KATEDRA
INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI

Jazyk C: opakování

KMI/OS1 Operační systémy I

Mgr. Markéta Trnečková, Ph.D.
www.marketa-trneckova.cz

Organizační informace

- **Vyučující**: Mgr. Petr Krajča, Ph.D.
- **Primární zdroj informací**: `https://phoenix.inf.upol.cz/~krajcap/courses/2025LS/OS1/`
- **Cvičící**: Mgr. Markéta Trnečková, Ph.D.
- **Web**: `https://www.marketa-trneckova.cz/`
- **Teams**: KMI/OS1
- **E-mail**: `marketa.trneckova@gmail.com`
- Organizační informace dále jsou převzaty od vyučujícího

Organizační informace

Náplň cvičení

- praktické seznámení s:
  - činností procesoru
  - překladu programů
  - základního rozhraní operačních systémů
- práce s procesy a vlákny v operačním systému Windows a Linux
- assembler – umožňuje vytvářet programy na úrovni jednotlivých instrukcí procesoru
- volání služeb OS

Pro úspěšné zvládnutí předmětu je nutná předchozí znalost jazyka C a základní uživatelské práce v operačních systémech Windows i Linux

Organizační informace

Materiály ke cvičení

- Učební text vyučujícího!
- Není potřeba pracovat s dalšími zdroji.
- Úkoly ke cvičením nejsou bodovány a slouží k tomu, abyste si mohli vyzkoušet, že probírané látce rozumíte!

- **Nepoužívejte nástroje postavené na velkých jazykových modelech (LLM)**
- Smyslem cvičení je seznámit se s činností procesoru a OS, nikoliv vyřešit zadané úkoly za každou cenu. Pokud úkol za vás vyřeší někdo (nebo něco) jiný, nebude to pro vás mít žádný přínos.
- Programy, které budeme na cvičeních vytvářet, budou velice úzce spojeny s konkrétním typem procesoru a operačního systému. LLM jsou z podstaty především generátory textu a nepracují se znalostmi o procesorech a operačních systémech. Proto při generování kódu v assembleru velice často generují chybný kód, „pletou si" instrukce, nechápou pravidla, která je na jednotlivých OS nutné dodržovat apod.
- To samé se týká služeb typu StackOverflow. Zde je potřeba sledovat, jestli odpověď odpovídá přesně vašim potřebám, například, jestli není odpověď pro OS Windows, přičemž vy hledáte řešení pro Linux.

Organizační informace

Hardware a software

- Cvičení budou podle potřeby probíhat v operačním systému Linux a Windows.
- Programování v assembleru bude demonstrováno na procesorech s instrukční sadou AMD64, což zahrnuje i 64bitové procesory Intel. Z tohoto důvodu nepůjde řešit některé úkoly na počítačích s jiným typem procesoru, nejtypičtěji se to týká počítačů Apple s procesory M1 a výš.
- Na učebnách jsou nachystány počítače s oběma operačními systémy.
- K dispozici je dále Linuxový server os.inf.upol.cz, kam se dá přihlásit přes SSH. Přihlašovací údaje jsou ve tvaru login@PRFAD a heslo je stejné jako do domény.
- Je dobré mít oba OS na počítači, např. pomocí virtualizace.

Organizační informace

Podmínky pro udělení zápočtu

- V průběhu semestru budou dvě zápočtové písemky hodnocené 10 body a jedna opravná. (Počítají se vždy dva nejlepší výsledky). Tyto písemky budou vycházet z učiva probíraného na cvičeních.
- Za aktivní účast na cvičení je možné získat 0,5 bodu.
- K udělení zápočtu je nutné získat minimálně 70% bodů.
- K dispozici je dále Linuxový server os.inf.upol.cz, kam se dá přihlásit přes SSH. Přihlašovací údaje jsou ve tvaru login@PRFAD a heslo je stejné jako do domény.
- Pokud se studentovi či studentce nepodaří získat zápočet pomocí bodů z písemek a cvičení, může mu nebo jí být přidělen bonusový úkol hodnocený až 5 body. Zadání bonusového úkolu je podmíněno získáním alespoň 55% bodů za cvičení a alespoň 50% z písemek.

Aktivní účastí na cvičení se myslí přítomnost v učebně a řešení zadaných úkolů.
Účast na cvičeních není povinná, ale je silně doporučovaná.

Jazyk C: opakování

Učební text ke cvičení:
`https://phoenix.inf.upol.cz/~krajcap/courses/2025LS/OS1/tutorial01.pdf`

Pro více detailů o jazyce C (skripta Jazyk C):
`https://www.dropbox.com/s/9txvyf3ux7cphnm/skripta_jc.pdf?dl=0`

Jazyk C: opakování

Příklad

Zařaďte jazyk C mezi ostatní programovací jazyky.

Základní datové typy

- **celočíselné**: `int`, `short int` (zkráceně `short`) a `long int` (zkráceně `long`)
- `sizeof(short int)` $\leq$ `sizeof(int)` $\leq$ `sizeof(long int)`
- velikost dána počítačem (nejběžnější překladače): `short` 16 bitů, `int` 32 bitů, `long long` 64 bitů, `long` 32/64 bitů (Win/Linux)
- nový standard: celočíselné typy s přesně daným počtem bitů.
- **znaky**: `char`
- velikost `char`: 1 byte
- **desetinné**: `float` (jednoduchá přesnost), `double` (dvojitá přesnost) a `long double` (čtyřnásobná přesnost)
- `sizeof (float)` $\leq$ `sizeof(double)` $\leq$ `sizeof(long double)`
- Běžná velikost: 32, 64 a 80 bitů
- celočíselné datové typy mohou být `signed` (znaménkové) nebo `unsigned` (neznaménkové)
- `sizeof(unsigned int)` $=$ `sizeof(signed int)`

Pojmenování typů

```
typedef deklarace nazev;
```

Literály

- **celočíselné**:
  - desítková soustava: `1`, `42`, `123`
  - osmičková soustava: `01`, `052`, `0173`
  - šestnáctková soustava: `0x1`, `0x2a`, `0X7B`
- implicitně `int`
- přípony: L (l) – `long`, U (u) – `unsigned` 123LU
- záporné hodnoty –
- **reálné**:
  - s desetinnou tečkou: `.123`, `123.`, `1.23`
  - exponenciální tvar: `1e23`, `12E3`
- implicitně `double`
- přípony: F (f) – `float`, L (l) – `long double`
- **znaky**:
  - apostrofy: `'a'`, `'*'`, `'2'`
  - escape sekvence: `'\n'`, `'\"'`, `'\0'`
  - pomocí čísel: `'\ddd'`, `'\xhh'`
- **řetězce**:
  - v uvozovkách: `"Ja jsem \" retezcova \" hodnota \n"`

Struktury

```
struct jmeno {
polozka1
polozka2
...
polozkan
} seznam promennych;
```

- `struct jmeno foo = {h1, ... hn};`
- `foo.polozka1`

Operátory

- Operace, vždy vrací hodnotu
- Arita operátoru
- Priorita operátoru
- Asociativita

Operátory

Priority operátorů

priorita	operátor	význam	asociativita
1	`()`	volání funkce	zleva doprava
	`[]`	index do pole	
	`->`	přístup k prvku struktury	
	`.`	přístup k prvku struktury	
2	`!`	negace logického výrazu	zprava doleva
	`~`	jedničkový doplněk	
	`++`	inkrementace	
	`--`	dekrementace	
	`+`	unární plus (+1)	
	`-`	unární mínus (−1)	
	`(typ)`	přetypování	
	`*`	dereference	
	`&`	reference (adresový operátor)	
	`sizeof`	velikost typu	
3	`*`	součin	zleva doprava
	`/`	dělení	
	`%`	dělení modulo	
4	`+`	součet	zleva doprava
	`-`	rozdíl	
5	`<<`	bitový posun doleva	zleva doprava
	`>>`	bitový posun doprava	

Operátory

Priority operátorů

priorita	operátor	význam	asociativita

6	`<`
	`<=`
	`>`
	`>=`
		menší než
		menší nebo rovno než
		větší než
		větší nebo rovno než
			zleva doprava

7	`==` `!=`	rovnost nerovnost	zleva doprava
8	`&`	bitový součin	zleva doprava
9	`^`	exkluzivní bitový součin	zleva doprava
10	`|`	bitový součet	zleva doprava
11	`&&`	logický součin	zleva doprava
12	`||`	logický součet	zleva doprava
13	`? :`	ternární (podmínkový) operátor	zprava doleva

14	`=`
	`>>=`
	`<<=`
	`+= -=`
	`*= /=`
	`%= &=`
	`|= ^=`
		přiřazovací operátory	zprava doleva

15	`,`	operátor čárky	zleva doprava

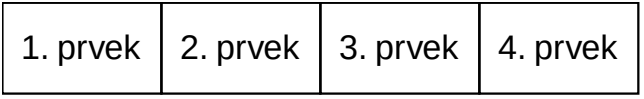
Ukazatele

- **paměť** = paměťové buňky 1 byte
- **adresa paměti** = ukazatel (pointer)
- typ pointeru
- `typ *jmeno = init;`
- **operátory**: `*`, `&`
- **operátor adresy &**
- **operátor dereference ***
- **nulový ukazatel** = NULL

Příklad

Proč potřebujeme definovat u ukazatele, na jaký datový typ ukazuje?

Statická pole



- **index**: 0, 1, ...
- `typ jmeno[velikost];`
- `typ jmeno[velikost]={p1, p2, ..., pn};`
- `jmeno[index];`

Příklad

Jaký je vztah mezi poli a ukazateli?

Řetězce (pole char)

- `char retezec[11]= {'A','h','o','j',' ','s','v','e','t','e','\0'};`

'A'	'h'	'o'	'j'	' '	's'	'v'	'e'	't'	'e'	'\0'

- `char retezec[]= "Ahoj svete";`

Příklad

Jsou tyto dva příkazy shodné?

```
char retezec[]= "Ahoj svete";
char* retezec= "Ahoj svete";
```

Dynamická alokace

- `#include <stdlib.h>`
- **alokace:** `void *malloc(size_t velikost);`
- **alokace:** `void *calloc(size_t polozky, size_t velikost);`
- **přetypování:**
  - implicitní: přiřazení
  - explicitní: `(novy_typ) promenna;`
- **změna velikosti alokované paměti:** `void *realloc(void *adresa, size_t velikost);`
- **uvolnění paměti:** `void free(void *adresa);`

Pointerová aritmetika

- K adrese lze přičíst nebo od ní odečíst nezáporné celé číslo
- Lze spočítat rozdíl adres stejného typu
- Adresy lze porovnávat stejně jako čísla

Příklad

Jaké je praktické využití jednotlivých operací?

Přístup k prvkům pole

Příklad

1. Mějme pole: `int pole[]= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };`
2. Jak zjistíte adresu 4. prvku pole?
3. Máme ukazatel, který ukazuje na některý prvek pole pole. Jak zjistíte index tohoto prvku v poli?
4. Máme 2 ukazatele do stejného pole. Jak zjistíme, který z ukazatelů ukazuje na prvek, který je v poli dříve?

Funkce

- **Hlavička funkce**: `typ jmeno(typ1 a1, typ2 a2, ..., typn an)`
- **Tělo funkce**
- `return hodnota;`
- **Deklarace**: `double mocnina(double, int);`
- `typ y = jmeno(a1, a2, ..., an);`

Funkce

- **Předávání argumentů**:
  - hodnotou
  - odkazem

Příklad

Jakým způsobem jsou předávány struktury?

Větvení programu

Příklad

Existuje v jazyce C logický datový typ?

Podmínky: $<, <=, >, >=, ==, !=$
Spojování podmínek: $||, \&\&$ (líné vyhodnocování)

- `if-else`
- `switch`
- `jednoduché podmínky:` `podminka ? vyraz1 : vyraz2;`

Cykly

- `while`
- `for`
- `do while`

Bitové operace

- **bitový součin** = AND
- **bitový součet** = OR
- **bitový exkluzivní součet** = XOR
- **posun doleva**
- **posun doprava**
- **jedničkový doplněk** = NOT

Bitový součin

- `x & y`

Příklad

Výsledkem operace `100 & 50` bude 32 protože:

100	0 1 1 0 0 1 0 0
50	0 0 1 1 0 0 1 0
100 & 50	0 0 1 0 0 0 0 0

Liché číslo

Příklad

```
#define liche(x) (1 & (x))
```

Bitový součet

- `x | y`

Převod velkého písmena na malé

Příklad

```
#define na_mala(c) (c | 0x20)
```

0x20 odpovídá binárně 0 0 1 0 0 0 0 0

Bitový exkluzivní součet

- `x ^ y`

Porovnání dvou čísel

Příklad

```
if( x ^ y)
    /* cisla jsou rozdilna */
```

Bitový posun doleva

- `x « n`

Násobení mocninou 2.

Příklad

`x = y « 1` vynásobí číslo 2

```
 1       =  0 0 0 0 0 0 0 1
 1 « 1   =  0 0 0 0 0 0 1 0
```

Bitový posun doprava

- `x » y`

Celočíselné dělení mocninou 2.

Příklad

`x = y » 3` vydělí číslo 8

Jedničkový doplněk = negace bit po bitu

- $\tilde{x}$

Doplněk: závisí na typu čísla

Příklad

```
int main()
{
    int i = 10, i2;
    unsigned char j = 10, j2;

    i2 = ~i;
    j2 = ~j;

    printf("%i %i \n", sizeof(char), sizeof(int));
    printf("%u %i", i2, j2);
}
```

Jazyk C

Úkoly k procvičení

1. Napište funkci `void int2bits(char *, int)`, která převede číslo na textový řetězec představující jeho zápis v binární podobě.
2. Napište funkci `int bits2int(char *)`, která převede textový řetězec představující zápis čísla v binární podobě (tj. `010110010010...`) na hodnotu typu int.
3. Implementujte funkci `void my_memcpy(void *dest, void *src, size_t size)`, která se chová jako funkce `memcpy` a přenese po jednotlivých bytech obsah paměti z jednoho místa na druhé, předpokládejte, že úseky paměti se nepřekrývají.
4. Navrhněte vhodnou strukturu pro spojový seznam obsahující dvě hodnoty jméno (textový řetězec) a věk (celé číslo). Napište funkci, která bude přidávat prvky do seznamu a funkci, která vypíše obsah tohoto seznamu.
5. Napište funkci `short encode_date(char day, char month, short year)`, která zakóduje datum do 16bitového čísla následovně: YYYY-YYYM-MMMD-DDDD.
6. Napište funkci `void decode_date(short date, int *day, int *month, int *year)`, která dekóduje datum vytvořené předchozí funkcí a vrátí hodnoty pomocí předaných ukazatelů.
7. Napište funkci, která zjistí, v jakém pořadí jsou vyhodnocovány argumenty.