Definice funkcí a procedur

- Mnoho operací provozujeme opakovaně, proto je hloupé programovat je při každém použití znovu.

Definice funkcí a procedur

- Mnoho operací provozujeme opakovaně, proto je hloupé programovat je při každém použití znovu.
- Procedury a funkce představují možnost je naprogramovat jednou a použít mnohokrát.

Definice funkcí a procedur

- Mnoho operací provozujeme opakovaně, proto je hloupé programovat je při každém použití znovu.
- Procedury a funkce představují možnost je naprogramovat jednou a použít mnohokrát.
- Procedura: Součást programu schopná přijmout parametry a zpracovat.

Definice funkcí a procedur

- Mnoho operací provozujeme opakovaně, proto je hloupé programovat je při každém použití znovu.
- Procedury a funkce představují možnost je naprogramovat jednou a použít mnohokrát.
- Procedura: Součást programu schopná přijmout parametry a zpracovat.
- Funkce: Součást programu schopná přijmout parametry, zpracovat a vrátit výsledek.

Definice funkcí a procedur

- Mnoho operací provozujeme opakovaně, proto je hloupé programovat je při každém použití znovu.
- Procedury a funkce představují možnost je naprogramovat jednou a použít mnohokrát.
- Procedura: Součást programu schopná přijmout parametry a zpracovat.
- Funkce: Součást programu schopná přijmout parametry, zpracovat a vrátit výsledek.
- Příklady: Přejdi ulici, vypiš hlášení; dojeď vlakem někam, spočítej faktoriál.

Definice funkcí

`function` nazev(argument :`typ`;...):`navratovy_typ`

- Zahajujeme klíčovým slovem function, následuje název funkce,

Definice funkcí

`function` nazev(argument :`typ`;...):`navratovy_typ`

- Zahajujeme klíčovým slovem function, následuje název funkce,
- argumenty (parametry) zapisujeme do kulatých závorek jako bychom definovali proměnné.

Definice funkcí

`function` nazev(argument :`typ`;...):`navratovy_typ`

- Zahajujeme klíčovým slovem function, následuje název funkce,
- argumenty (parametry) zapisujeme do kulatých závorek jako bychom definovali proměnné.
- Jednotlivé parametry oddělujeme středníkem,

Definice funkcí

`function` nazev(argument :`typ`;...):`navratovy_typ`

- Zahajujeme klíčovým slovem function, následuje název funkce,
- argumenty (parametry) zapisujeme do kulatých závorek jako bychom definovali proměnné.
- Jednotlivé parametry oddělujeme středníkem,
- za dvojtečkou pak uvedeme návratový typ dotyčné funkce.

Definice funkcí

`function` nazev(argument :`typ`;...):`navratovy_typ`

- Zahajujeme klíčovým slovem function, následuje název funkce,
- argumenty (parametry) zapisujeme do kulatých závorek jako bychom definovali proměnné.
- Jednotlivé parametry oddělujeme středníkem,
- za dvojtečkou pak uvedeme návratový typ dotyčné funkce.
- Návratová hodnota se přiřadí do proměnné jmenující se stejně jako příslušná funkce.

Scope resolution

- Kromě globálních proměnných vznikají i proměnné *lokální*.

Příklad:

```
function secti(a:integer; b:integer):integer;
begin
        secti:=a+b;
        a:=0;
end;
begin
        x:=5; y:=10; c:=secti(x,y);
        writeln(x);
end.
```

Scope resolution

- Kromě globálních proměnných vznikají i proměnné *lokální*.
- Pokud je konflikt v názvu globální a lokální proměnné, platí ta lokální, jako v minulém příkladu.

Příklad:

```
function secti(a:integer; b:integer):integer;
begin
        secti:=a+b;
        a:=0;
end;
begin
        x:=5; y:=10; c:=secti(x,y);
        writeln(x);
end.
```

Scope resolution

- Kromě globálních proměnných vznikají i proměnné *lokální*.
- Pokud je konflikt v názvu globální a lokální proměnné, platí ta lokální, jako v minulém příkladu.
- Proměnné jsou předávány hodnotou, tedy hodnota proměnné je okopírována.

Příklad:

```
function secti(a:integer; b:integer):integer;
begin
        secti:=a+b;
        a:=0;
end;
begin
        x:=5; y:=10; c:=secti(x,y);
        writeln(x);
end.
```

Předání argumentu referencí (odkazem)

Co když naopak chceme obsah předávané proměnné ve funkci změnit?
Použijeme při popisu argumentu klíčové slovo `var`:

```
function f(var a:integer; b:integer):integer;
begin
        a:=5;
        b:=5;
end;
...
x:=0; y:=0; a:=f(x,y);
writeln(x); writeln(y);
...
```

Výsledek: 5 a 0; není-li referencí předána proměnná $\Rightarrow$ chyba!

Funkce bez parametrů

Má smysl definovat funkci bez parametrů (kupř. chceme-li načítat).
V tom případě můžeme vynechat kulaté závorky jak při definici,
tak při volání:

```
function x:integer;
begin
        x:=10;
end;

...
a:=x;
...
```

Procedury

"Procedury jsou funkce, které nevracejí hodnotu."

```
procedure jmeno(argumenty);

... jmeno(argumenty);...
```

Příklad:

```
procedure vyp(a:integer;b:integer);
begin
        writeln(a); writeln(b);
        {Vypsali jsme argumenty}
end;

... vyp(5,10);...
```

Vnořené funkce a procedury

Proceduru či funkci lze definovat i uvnitř jiné procedury nebo funkce:

```
procedure f(a:integer);
    procedure g(b:integer);
    begin
        writeln('Proc. g v procedure f s par-em ',b);
    end;
begin
    writeln('Procedura f s parametrem ',a);
    g(2);{Volame vnorenou proc. g}
end;
```

Scope resolution

- Procedura/funkce vidí jen funkce (procedury) vnořené přímo do sebe (ne do svých vnořených funkcí/procedur)!
- Vnořené funkce vnášejí další zmatek do významu proměnných (a dokonce procedur a funkcí).
- Identifikátor má takový význam, jaký je v danou chvíli "nejlokálnější".

Příklad

```
procedure f(h:integer);
    procedure g(b:integer);
        procedure h(c:integer);
        begin
            writeln('Procedura h s par. ',c);
        end;
    begin
        writeln('Procedura g s parametrem ',b);
        h(5);
    end;
begin
    writeln('Funkce f s parametrem ',h);
    g(3); f(5); {zatim OK, ale h(4) udela chybu!}
end;
```

Rekurze

- Může mít dobrý smysl volat z funkce sebe sama.
- Této metodě říkáme **rekurze.**
- Rekurze není nic jiného, než vtipně pojmenovaná indukce!
- Příklady: Úředníci na úřadech, Faktoriál, Caesarův kód...

Úředníci na úřadech

- Člověk chce nechat provést úřední výkon.

Úředníci na úřadech

- Člověk chce nechat provést úřední výkon.
- Úředník požaduje vyplnění papírů vyžadující návštěvy dalších úřadů.

Úředníci na úřadech

- Člověk chce nechat provést úřední výkon.
- Úředník požaduje vyplnění papírů vyžadující návštěvy dalších úřadů.
- Řešení:

```
procedure vypln(musime_vyplnit:seznam_papiru);
var x:seznam_papiru;
for papir in musime_vyplnit do
begin
    x:=zeptej_se_urednika(papir);
    vypln(x);
end;
```

Faktoriál

- $n! = 1 \cdot 2 \cdot \ldots \cdot n$

Faktoriál

- $n! = 1 \cdot 2 \cdot \ldots \cdot n$
- Jak tuto funkci naprogramovat?

Faktoriál

- $n! = 1 \cdot 2 \cdot \ldots \cdot n$
- Jak tuto funkci naprogramovat?
- Cyklem:

```
fakt:=1;
for i:=1 to n do
    fakt:=fakt*i;
```

Faktoriál

- $n! = 1 \cdot 2 \cdot \ldots \cdot n$
- Jak tuto funkci naprogramovat?
- Cyklem:

```
fakt:=1;
for i:=1 to n do
    fakt:=fakt*i;
```

- nebo rekurzí.

Faktoriál rekurzí:

```
function faktorial(a:integer):integer;
begin
    if a<2 then
          faktorial:=1;
    else  faktorial:=a*faktorial(a-1);
end;
```

Jaká je složitost výpočtu funkce faktoriál?

Anotace

- Ordinalni typy - typ char, funkce ord, chr, succ, prev, inc, dec, Motivace: Máme dlouhé číslo (nebo číslo ve stringu).
- Zapis cisla v pozicni soustave, jeho vyhodnoceni Hornerovym schematem,
- Evaluace polynomu Hornerovym schematem,
- Umocnovani cisla,
- MAXINT, preteceni, dlouha cisla, typ real a jeho presnost.
- Operace s dlouhými čísly.

Ordinální datové typy

- Typy, jejichž hodnoty tvoří lineárně uspořádanou množinu (tedy pro každou dvojici je jasné, který prvek je větší).
- Z pascalských typů jsou to: `integer`, `longint`, `byte`, `char`, `word`, `boolean` a `shortint` (a další definované uživatelem, zejména výčtový typ a interval).
- Pro ordinální typy jsou definovány funkce "Ord", "Pred" a "Succ".

Funkce pro ordinální typy

- Funkce `Ord` vrátí ordinální hodnotu (tedy hodnotu pro část typů, konkrétně pro `integer`, `longint`, `byte`, `word` a `shortint`).
- Pro typ `char` vrátí ASCII hodnotu příslušného znaku. Umíme tedy zjistit ASCII-hodnotu jednotlivých znaků.
- Co naopak? Použijeme funkci `Chr`, která vrátí znak s příslušným číslem.
- Úloha: Jak převedeme číslo uložené v integeru na řetězec znaků?
- Řetězec je reprezentován jako pole znaků (až na drobná omezení).

Příklad

```
var a,i,j:integer; text,pom:string[255];
begin pom:='xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx';
      readln(a); i:=1; j:=1;
      while a<> 0 do
      begin pom[i]:=chr(a mod 10+48);
            a:=a div 10; i:=i+1;
      end;
      delete(pom,i,255-i); i:=i-1;
      text:=copy(pom,1,length(pom));
      while i>0 do
      begin text[j]:=pom[i];
            i:=i-1; j:=j+1;
      end;
      writeln(text);
end.
```

Hornerovo schéma

- Co když chceme konvertovat obráceně? (řetězec na integer)
- Použijeme tzv. Hornerovo schéma, tedy začneme od začátku řetězce (nejvyšší číslice).
- Zjistíme její hodnotu a postupujeme indukcí:
  Dosavadní výsledek vynásobíme deseti a přičteme číslici na dalším řádu.

číslo $a_n a_{n-1} a_{n-2} \ldots a_0$ zapsané v desítkovém zápisu je vlastně $a_n 10^n + a_{n-1} 10^{n-1} + \ldots + a_0$. A platí:

$$a_n 10^n + a_{n-1} 10^{n-1} + \ldots + a_0 = (\ldots((a_n * 10) + a_{n-1} * 10) + \ldots + a_1) * 10 + a_0$$

Tímto způsobem můžeme vyhodnocovat čísla zapsaná v různých (pozičních) soustavách (dvojkové, čtyřkové, pětkové, šestkové...).

Příklad

```
program x;
var a:string;
    i,hod:longint;
begin
     readln(a); i:=1; hod:=0;
     while i<=length(a) do
     begin
     hod:=10*hod+ord(a[i])-ord('0');
          i:=i+1;
     end;
     writeln(hod);
end.
```

Evaluace polynomu

- Máme zadán polynom $a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0$.

Evaluace polynomu

- Máme zadán polynom $a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0$.
- Chceme tento polynom vyhodnotit (evaluovat), tedy určit jeho hodnotu v zadaném bodě x.

Evaluace polynomu

- Máme zadán polynom $a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0$.
- Chceme tento polynom vyhodnotit (evaluovat), tedy určit jeho hodnotu v zadaném bodě x.
- Možnosti?

Evaluace polynomu

- Máme zadán polynom $a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0$.
- Chceme tento polynom vyhodnotit (evaluovat), tedy určit jeho hodnotu v zadaném bodě x.
- Možnosti?
- Hrubá síla (počítat $a_n x^n$, $a_{n-1} x^{n-1}$,... a sečíst)

Evaluace polynomu

- Máme zadán polynom $a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0$.
- Chceme tento polynom vyhodnotit (evaluovat), tedy určit jeho hodnotu v zadaném bodě x.
- Možnosti?
- Hrubá síla (počítat $a_n x^n$, $a_{n-1} x^{n-1}$,... a sečíst)
- nebo Hornerovo schéma

$$\sum_{i=0}^{n} a_i x^i = ((\ldots(a_n x + a_{n-1})x + \ldots + a_1)x + a_0).$$

Evaluace polynomu Hornerovým schématem

- 1: Načti koeficient u nejvyššího (dosud neprošlého) stupně,
- dosud načtené hodnoty vynásob hodnotou x,
- přičti hodnotu nově načteného koeficientu,
- GOTO 1;

Evaluace polynomu Hornerovým schématem

```
program nic;
var i,a,souc,stupen,x:integer;
{Vyhodnot polynom stupne stupen v bode x, do a
nacitej koeficienty}
begin
      readln(stupen); readln(x);
      souc:=0;
      for i:=0 to stupen do
      begin souc:=souc*x;
            readln(a);
            souc:=souc+a;
      end;
      writeln('Hodnota polynomu je: ',souc);
end.
```

Odbočka – labely a GOTO

- V Pascalu je možno provádět poměrně nekontrolované skoky po programu.
- Za sekcí globálních proměnných (`var`) vytvoříme sekci `label`, kde vyjmenujeme použité labely,
- následně můžeme těmito labely označit vybraná místa programu
- a pomocí `goto label;` udělat nepodmíněný skok.
- GOTO nepoužívejte, používám ho jen já v pseudokódu k pedagogickým účelům.

Fronta a zásobník

- Fronta je datová struktura osazená operacemi `zařaď` a `vyřaď`,
- `zařaď` zařadí daný prvek na konec fronty,
- `vyřaď` vyřadí daný prvek ze začátku fronty.
- Zásobník je datová struktura osazená operacemi `push` a `pull`.
- `push` přidá na konec zásobníku, `pull` vytáhne z konce zásobníku.

Figurky na šachovnici anebo prohledávání grafu - algoritmus vlny

- Na děravé šachovnici je umístěn král. Určete jak rychle se dostane na určené (cílové) pole?

Figurky na šachovnici anebo prohledávání grafu - algoritmus vlny

- Na děravé šachovnici je umístěn král. Určete jak rychle se dostane na určené (cílové) pole?
- Vlastně jde o úlohu prohledávání grafu.

Figurky na šachovnici anebo prohledávání grafu - algoritmus vlny

- Na děravé šachovnici je umístěn král. Určete jak rychle se dostane na určené (cílové) pole?
- Vlastně jde o úlohu prohledávání grafu.
- Podobné: Thesseus hledá Mínótaura.

Figurky na šachovnici anebo prohledávání grafu - algoritmus vlny

- Na děravé šachovnici je umístěn král. Určete jak rychle se dostane na určené (cílové) pole?
- Vlastně jde o úlohu prohledávání grafu.
- Podobné: Thesseus hledá Mínótaura.
- Rozdíl: Tehdy nám nešlo o nejkratší cestu.

Algoritmus vlny (myšlenky)

- Na šachovnici vysadíme na specifikované políčko mravence,
- mravenci polezou rovnoměrně všemi dostupnými směry,
- mravenci zkusí všechny cesty, rychleji než první mravenec se tam dostat nelze.
- Jiná intuice: Na příslušné poličko vychrstneme vodu,
- voda teče všemi dostupnými cestami, až doteče na cílové políčko.

Algoritmus vlny (implementace)

- Vytvoř prázdnou frontu f.
- Nastav vzdálenost do všech políček kromě startovního na nekonečno, vzdálenost do startovního nastav na 0.
- Přidej do f startovní políčko.
- Dokud není fronta f prázdná, opakuj:
  - a:=vyřaď(f);
  - Pro všechny sousedy z pole a zkus:
  - Pokud vzdálenost do z je větší než vzdálenost do a +1, nastav políčku z vzdálenost $a+1$ a zařaď(z, f).

Přednášející jde do posluchárny

aneb rekurze

- Přednášející při cestě po posluchárny na schodech vždycky buďto šlápne na následující schod, nebo jeden schod překročí.
- Kolika způsoby může vylézt do posluchárny F1?
  (schody nepočítejte, jejich počet odhadněte)
- Nápady?

Přednášející do posluchárny – řešení

- Vzniká rekurence $f_n = f_{n-1} + f_{n-2}$.

Přednášející do posluchárny – řešení

- Vzniká rekurence $f_n = f_{n-1} + f_{n-2}$.
- Rekurence není nic jiného, než matematický zápis rekurze.

Přednášející do posluchárny – řešení

- Vzniká rekurence $f_n = f_{n-1} + f_{n-2}$.
- Rekurence není nic jiného, než matematický zápis rekurze.
- Řešení:

```
function schody(a:integer):integer;
begin
    if a=1 then schody=1;
    else if a=2 then schody=2;
        else
            schody:=schody(a-1)+schody(a-2);
end;
```

Přednášející do posluchárny – řešení

- Vzniká rekurence $f_n = f_{n-1} + f_{n-2}$.
- Rekurence není nic jiného, než matematický zápis rekurze.
- Řešení:

```
function schody(a:integer):integer;
begin
    if a=1 then schody=1;
    else if a=2 then schody=2;
        else
            schody:=schody(a-1)+schody(a-2);
end;
```

- Jaký je problém?

Přednášející do posluchárny – řešení

- Vzniká rekurence $f_n = f_{n-1} + f_{n-2}$.
- Rekurence není nic jiného, než matematický zápis rekurze.
- Řešení:

```
function schody(a:integer):integer;
begin
    if a=1 then schody=1;
    else if a=2 then schody=2;
        else
            schody:=schody(a-1)+schody(a-2);
end;
```

- Jaký je problém?
- Ano, příšerná složitost.