

Funkce s proměnným počtem parametrů

Jiří Zacpal



DEPARTMENT OF COMPUTER SCIENCE
PALACKÝ UNIVERSITY, OLOMOUC

KMI/ZP2 Základy programování 2

- při volání funkce se skutečné parametry kopírují na zásobník (paměť používaná pro uložení lokálních proměnných a parametrů funkce)
- pokud tedy známe adresu a typ některého parametru, můžeme přistoupit k následujícímu parametru
- díky makrům ze `stdarg.h` nemusíme znát strukturu zásobníku (je implementačně závislá)
- FPPP musí mít vždy alespoň jeden pevný parametr, od kterého začínáme na zásobníku hledat
- musí být jasné, kolik parametrů v paměti následuje (je znám počet nebo je použita nějaká zarážka)
- musí být známé typy parametrů

Typické příklady FPPP



- počet parametrů a jejich typy jsou dány předaným formátovacím textovým řetězcem (např. funkce `printf`)
- předpokládá se určitý typ parametrů a předá se jejich počet (výpočet sumy, průměru čísel apod.)
- předpokládá se určitý typ parametrů a konec seznamu parametrů je označen zarážkou – domluvenou hodnotou (např. spojení textových řetězců)

Deklarace a volání FPPP

- při deklaraci (definici) FPPP se používá výpustek („...“)

- definice funkce:

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

- volání funkce:

```
double prum;
```

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

Zpracování parametrů – makra z stdarg.h

- poskytuje programátorům typ `va_list` a makra `va_start()`, `va_arg()` a `va_end()`
- inicializace „ukazatele na zásobník“ (proměnné typu `va_list`):

```
va_list param;  
va_start(param, první);
```
- přístup k hodnotě parametru a posun „ukazatele“ na začátek paměti dalšího parametru:

```
cislo = va_arg(param, double);
```
- ukončení práce s „ukazatelem na zásobník“:

```
va_end(param);
```
- při práci s FPPP je nutné věnovat zvýšenou pozornost použitým datovým typům

```
prum = prumer(3, 1, (double)3, (double)2);
```

Příklad 1



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

Příklad 2



```
long double prumer(char* format, ...)
{
    long double out=0;
    int i=0;
    int pocet=strlen(format);
    va_list param;
    va_start(param, format);
    while (i<pocet)
    {
        switch(format[i])
        {
            case 'i':out+=va_arg(param, int);break;
            case 'd':out+=va_arg(param, double);break;
            case 'l':out+=va_arg(param, long double);break;
        }
        i++;
    }
    va_end(param);
    out = out/pocet;
    return out;
}
```

Bodovaný úkol



Napište v jazyku C funkci

`komplexni suma(int pocet, ...),`

která vypočítá součet předaných komplexních čísel. Počet sčítaných čísel je určen pevným parametrem `pocet`, za nímž pak ve volání funkce následují hodnoty, které má funkce sčítat. Pro práci s komplexními čísly využijte vámi definovaný strukturovaný datový typ `komplexni`.

Příklad použití:

```
komplexni vysledek;  
komplexni a = {3.1, -2.3};  
komplexni b = {0.5, -3};  
komplexni c = {0, 1.2};  
vysledek = suma(3, a, b, c);  
printf("Suma je %g + %gi. \n",  
vysledek.realna, vysledek.imaginari);
```

The screenshot shows a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The window contains the output of the C program: "Suma je 3.6 + -4.1i." followed by "Press any key to continue . . .". The cursor is positioned at the end of the second line.