

Implementace ADT. Programové moduly

Programovací techniky

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

- Samostatně kompilovatelná část programového díla

- Samostatně kompilovatelná část programového díla
- Uložen v samostatném souboru

- Samostatně kompilovatelná část programového díla
- Uložen v samostatném souboru
- Umožňuje dělbu práce na programu

- Samostatně kompilovatelná část programového díla
- Uložen v samostatném souboru
- Umožňuje dělbu práce na programu
- Podporuje strukturovaný přístup

- Samostatně kompilovatelná část programového díla
- Uložen v samostatném souboru
- Umožňuje dělbu práce na programu
- Podporuje strukturovaný přístup
- Vhodná implementace abstraktního typu

- **Hlavička:** `unit id;`

- **Hlavička:** `unit id;`
- Identifikátor modulu musí být shodný se jménem souboru, v němž je modul zapsán

- **Hlavička:** `unit id;`
- Identifikátor modulu musí být shodný se jménem souboru, v němž je modul zapsán
- **Rozhraní (veřejná část):** uvozeno klíčovým slovem `interface`

- **Hlavička:** `unit id;`
- Identifikátor modulu musí být shodný se jménem souboru, v němž je modul zapsán
- **Rozhraní (veřejná část):** uvozeno klíčovým slovem `interface`
- V rozhraní jsou běžné definice a deklarace, podprogramy jsou zde uvedeny jen svými hlavičkami

- **Hlavička:** `unit id;`
- Identifikátor modulu musí být shodný se jménem souboru, v němž je modul zapsán
- **Rozhraní (veřejná část):** uvozeno klíčovým slovem `interface`
- V rozhraní jsou běžné definice a deklarace, podprogramy jsou zde uvedeny jen svými hlavičkami
- Všechny elementy uvedené v sekci rozhraní jsou viditelné z vnějšku modulu

- **Implementace (soukromá část):** uvozena klíčovým slovem `implementation`

- **Implementace (soukromá část):** uvozena klíčovým slovem `implementation`
- Obsahuje rovněž běžné definice a deklarace, jsou zde uvedena těla podprogramů, jejichž hlavičky jsou v rozhraní

- **Implementace (soukromá část):** uvozena klíčovým slovem `implementation`
- Obsahuje rovněž běžné definice a deklarace, jsou zde uvedena těla podprogramů, jejichž hlavičky jsou v rozhraní
- Všechny elementy uvedené v sekci implementace nejsou z vnějšku modulu dostupné

- **Implementace (soukromá část):** uvozena klíčovým slovem `implementation`
- Obsahuje rovněž běžné definice a deklarace, jsou zde uvedena těla podprogramů, jejichž hlavičky jsou v rozhraní
- Všechny elementy uvedené v sekci implementace nejsou z vnějšku modulu dostupné
- **Inicializační část:** posloupnost příkazů ohraničená slovy `begin` a `end`.

- **Implementace (soukromá část):** uvozena klíčovým slovem `implementation`
- Obsahuje rovněž běžné definice a deklarace, jsou zde uvedena těla podprogramů, jejichž hlavičky jsou v rozhraní
- Všechny elementy uvedené v sekci implementace nejsou z vnějšku modulu dostupné
- **Inicializační část:** posloupnost příkazů ohraničená slovy `begin` a `end`.
- Nejsou-li potřeba inicializační příkazy, může chybět klíčové slovo `begin`

- V jiném programu nebo modulu se připojí příkazem `uses`

- V jiném programu nebo modulu se připojí příkazem `uses`
- Hierarchie modulů programového celku

- V jiném programu nebo modulu se připojí příkazem `uses`
- Hierarchie modulů programového celku
- Křížové reference – modul A potřebuje modul B a ten potřebuje modul A

- V jiném programu nebo modulu se připojí příkazem `uses`
- Hierarchie modulů programového celku
- Křížové reference – modul A potřebuje modul B a ten potřebuje modul A
- Signál chyby v koncepci a návrhu díla

- V jiném programu nebo modulu se připojí příkazem `uses`
- Hierarchie modulů programového celku
- Křížové reference – modul A potřebuje modul B a ten potřebuje modul A
- Signál chyby v koncepci a návrhu díla
- Lze vyřešit voláním pomocí příkazu `uses` umístěném v modulu A v sekci rozhraní a v modulu B v sekci implementace

Implementace abstraktního typu pomocí modulu

Implementace abstraktního typu pomocí modulu

- Odvození datového typu hodnot v rozhraní

Implementace abstraktního typu pomocí modulu

- Odvození datového typu hodnot v rozhraní
- Hlavičky operací v rozhraní

Implementace abstraktního typu pomocí modulu

- Odvození datového typu hodnot v rozhraní
- Hlavičky operací v rozhraní
- Těla operací v sekci implementace

Implementace abstraktního typu pomocí modulu

- Odvození datového typu hodnot v rozhraní
- Hlavičky operací v rozhraní
- Těla operací v sekci implementace
- Všechny prvky zmíněné v diagramu signatury se objeví v rozhraní modulu

Implementace abstraktního typu pomocí modulu

- Odvození datového typu hodnot v rozhraní
- Hlavičky operací v rozhraní
- Těla operací v sekci implementace
- Všechny prvky zmíněné v diagramu signatury se objeví v rozhraní modulu
- Příklad: ADT Zásobník (diagram signatury v minulé přednášce)

Implementace abstraktního typu pomocí modulu

- Odvození datového typu hodnot v rozhraní
- Hlavičky operací v rozhraní
- Těla operací v sekci implementace
- Všechny prvky zmíněné v diagramu signatury se objeví v rozhraní modulu
- Příklad: ADT Zásobník (diagram signatury v minulé přednášce)
- Použití: Obrácení posloupnosti řetězců na standardním vstupu


```
unit Zasobnik;  
interface  
type  
    TypData = string;  
    PClen = ^Clen;  
    Clen = record  
        data: TypData;  
        dalsi: PClen  
    end;  
    Stack = PClen;  
  
procedure Init(var S: Stack);  
function Empty(S: Stack): boolean;  
procedure Add(var S: Stack; D: TypData);  
procedure Pop(var S: Stack; var D: TypData);
```

```
implementation
```

```
procedure Error;
```

```
begin
```

```
    {řešení podle implementace}
```

```
    {zvnějšku není tato procedura dostupná}
```

```
end;
```

```
procedure Init(var S: Stack);
```

```
begin S:=nil
```

```
end;
```

```
{... podobně všechny ostatní operace}
```

```
end.
```

Použití modulu v programu

```
program UzijModul;
uses Zasobnik;
var R: string;
    S: Stack;
begin
    Init(S);
    while not eof do begin
        readln(R);
        Push(S, R)
    end;
    while not Empty(S) do begin
        Remove(S, R);
        writeln(R);
    end;
end.
```


- Implementovaný datový typ lze využít pro jakoukoliv úlohu – nezáleží na datech

- Implementovaný datový typ lze využít pro jakoukoliv úlohu – nezáleží na datech
- Pro manipulaci se strukturními prvky to neznamena žádnou změnu

- Implementovaný datový typ lze využít pro jakoukoliv úlohu – nezáleží na datech
- Pro manipulaci se strukturními prvky to neznamena žádnou změnu
- V případě manipulace s daty je nezbytné svěřit rozhodující části uživateli modulu

- Implementovaný datový typ lze využít pro jakoukoliv úlohu – nezáleží na datech
- Pro manipulaci se strukturními prvky to neznamena žádnou změnu
- V případě manipulace s daty je nezbytné svěřit rozhodující části uživateli modulu
- Na toto použití je nutné provést v modulu odpovídající přípravu

- Implementovaný datový typ lze využít pro jakoukoliv úlohu – nezáleží na datech
- Pro manipulaci se strukturními prvky to neznamena žádnou změnu
- V případě manipulace s daty je nezbytné svěřit rozhodující části uživateli modulu
- Na toto použití je nutné provést v modulu odpovídající přípravu
- Modifikace modulu: místo konkrétního datového typu složky se použije obecný ukazatel

- Implementovaný datový typ lze využít pro jakoukoliv úlohu – nezáleží na datech
- Pro manipulaci se strukturními prvky to neznamena žádnou změnu
- V případě manipulace s daty je nezbytné svěřit rozhodující části uživateli modulu
- Na toto použití je nutné provést v modulu odpovídající přípravu
- Modifikace modulu: místo konkrétního datového typu složky se použije obecný ukazatel
- Jediná změna v modulu:

```
|     TypData = pointer;
```

Změna v použití modulu

```
program UzijModul;
uses Zasobnik;
var R: string;
    S: Stack;
    {*} P: pointer;
begin Init(S);
    while not eof do begin
        readln(R);
        {*} GetMem(P, Length(R)+1);
        {*} P^:=R;
        Push(S, P)
    end;
    while not Empty(S) do begin
        Remove(S, P);
        {*} writeln(string(P^));
    end;
end.
```

Implementace další operace

Implementace další operace

- Vyhledání prvku v zásobníku – výsledkem hledání je logická hodnota

Implementace další operace

- Vyhledání prvku v zásobníku – výsledkem hledání je logická hodnota
- Modifikace úlohy: do zásobníku se budou vkládat jen ty hodnoty, které tam ještě nejsou

Implementace další operace

- Vyhledání prvku v zásobníku – výsledkem hledání je logická hodnota
- Modifikace úlohy: do zásobníku se budou vkládat jen ty hodnoty, které tam ještě nejsou
- V modulu: operace `Pritomen`

```
function Pritomen(S: Stack;  
                  D: TypData): boolean;  
  
  begin Pom:=S;  
        while (Pom<>nil) and  
          {JAK zjistit, že nebylo nalezeno?}  
        do Pom:=Pom^.Dalsi;  
        Pritomen:=Pom<>nil  
  
  end;
```

```
type TypNerovno = function (A, B: TypData):  
                                boolean;  
  
function Pritomen(S: Stack; D: TypData;  
                Nerovno: TypNerovno): boolean;  
  
begin Pom:=S;  
      while (Pom<>nil) and  
            Nerovno(Pom^.Data, D)  
      do Pom:=Pom^.Dalsi;  
      Pritomen:=Pom<>nil  
  
end;
```



```
function MojeNerovno(X, Y: TypData): boolean;  
  begin MojeNerovno:=string(X^)<>string(Y^)  
  end;  
  
...  
  
  while not eof do begin  
    readln(R);  
    GetMem(P,Length(R)+1);  
    string(P^):=R;  
    if not Pritomen(S, P, @MojeNerovno)  
      then Push(S, P)  
      else FreeMem(P,Length(R)+1)  
  end;
```

Modifikace příkladu

- Rozhodneme se, že místo řetězců budou nyní na vstupu reálná čísla, úloha jinak zůstává stejná

- Rozhodneme se, že místo řetězců budou nyní na vstupu reálná čísla, úloha jinak zůstává stejná
- Změny v modulu nebudou žádné

- Rozhodneme se, že místo řetězců budou nyní na vstupu reálná čísla, úloha jinak zůstává stejná
- Změny v modulu nebudou žádné
- Program užívající modul se změní jen minimálně:

```
program UzijModul;  
uses Zasobnik;  
var R: real;  
    S: Stack;  
    P: pointer;  
function DruheNerovno(X, Y: TypData): boolean;  
begin DruheNerovno:=real (X^)<>real (Y^)  
end;
```

```
begin Init(S);  
    while not SeekEof do begin  
        read(R);  
        GetMem(P, SizeOf(R));  
        real(P^) := R;  
        if not Pritomen(S, P, @DruheNerovno)  
            then Push(S, P)  
            else FreeMem(P, SizeOf(R))  
    end;  
    while not Empty(S) do begin  
        Remove(S, P);  
        writeln(real(P^));  
    end;  
end.
```