Funkce – pokročilé možnosti

Úvod do programování 2

Tomáš Kühr

Funkce – co už víme

- Nebo alespoň máme vědět... ☺
- Co je to funkce?
- Co jsou to parametry funkce?
- Co je to deklarace a definice funkce? K čemu slouží?
- Jak funguje předávání parametrů do funkce (hodnotou a odkazem)?
- Co je to návratová hodnota funkce?
- Jak probíhá vrácení výsledku funkce?
- Jaké rozlišujeme rozsahy platnosti proměnných (nebo obecně identifikátorů)?
- Se kterou proměnnou se bude pracovat, pokud jich existuje více se stejným identifikátorem?

Rekurzivní funkce

Rekurzivní funkce

- Už (velmi dobře) známe z Paradigmat programování... ☺
- Funkce, která ve svém těle volá sama sebe
- Princip = postupné zjednodušování problému
- Až k elementární situaci (jejíž řešení známe či snadno spočítáme)
- Přerušení rekurze při splnění mezní podmínky
- Příklady:
  - Výpočet faktoriálu,
  - Výpočet fibonacciho čísla,
  - QuickSort
  - MergeSort
  - a další (teoreticky cokoli s cyklem).

Příklad – výhody a nevýhody rekurze

```
long double factrec(unsigned int n){
    if (n == 0) return 1;
    return n * factrec(n - 1);
}
long double factiter(unsigned int n){
    long double out = 1;
    unsigned int i;
    for (i = 2; i <= n; i++)
        out *= i;
    return out;
}
int main(){
    long double v1, v2;
    v1 = factrec(5);
    v2 = factiter(5);
    return 0;
}
```

Výhody a nevýhody rekurze v C

- Výhody:
  - Stručnější a přehlednější zápis (v některých situacích)
  - Přímočarý návrh algoritmu (u některých problémů)
- Nevýhody:
  - Větší spotřeba paměti
  - Pomalejší běh programu
- Rekurzi obvykle použijeme:
  - Pokud řešíme problém, který ze své podstaty „rekurzivní“ (faktoriál, fibonaccioho čísla, ...)
  - Pokud implementujeme algoritmy založené na principu „rozděl a panuj“
  - A pokud nám moc nezáleží na rychlosti a požadavcích na paměť...

Ukazatelé na funkce

Vytvoření ukazatele na funkci

- Ukazatel je definován vždy pro určitý typ funkcí
  - Daný počet a typy parametrů
  - Daný typ návratové hodnoty
- Příklad definice (funkce s 1 parametrem double vracející double):

```
double(*p_fd2d)(double);
```

- Příklad definice s inicializací:

```
double polynomA(double x){
   return 3 * x*x + 4 * x - 10;
}
double(*p_fd2d)(double) = polynomA;
```

- Příklad definice s využitím vlastního datového typu:

```
typedef double(*P_FD2D)(double);
P_FD2D p_fd2d = polynomA;
```

Práce s ukazateli na funkce

- Do ukazatele můžeme kdykoli vložit adresu funkce:

```
P_FD2D dalsi_ukazatel;
p_fd2d = polynomA;
dalsi_ukazatel = p_fd2d;
```

- Funkci uloženou v ukazateli můžeme zavolat:

```
double v;
v = polynomA(-2);
v = (*p_fd2d)(-2);
v = dalsi_ukazatel(-2);
```

- Výše uvedené možnosti volání funkce jsou zcela ekvivalentní
- Vztah mezi ukazateli na funkce a identifikátory funkcí je podobný jako vztah polí a ukazatelů na první prvek pole
- Identifikátor funkce se chová jako konstantní ukazatel na funkci

Funkce jako parametr funkce

- Definice funkce s parametrem typu ukazatel na funkci:

```
double *map(double(*fce)(double), double *vstup, double pocet);
```

- Volání funkce s parametrem typu ukazatel na funkci:

```
double na3(double x){
    return x*x*x;
}

double pole[] = { 1, 2, 3, 4, 5 };
double *out = map(na3, pole, 5);
```

- Varianta s předáním funkce pomocí ukazatele:

```
double(*p_na3)(double) = na3;
double *out = map(p_na3, pole, 5);
```

Pole funkcí

- Ukazatele v poli musí mít stejný počet a typy parametrů a stejný typ návratové hodnoty
- Definice pole ukazatelů na funkce:
```
double(*pole_fci[5])(double);
```
- Příklad definice s inicializací:
```
double(*pole_fci[])(double) = {na3,na4,sin,cos,tan};
```
- Varianta s použitím dříve definovaného typu:
```
P_FD2D pole_fci[] = {na3, na4, sin, cos, tan};
```
- Volání funkcí z pole:
```
double vysledek = pole_fci[1](-10);
```

Funkce s proměnlivým počtem parametrů

Princip FPPP

- Při volání funkce se parametry kopírují na zásobník.
- Pokud známe
  - adresu parametru na zásobníku
  - a typ tohoto parametru,
- můžeme přistoupit k následujícímu parametru.
- Problém: struktura zásobníku se může lišit
- Díky standardní knihovně **stdarg.h** ji ale nemusíme znát
- FPPP musí splňovat následující omezení
  - mít alespoň jeden pojmenovaný / pevný parametr
  - musí být znám i počet parametrů a jejich datové typy.

Obvyklé varianty FPPP

- Počet a typy parametrů dány textovým řetězcem
  (např. `printf`, `scanf`)
- Pevně dán typ parametrů a předá se jejich počet
  (např. výpočet sumy, průměru čísel)
- Pevně dán typ parametrů a jako poslední parametr se použije „zarážka“,
  tj. nějaká předem určená speciální hodnota (`NULL`, `'\0'`, `0.0`, …)
  (např. spojení textových řetězců)

Definice a použití FPPP

- Při deklaraci i definici FPPP se používá výpustka („...”)
- Výpustka oznamuje překladači, že skutečný počet parametrů při volání funkce může být větší
- Příklady deklarace funkcí:

```
double prumer(int pocet, double prvni, ...);
int my_print(char* format, ...);
```

- Příklady definice funkcí:

```
double prumer(int pocet, double prvni, ...){ tělo }
int my_print(char* format, ...){ tělo }
```

- Příklady volání funkcí:

```
double vysledek = prumer(5, 1.2, 2.3, 4.3, 2.5, 7.1);
print("Průměr je %g. \n", vysledek);
```

Makra ze stdarg.h

- Knihovna **stdarg.h** poskytuje datový typ `va_list` a makra `va_start`, `va_arg` a `va_end`
- Proměnné typu `va_list` uchovávají informace o zásobníku
- Vytvoření a inicializace proměnné typu `va_list`:

```
va_list zasobnik;
va_start(zasobnik, posledni_pojmenovany);
```

- Přístup k dalšími parametru v pořadí:

```
data = va_arg(zasobnik, double);
```

- Ukončení práce se zásobníkem: `va_end(zasobnik);`
- Pozor na datové typy parametrů:

```
double vysledek = prumer(5, 1.0, 3.0, 2.0); //spravne
double vysledek = prumer(5, 1, 3.0, 2.0);//spravne
double vysledek = prumer(3, 1, (double)3, (double)2); //spravne
double vysledek = prumer(3, 1, 3, 2); //chybne - proc?
```

Příklad FPPP

```
double prumer(int pocet, double prvniCislo, ...){
    double vysledek;
    va_list zasobnik;
    int i;
    vysledek = prvniCislo;
    i = pocet - 1;
    va_start(zasobnik, prvniCislo);
    while (i > 0) {
        vysledek += va_arg(zasobnik, double);
        i--;
    }
    va_end(zasobnik);
    return vysledek/pocet;
}
```

Příklad FPPP s komentáři

```
double prumer(int pocet, double prvniCislo, ...){
   // prvni cislo je povinny/pojmenovany parametr, ostani volitelne
   double vysledek;    // soucet cisel
   va_list zasobnik;    // ”ukazatel na zasobnik”
   int i;    // ridici promenna cyklu

   vysledek = prvniCislo;    // vložíme primo prvni cislo
   i = pocet - 1;    // prvni uz mame zpracovane
   va_start(zasobnik, prvniCislo);    // inicializace
   while (i > 0) {    // dokud nezpracujeme ocekavany pocet
      vysledek += va_arg(zasobnik, double);    // nacteni dalsiho
      i--;    // zbyva nacist o 1 mene
   }
   va_end(zasobnik);    // konec prace se zasobnikem
   return vysledek/pocet;    // dokonceni vypoctu
}
```