### Dynamické datové typy a struktury

#### Programovací techniky

doc. Ing. Jiří Rybička, Dr. ústav informatiky PEF MENDELU v Brně rybicka@mendelu.cz



Uchovávají adresu v paměti

- Uchovávají adresu v paměti
- Určitý ukazatel adresa proměnné známého typu

- Uchovávají adresu v paměti
- Určitý ukazatel adresa proměnné známého typu
- Obecný ukazatel adresa libovolné proměnné

- Uchovávají adresu v paměti
- Určitý ukazatel adresa proměnné známého typu
- Obecný ukazatel adresa libovolné proměnné
- Podprogram adresa podprogramu

• Definice typu:

```
type neco = ^baze;
```

Definice typu:

```
type neco = ^baze;
```

• Operace:

```
var P: neco;
    {deklarace ukazatele}
    new(P);
    {alokace paměti, vytvoření proměnné}
    P^:=...
    {přístup k dynamické proměnné}
    dispose(P)
    {uvolnění paměti, zrušení proměnné}
```



• Alternativní alokace a uvolnění paměti:

```
GetMem(X, N);
FreeMem(X, N)
```

• Alternativní alokace a uvolnění paměti:

```
GetMem(X, N);
FreeMem(X, N)
```

• N je požadovaný počet bytů

• Alternativní alokace a uvolnění paměti:

```
GetMem(X, N);
FreeMem(X, N)
```

- N je požadovaný počet bytů
- Zde se nekontroluje velikost paměti a potřebná velikost pro danou proměnnou



• Deklarace (viz předchozí přednáška): X: pointer;

- Deklarace (viz předchozí přednáška): X: pointer;
- Alokace a dealokace paměti je umožněna výhradně pomocí GetMem a FreeMem

- Deklarace (viz předchozí přednáška): X: pointer;
- Alokace a dealokace paměti je umožněna výhradně pomocí GetMem a FreeMem
- Přístup k proměnné je možný jen po přetypování překladač potřebuje informaci o tom, jak má s danou proměnnou pracovat

- Deklarace (viz předchozí přednáška): X: pointer;
- Alokace a dealokace paměti je umožněna výhradně pomocí GetMem a FreeMem
- Přístup k proměnné je možný jen po přetypování překladač potřebuje informaci o tom, jak má s danou proměnnou pracovat
- Masové využití pro struktury, které mohou uchovávat data různých typů.



• Ukazatel na proceduru nebo funkci.

- Ukazatel na proceduru nebo funkci.
- Definice typu představuje "šablonu" hlavičky podprogramu, např.

```
type realnafunkce = function (X: real): real;
```

- Ukazatel na proceduru nebo funkci.
- Definice typu představuje "šablonu" hlavičky podprogramu, např.

```
type realnafunkce = function (X: real): real;
```

Deklarace proměnné:

```
var F: realnafunkce;
```

- Ukazatel na proceduru nebo funkci.
- Definice typu představuje "šablonu" hlavičky podprogramu, např.

```
type realnafunkce = function (X: real): real;
```

• Deklarace proměnné:

```
var F: realnafunkce;
```

Přiřazení hodnoty do proměnné typu podprogram:

```
F:=@MojeFunkce;
```

- Ukazatel na proceduru nebo funkci.
- Definice typu představuje "šablonu" hlavičky podprogramu, např.

```
type realnafunkce = function (X: real): real;
```

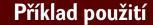
• Deklarace proměnné:

```
var F: realnafunkce;
```

• Přiřazení hodnoty do proměnné typu podprogram:

```
F:=@MojeFunkce;
```

 Operátor @ vydá adresu podprogramu, která je dosazena do ukazatele. Dřívější verze (TP) tento operátor nepotřebovaly.



### Příklad použití

 Příklad: Výpočet určitého integrálu reálné funkce jedné proměnné

### Příklad použití

- Příklad: Výpočet určitého integrálu reálné funkce jedné proměnné
- Knihovní funkce musíme "zabalit" do vlastních funkcí, na které se pak můžeme odkazovat operátorem @

#### Příklad použití

- Příklad: Výpočet určitého integrálu reálné funkce jedné proměnné
- Knihovní funkce musíme "zabalit" do vlastních funkcí, na které se pak můžeme odkazovat operátorem @
- Vytvoříme funkce vyhovující uvedené šabloně:

### Příklad použití – pokračování

```
function Integral (A, B: real;
                    F: realnafunkce):real;
 var vysl:real; krok,x:real; I:word;
begin krok:=(B-A)/100;
       vysl := (f(A) + f(B)) / 2;
       x := A + krok;
        for I:=1 to 100 do begin
          vysl:=vysl+f(x);
          x := x + k r \circ k
       end:
       Integral:=vvsl*krok;
 end:
```

### Příklad použití – dokončení



### Implementace objektů

 Datový typ podprogram je základní součástí implementace objektů

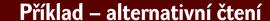
### Implementace objektů

- Datový typ podprogram je základní součástí implementace objektů
- Záznam obsahující datové složky může být doplněn o funkční složky formou ukazatelů na podprogramy

### Implementace objektů

- Datový typ podprogram je základní součástí implementace objektů
- Záznam obsahující datové složky může být doplněn o funkční složky formou ukazatelů na podprogramy
- Implementační model objektu:

```
type neco = record
    A: integer;
    B: string[45];
    C: realnafunkce
end;
```



#### Příklad – alternativní čtení

 Zadání příkladu: Vstup – řada řetězců; výstup – vstupní řada řetězců v opačném pořadí.

```
const MaxPole=1000;
      MaxString=100;
type Indexy = 1..MaxPole;
     Pristupy = 0..MaxPole;
     Retez = string[MaxString];
     Pole = array [Indexy] of Retez;
var P: Pole:
    I, Pocet: Pristupy;
    k: bvte;
```

#### Příklad – alternativní čtení

```
function CtiJedenRetezec: Retez;
var x: byte; pom: Retez;
begin Pom:='';
       for x:=1 to K do begin
          read(Pom[x]);
          if Pom[x] < ' ' then Pom[x] := ' '
       end:
       Pom[0]:=char(K);
       CtiJedenRetezec:=Pom:
end:
```

#### Příklad – alternativní čtení

```
function CtiDruhyRetezec: Retez;
var x: byte; pom: Retez; znak: char;
begin Pom:='';
       read(znak);
       while not ((znak in [#0..' ']) or eof)
          do begin
           Pom:=Pom+Znak;
           read(Znak)
       end:
       CtiDruhyRetezec:=Pom
end:
```

#### Příklad – alternativní čtení

```
function CtiTretiRetezec: Retez:
var pom: Retez;
begin readln(pom);
       CtiTretiRetezec:=Pom
end:
type TypCteni = function: Retez;
var Cteni: TypCteni;
begin K:=6; Cteni:=@CtiPrvniRetezec;
      Pocet:=0:
      while not eof do begin
         Inc (Pocet);
         P[Pocet]:=Cteni;
      end:
      for I:=Pocet downto 1 do writeln(P[I])
end.
```



 Seznamy – jednosměrný, obousměrný, kruhový, aktivní, zásobník, fronta.

- Seznamy jednosměrný, obousměrný, kruhový, aktivní, zásobník, fronta.
- Strom uzly a následníci, binární, ternární, n-ární,

- Seznamy jednosměrný, obousměrný, kruhový, aktivní, zásobník, fronta.
- Strom uzly a následníci, binární, ternární, n-ární,
- Graf orientovaný, neorientovaný

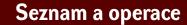
- Seznamy jednosměrný, obousměrný, kruhový, aktivní, zásobník, fronta.
- Strom uzly a následníci, binární, ternární, n-ární,
- Graf orientovaný, neorientovaný
- Konstrukce struktur prvky = kontejnery s daty a strukturními složkami (záznam)

```
type Ukazatelnaprvek = ^prvek;
    prvek = record
    Data: Typdata;
    {datová složka}
    ukaz: Ukazatelnaprvek;
    {strukturní složka}
end;
```

- Seznamy jednosměrný, obousměrný, kruhový, aktivní, zásobník, fronta.
- Strom uzly a následníci, binární, ternární, n-ární,
- Graf orientovaný, neorientovaný
- Konstrukce struktur prvky = kontejnery s daty a strukturními složkami (záznam)

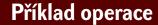
```
type Ukazatelnaprvek = ^prvek;
    prvek = record
    Data: Typdata;
    {datová složka}
    ukaz: Ukazatelnaprvek;
    {strukturní složka}
end;
```

• Kontejnery jsou svázány ukazateli.



#### Seznam a operace

- · Obyčejný seznam:
  - 1 Vytvoření prázdného seznamu.
  - Vložení prvku do seznamu na začátek, konec, před/za aktivní prvek, obecně
  - 3 Odstranění prvku začátek, konec, aktivní, jinak
  - 4 Změna pořadí prvků strukturně, datově
  - Průchody získání jiné datové struktury, počítání prvků, hledání



### Příklad operace

Manipulace s ukazateli při vkládání do dvousměrného seznamu

```
if aktiv<>nil then begin
  if aktiv^.Predch<>nil then begin
      aktiv^.Predch^.Nasled:=aktiv^.Nasled;
  end else Zac:=aktiv^.Nasled;
  if aktiv^.Nasled<>nil then begin
      aktiv^.Nasled^.Predch:=aktiv^.Predch;
  end else Kon:=aktiv^.Predch;
end;
```



## Příklad operace

#### Vyhledání a zrušení prvku

```
Pom:=Zac:
if Pom<>nil then begin {neprázdný seznam}
   if Pom^.D=hledaný then begin {rušený je první}
      Zac:=Zac^.Nasled; Dispose(Pom)
   end else begin {rušený je jinde}
      if Pom^.Nasled<>nil then begin
         while (Pom^.Nasled<>nil) and
                (Pom^.Nasled^.D<>hledaný)
               do Pom:=Pom^.Nasled;
         if Pom^.Nasled<>nil then begin
            Ruseny:=Pom^.Nasled;
            Pom^.Nasled:=Rusenv^.Nasled;
            dispose (Ruseny)
         end
       end
   end
end;
```