10. lekce

Úvod do jazyka C – 4. část

Funkce, rekurze

Miroslav Jílek

Funkce

- samostatná část programu, ve které se zpracovávají data

Funkce

- samostatná část programu, ve které se zpracovávají data
- při deklaraci funkce definujeme datový typ výstupu, název funkce a typy a názvy parametrů

Funkce

- samostatná část programu, ve které se zpracovávají data
- při deklaraci funkce definujeme datový typ výstupu, název funkce a typy a názvy parametrů
- po exekuci funkce se běh (run) aplikace vrací na místo, ze kterého byla funkce volána (call)

Funkce

- samostatná část programu, ve které se zpracovávají data
- při deklaraci funkce definujeme datový typ výstupu, název funkce a typy a názvy parametrů
- po exekuci funkce se běh (run) aplikace vrací na místo, ze kterého byla funkce volána (call)
- funkce může volat další funkci nebo sama sebe (rekurze)

Funkce

- samostatná část programu, ve které se zpracovávají data
- při deklaraci funkce definujeme datový typ výstupu, název funkce a typy a názvy parametrů
- po exekuci funkce se běh (run) aplikace vrací na místo, ze kterého byla funkce volána (call)
- funkce může volat další funkci nebo sama sebe (rekurze)

Proč používáme funkce?

Funkce

- samostatná část programu, ve které se zpracovávají data
- při deklaraci funkce definujeme datový typ výstupu, název funkce a typy a názvy parametrů
- po exekuci funkce se běh (run) aplikace vrací na místo, ze kterého byla funkce volána (call)
- funkce může volat další funkci nebo sama sebe (rekurze)

Proč používáme funkce?

– pro přehlednost (clarity, ясность) kódu

Funkce

- samostatná část programu, ve které se zpracovávají data
- při deklaraci funkce definujeme datový typ výstupu, název funkce a typy a názvy parametrů
- po exekuci funkce se běh (run) aplikace vrací na místo, ze kterého byla funkce volána (call)
- funkce může volat další funkci nebo sama sebe (rekurze)

Proč používáme funkce?

- pro přehlednost (clarity, ясность) kódu
- zkrácení kódu – opakující části se vloží do jedné funkce, kterou voláme z různých míst programu

Vstupní a výstupní parametry funkce

- definujeme do závorky za jménem funkce
- parametr definujeme: datový typ a jméno parametru, jednotlivé parametry oddělujeme čárkou

Příklad vstupních parametrů:

int mojefunkce(int a, char b)

Příklad výstupních parametrů:

Jako výstupní parametry se používají pointery (ukazatele)!
To znamená, že hodnoty ukládáme na adresy a z těchto adres v dalších částech programu čteme hodnoty.

Vstupní a výstupní parametry funkce

- definujeme do závorky za jménem funkce
- parametr definujeme: datový typ a jméno parametru, jednotlivé parametry oddělujeme čárkou

Vstupní a výstupní parametry funkce

- definujeme do závorky za jménem funkce
- parametr definujeme: datový typ a jméno parametru, jednotlivé parametry oddělujeme čárkou

Příklad vstupních parametrů:

int mojefunkce(int a, char b)
int mojefunkce(void) - funkce nemá žádné parametry

Vstupní a výstupní parametry funkce

- definujeme do závorky za jménem funkce
- parametr definujeme: datový typ a jméno parametru, jednotlivé parametry oddělujeme čárkou

Příklad vstupních parametrů:

int mojefunkce(int a, char b)
int mojefunkce(void) - funkce nemá žádné parametry

Příklad výstupních parametrů:

Jako výstupní parametry se používají pointery (ukazatele)!
To znamená, že hodnoty ukládáme na adresy a z těchto adres v dalších částech programu čteme hodnoty.

Příklad funkce s výstupním parametrem

```
void mojefunkce (int *a)
{
    *a=10;
}

int main(void)
{
    int *b, c;
    b=(int*)malloc(sizeof(int));    //alokujeme místo v RAM pro int
    mojefunkce(b);
    c=*b;
    return 0;
}
```

int *b; - *zde alokujeme místo v paměti pro uložení adresy (32/64 bitů)*
b=(int*)malloc(sizeof(int)); - *alokujeme místo v RAM pro integer (4 Byty) a do proměnné b se vloží nová adresa, kde operační systém alokoval místo*

Při volání funkce moje funkce posíláme jako parametr adresu, kterou program vloží do proměnné a (pointer). Ve funkci na tuto adresu vložíme hodnotu 10.

Návratová hodnota funkce

- typ definujeme před názvem funkce
- jestli použijeme void, pak funkce nevrací žádnou hodnotu

Návratová hodnota funkce

- typ definujeme před názvem funkce
- jestli použijeme void, pak funkce nevrací žádnou hodnotu

Použití příkazu return

- ukončuje funkci (z libovolného místa funkce)
- jestli je za příkazem return hodnota nebo proměnná, pak hodnotu nebo hodnotu proměnné vrací na místo, odkud byla funkce volaná – vrací konstantní hodnotu nebo hodnotu jedné proměnné
- pokud potřebujeme z procedury vracet více hodnot, pak máme dvě možnosti
  - v proceduře měníme hodnoty proměnných typu pointer – tím změna hodnoty těchto proměnných (hodnot, na které proměnné typu pointer ukazují) v proceduře bude také v mainu
  - v returnu použijete proměnnou typu **struct**

```
#include <stdio.h>
#include <stdlib.h>

typedef struct BALIKDAT
{
	int Hodnota1;
	int Hodnota2;
} Balikdat;

Balikdat MojeFunkce (int Vstup)
{
	Balikdat Data;
	Data.Hodnota1=3*Vstup;
	Data.Hodnota2=3+Vstup;
	return Data;
}

int main (void)
{
	int C=10;
	Balikdat NovaData;
	NovaData=MojeFunkce(C);
	printf("Hodnnota1 je %d, Hodnota2 je %d.\n",NovaData.Hodnota1, NovaData.Hodnota2);
	return 0;
}
```

```
Hodnnota1 je 30, Hodnota2 je 13.
```

Příklady použití funkcí:

Funkce, která vrátí menší číslo (jedno ze dvou):

```
int maximum (int e, int f)                    // funkce
{
      if (e>f) return e; else return f;
}

int main (void)                               // hlavní program
{
      int a, b=1, c=2;
      ...
      a=maximum(b, c);
      ...
}
```

Rekurzivní funkce

- rekurzivní funkce volá (call) sama sebe
- každé rekurzivní zavolání vytváří nové instance proměnných – např. jestli funkci rekurzivně voláme stokrát, pak v RAM budou její proměnné alokovány také stokrát. To může v extrémním případě způsobit problém s dostatkem paměti. Proto musíme vymyslet rekurzi tak, aby byla konečná.

Rekurze (výpis mocnin prvních deseti přirozených čísel)

```
void mocniny (int C)
{
        C++;
        printf("%d, ",C*C);
        if (C==10) return;
        mocniny(C);
}

int main (void)
{
        mocniny(0);
        return 0;
}
```

Rekurze (výpočet faktoriálu):

```
int faktorial (int e)
{
	if (e<2) return 1;
	return e * faktorial(e-1);
}

int main (void)
{
	int a;
	...                    // test: a>=0
	a=faktorial(a);
	...
}
```

Rekurze (výpočet faktoriálu):

```
int faktorial (int e)
{
        if (e<2) return 1;
        return e * faktorial(e-1);
}

int main (void)
{
        int a;
        ...                     // test: a>=0
        a=faktorial(a);
        ...
}
```

Princip: **n! = n*(n-1)!**

Pro a=4 bude

```
level 1:  return faktorial(3)*4;
level 2:        return faktorial(2)*3;
level 3:              return faktorial(1)*2;
level 4:                    return 1;
Výsledek:                         4*3*2*1
```

Ukázka vstupu dat

Vstup do statické proměnné:

```
if (scanf("%d",&i) != 1)
{
	printf("Nespravny vstup\n");
	return 1;
}
```

Tímto způsobem prověříme, že vložení hodnoty do proměnné proběhlo správně. Jestli bude proměnná i typu integer, pak musí být hodnota typu integer. Jestli není, bude chyba – počet správných vložení nebude 1.

Jestli použijeme vložení více hodnot do více proměnných, pak počet správných vložení musí být roven počtu proměnných.

```
if (scanf("%d %d",&i, &j) != 2) … oddělovací znak mezera
if (scanf("%d,%d",&i, &j) != 2) … oddělovací znak čárka
```

Jestli uživatel zadá hodnotu 1.5, potom se do proměnné i vloží hodnota 1 (%d je integer) a do proměnné j přijde . (tečka). A to bude chyba!!!

Vstup do statické proměnné, vstup se opakuje, dokud není zadaná správná hodnota

```
printf("Zadej vstupni hodnotu:");
while(scanf(”%d”, &C)!=1)
{
    printf("Nespravny vstup – zadej znovu!\n");
    fflush(stdin);      //vyčistí vstupní proud – znaky z klávesnice, které jsou v bufferu
}
```

Vstup do statického pole:

```
#include <stdio.h>

int main (void)
{
    int pole[10], i;
    for (i = 0; i < 10; i++)
    {
        if (scanf("%d", pole + i) != 1)          // pole + i  <=>  &(pole[i])
        {
            printf("Nespravny vstup\n");
            return 1;
        }
    }
    for (i = 0; i < 10; i++)
        printf("%d, ", pole[i]);
    return 0;
}
```

Vstup do dynamického pole: *(konec zadávání: Enter za posledním číslem, Ctrl+Z)*

```
#include <stdio.h>
#include <stdlib.h>

int main (void)
{
	int i=0, j=10;
	int * Pole=(int*)malloc(j*sizeof(int)); //ukazatel na integer lze take : int * Pole; Pole=(int*)malloc(j*sizeof(int));
	printf("Zadej radu cisel, konec zadavani je CTRL+Z: ");
	while (!feof(stdin))
	{
		if ((scanf("%d", & (Pole[i++])) != 1) && !feof(stdin))  //lze take: Pole + i
		{
			printf("Nespravny vstup\n");
			return 1;
		}
		if (i==j)
		{
			j*=2;
			Pole = realloc(Pole, j*sizeof(int));
		}
	}
	for (j = 0; j < i-1; j++) printf("%d, ", Pole[j]);
	return 0;
}
```

Vstup do dynamického pole charů:

```
#include <stdio.h>
#include <stdlib.h>

int main (void)
{
	int i=0, j=10;
	char * Pole=(char*)malloc(j*sizeof(char));
	printf("Zadej radu znaku: ");
	while (Pole[i-1]!='\n')				//konce zadavani je Enter
	{
		scanf("%c", &(Pole[i++]));
		if (i==j)
		{
			j*=2;
			Pole = realloc(Pole, j*sizeof(char));
		}
	}
	for (j = 0; j < i-1; j++) printf("%c, ", Pole[j]);
	return 0;
}
```

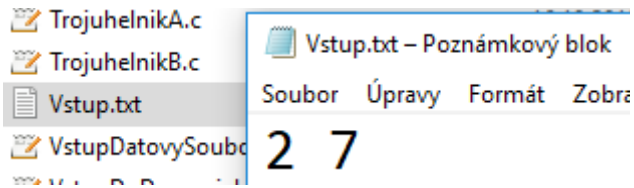
Vstup z datového souboru:

Program:

```
#include <stdio.h>

int main (void)
{
	int a, b;
	printf("Zadej dve cisla oddelena mezerou: ");
	if (scanf("%d %d",&a,&b)!=2)
	{
		printf("Chyba na vstupu!\n");
		return 1;
	}
	printf("Soucet cisel je %d.\n",a+b);
	return 0;
}
```

Datový soubor:



Konsole:

```
D:\Moje výuka\Přípravy Jílek\Informatika\1. semestr\Řešené příklady>a.exe <Vstup.txt
Zadej dve cisla oddelena mezerou: Soucet cisel je 9.
```