktor:string;
       objem_reklamy:real;
end;
```

- Třídy `kniha`, `casopis` i noviny zdědí společné údaje,
- stejně jako metody `pridej_do_knihovny` a `vytahni_z_knihovny`

Příklad – pokračování

...

```
noviny=object(tiskovina)
       sefredaktor:string;
       objem_reklamy:real;
end;
```

- Třídy `kniha`, `casopis` i noviny zdědí společné údaje,
- stejně jako metody `pridej_do_knihovny` a `vytahni_z_knihovny`
- a dokonce i ukazatel na `next`.

Příklad – pokračování

...

```
noviny=object(tiskovina)
       sefredaktor:string;
       objem_reklamy:real;
end;
```

- Třídy `kniha`, `casopis` i noviny zdědí společné údaje,
- stejně jako metody `pridej_do_knihovny` a `vytahni_z_knihovny`
- a dokonce i ukazatel na `next`.
- Příklad nasvědčuje, že pro objekty neplatí až tak striktní typová kontrola, jak známe a tedy že je možné na synovský objekt si ukázat jako na rodiče.

Příklad – pokračování

...

```
noviny=object(tiskovina)
       sefredaktor:string;
       objem_reklamy:real;
end;
```

- Třídy `kniha`, `casopis` i noviny zdědí společné údaje,
- stejně jako metody `pridej_do_knihovny` a `vytahni_z_knihovny`
- a dokonce i ukazatel na `next`.
- V konjunkci s tím, že je možné při dědění metody předefinovávat začíná pravá objektová legrace. Začne být zajímavé zjišťovat, které metody se kdy zavolají a jak se dobýt k těm správným.

Příklad – pokračování

...

```
noviny=object(tiskovina)
       sefredaktor:string;
       objem_reklamy:real;
end;
```

- Třídy `kniha`, `casopis` i noviny zdědí společné údaje,
- stejně jako metody `pridej_do_knihovny` a `vytahni_z_knihovny`
- a dokonce i ukazatel na `next`.
- K tomu použijeme tzv. *virtuální metody*, o těch si ale povíme až příště.

Konec

...děkuji za pozornost...

Otázky?