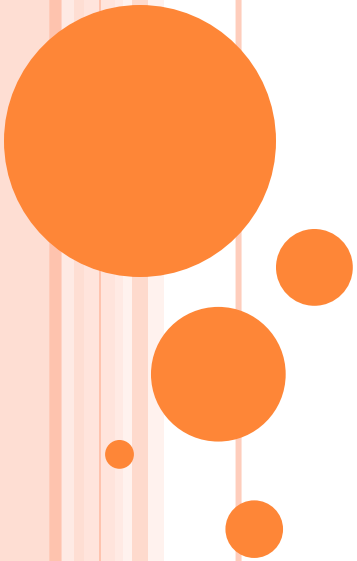


Programovacie jazyky pre vstavané systémy

Ukazovatele, polia, reťazce

Otázky z minulej prednášky

- Čo je to endianita?
- Čo sú to pamäťové triedy?
- Ako je organizovaný pamäťový priestor procesu?
- Aké príkazy pozná jazyk C?
- Čo je to „sequence point“?
- Ako vyzerá číslo 0.125 v dvojkovej sústave?

Typ ukazovateľ (smerník)

- Objektový typ, ktorý odkazuje na funkciu, na iný objekt (premenná, pole, prvok poľa,...) alebo na NULL.
- Môže byť asociovaný s nejakým údajovým typom, alebo je netypový (void*).
- Premenná typu ukazovateľ uchováva adresu funkcie alebo iného objektu.
- Adresa je celé číslo, pričom jeho veľkosť závisí od architektúry (napr. 4 alebo 8 bajtov).
- Adresu funkcie alebo objektu môžeme získať pomocou operátora &.
- Ukazovateľ môže uchovávať odkaz na premennú, ktorá je tiež ukazovateľom.

Dátové modely

- Na základe veľkosti celočíselných typov a typu ukazovateľ sa rozlišuje niekoľko dátových modelov, z ktorých dominujú:

	Dátový model	int	long	long long	pointer	Použitie
32 bitový systém	**LP32 2/4/4**	16	32	64	32	Win 16 API
	ILP32 4/4/4	32	32	64	32	Win 32 API, Unix (Linux, Mac OS x)
64 bitový systém	**LLP64 4/4/8**	32	32	64	64	Win64 API
	LP64 4/8/8	32	64	64	64	Unix (Linux, Mac OS x)

Typ ukazovateľ - syntax

- Definícia a inicializácia ukazovateľa (smerníka):

```
int x;
int *px = &x;
int **p_px;
p_px = &px;
```

- Použitie ukazovateľa (smerníka):

```
*px = 10;
printf("x = %d\n", x);
printf("x = %d\n", *px);
printf("x = %d\n", **p_px);
printf("&x = %p\n", &x);
printf("&x = %p\n", px);
printf("&x = %p\n", *p_px);
```

Operácie so smerníkmi

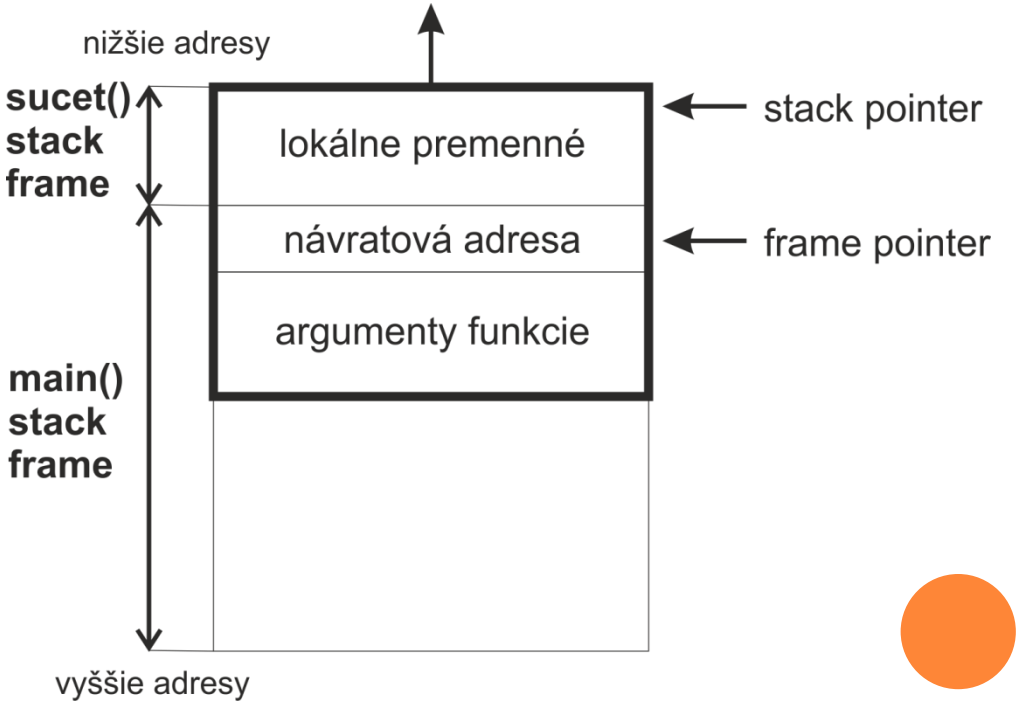
- & – získanie adresy objektu (alebo funkcie)
- * – dereferencia; prístup k objektu, ktorého adresu máme uloženú v smerníku
- Smerníková aritmetika – nasledujúce operátory slúžia na posun smerníka:
  - ++, -- (pre aj post)
  - +, - (binárne)
  - p2-p1 – počet položiek medzi smerníkmi p2 a p1
- O koľko bajtov sa smerník posunie pri aplikácii predchádzajúcich operátorov?
- Relačné operátory:
  - ==, !=, <, <=, >, >= – oba operandy sú smerníky rovnakého typu alebo jeden je smerník a druhý je konštanta NULL

Volanie funkcie

- V okamžiku volania funkcie (operátor ()) je nutné alokovať pamäťový priestor pre volanú funkciu.
- Pamäťový priestor sa alokuje v zásobníku.

- Predpokladajme, že vo funkcii main je nasledujúci kód:

```
int a = sucet(4, 6);
```



Parametre funkcie

- V jazyku C sa skutočné parametre (argumenty) odovzdávajú funkcii **len hodnotou**.
- Dôsledok – funkcia nepracuje s originálnymi argumentmi, ale len s ich kópiami.
- Ak chceme pracovať s hodnotami originálnych argumentov, do funkcie musíme poslať **adresu premennej**, t.j. ukazovateľ.
- Ako to funguje?

Parametre funkcie – príklad 1

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

void modifikuj(int n) {
    printf("modifikuj(): &n = %p\n", &n);
    n = 10;
    printf("modifikuj(): n = %d\n", n);
}

int main(int argc, char* argv[]) {
    int n = 20;

    printf("main(): &n = %p\n", &n);
    printf("main(): n = %d\n", n);

    modifikuj(n);

    printf("main(): &n = %p\n", &n);
    printf("main(): n = %d\n", n);
}
```

```
main(): &n = 0x28cc2c
main(): n = 20
modifikuj(): &n = 0x28cc10
modifikuj(): n = 10
main(): &n = 0x28cc2c
main(): n = 20
Press [Enter] to close the terminal ...
```

Parametre funkcie – príklad 2

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

void modifikuj2(int* n) {
    printf("modifikuj2(): &n = %p\n", n);
    *n = 10;
    printf("modifikuj2(): n = %d\n", *n);
}

int main(int argc, char* argv[]) {
    int n = 20;

    printf("main(): &n = %p\n", &n);
    printf("main(): n = %d\n", n);

    modifikuj2(&n);

    printf("main(): &n = %p\n", &n);
    printf("main(): n = %d\n", n);
}
```

```
main(): &n = 0x28cc2c
main(): n = 20
modifikuj2(): &n = 0x28cc2c
modifikuj2(): n = 10
main(): &n = 0x28cc2c
main(): n = 10
Press [Enter] to close the terminal ...
```

Pole (1)

- Zoznam objektov rovnakého typu, ktorého veľkosť je nemenná. K prvkom poľa sa pristupuje prostredníctvom indexu, ktorý začína od 0.
- Pre pole sa alokuje **súvislý úsek pamäte**, prvý prvok je umiestnený na najnižšej adrese, posledný prvok na najvyššej adrese.
- Jazyk C pozná 3 typy polí:
  - **polia konštantnej známej veľkosti** (veľkosť je známa pri preklade) – dátový segment, zásobník
  - **polia voliteľnej veľkosti** (VLA, C99) (veľkosť je známa až za behu) – zásobník
  - dynamicky alokované polia (veľkosť je známa až za behu) – halda

Pole (2)

- Definícia:

  typ_prvku identifikator [pocet_prvkov];

- Deklarácia:

  extern typ_prvku identifikator [pocet_prvkov];

  extern typ_prvku identifikator [];

- Objekty typu pole sú **nemodifikovateľné l-hodnoty**:

  int pole1[10];

  int pole2[10];

  ~~pole1 = pole2;~~

Polia voliteľnej veľkosti (VLA, C99)

- Povinné v C99, voliteľné v C11.
- Ak kompilátor definuje makro __STDC_NO_VLA__ s hodnotou 1, VLA nie sú podporované (C11).
- Môžu vystupovať ako lokálne premenné (bez modifikátora static) alebo ako formálne parametre funkcií.
- Nie je možné len deklarovať premennú typu VLA (prečo?):

~~extern int pole[n];~~

Pole – príklady definícií

- Definícia:

```
int pole[30];
int pole[n];
int pole[0]; //gcc
```

- Definícia a inicializácia:

```
int pole[] = {1,2,3,4,5,6};
int pole[20] = {}; //gcc
int pole[10] = {1,2,3};
int pole[10] = {1,2,3,[9] = 10, [7] = 7}; //C99
int pole[n] = {1,2,3};
int pole[n] = {};
```

- Koľko bajtov sa alokuje v predchádzajúcich definíciách?

Konverzia pola na smerník

- Každá l-hodnota typu pole sa implicitne konvertuje na r-hodnotu typu „smerník na prvok poľa,“ ktorý ukazuje na prvý prvok v poli.
- Výnimkou sú prípady, ak l-hodnota typu pole vystupuje ako:
  - operand operátora &
  - operand operátora sizeof
  - ako reťazcový literál použitý pre inicializáciu poľa
- Dôsledky – ak uvažujeme nasledujúcu definíciu:
  - int pole[10];

  tak potom platí:
  - *pole == pole[0];
  - pole == &pole[0];

Pole ako parameter funkcie

- V dôsledku predchádzajúcej konverzie sa do funkcie, ktorá očakáva parameter typu pole odovzdáva nie pole, ale smerník na prvý prvok poľa.
- Možné deklarácie funkcie s parametrom typu pole:
  - int funkcia(int n, int pole[n]); //C99
  - int funkcia(int n, int pole[*]); //C99
  - int funkcia(int n, int pole[]);
  - int funkcia(int n, int pole[10]);
  - int funkcia(int n, int pole[static 10]); //C99

- Ukážka:

```
void vypisPole(int n, int pole[n]) {
    for (int i = 0; i < n; i++) {
        printf("%d ", pole[i]);
    }
    printf("\n");
}
```

Reťazce

- V jazyku C Neexistuje špeciálny údajový typ, je to pole znakov, ukončené znakom '\0'.
- Reťazce majú neobmedzenú dĺžku.
- Pri zápise reťazcových literálov prekladač pridáva ukončovací znak automaticky.
- Operácie s reťazcami:
  - <stdio.h> – načítavanie, zapisovanie reťazcov, prevod iného dátového typu na reťazec
  - <stdlib.h> – prevod reťazca na iný dátový typ (číslo)
  - <string.h> – manipulácia s reťazcami (kopírovanie, spájanie, hľadanie)

Reťazce – príklady definícií

- Ako pole znakov:

```
char ret[30];
char ret[20] = {'A', 'h', 'o', 'j'};
...
```

- Definícia a inicializácia pomocou literálu:

```
char ret[10] = "Ahoj";
char ret[] = "Ahoj";
char ret[4] = "Ahoj";
char ret[5]; ~~ret = "Ahoj";~~ //len cez strncpy()
```