Jazyk C - základy

1 Paměť

- Uchovává programy
  - kód
  - proměnné
  - konstanty
- RAM je přepisovatelná
  - proměnné lze číst i zapisovat!
    - proto jsou proměnné...
- Na nejnižší úrovni
  - Paměť je **pole** paměťových buněk.
  - **Indexy** do tohoto pole jsou **adresy**, tj. **ukazatele**.
- Model paměti
  - Zásobník
    - Lokální proměnné
    - Alokuje se dynamicky za běhu programu.
    - Alokuje se automaticky při volání funkcí.
      - Aktivační záznam: kopie argumentů funkce + lokální proměnné.
    - Uvolní se automaticky při ukončení funkce.
      - Lokální proměnné na konci funkce zanikají.
      - Vracet z funkce ukazatel na lokální proměnnou nebo lokální pole je blbost...
    - Roste od konce paměti.
  - Halda
    - Dynamické proměnné
    - Přiděluje operační systém.
      - malloc v C
      - new v C++
    - Uživatel je povinen uvolnit.
      - free v C
      - delete v C++
    - OS v ní ukrývá celý zbytek RAM.
      - Ostatní procesy
    - Dynamicky alokovaná paměť.
    - Alokuje se až za běhu programu.
  - Datová oblast
    - Globální proměnné a konstanty
      - Součást přeložené binárky (.exe).
      - Při startu programu se toto zkopíruje do RAM ze souboru (.exe).
      - Důsledek: velké globální proměnné a konstanty zvětšují velikost přeloženého binárního souboru (.exe).
  - Kódová oblast
    - Kód programu
- Abstrakce paměti
  - Abstrakci modelu paměti vytváří OS.
  - Programu se zdá, že má k dispozici celou paměť.
  - OS pomocí ní může provozovat paralelně mnoho procesů najednou.

2 Proměnná a typ

- Identifikátor
  - Pojmenování proměnné, konstanty, typu nebo podprogramu.
  - Nesmí začínat číslicí. Může začínat pouze znaky a-zA-Z_ . Dále mohou následovat znaky a-zA-Z_0-9.
    - To platí pro jazyk C. Některé jazyky připouští i diakritiku.
    - ❗ Nesmí obsahovat mezery ani znaky, které jsou použity jako operátory (+,-,*,/, ...).
  - Musí být jednoznačný.
    - ❗ Jazyk C je case sensitive, tzn. identifikátory `cislo` a `Cislo` jsou dva různé identifikátory.
  - Nejde jej používat před deklarací!
    - Obecné pravidlo v jazyce C. Platí pro proměnné, typy i podprogramy.
    - ❗ Způsobuje varování nebo chybu při překladu.
- Proměnná
  - Obecně
    - pojmenovaná paměťová buňka (nebo kus paměti)
      - Je tedy přístupná pomocí přidělného identifikátoru.
      - Některé proměnné jsou přístupné pomocí zabudovaných operátorů, např. prvek pole pomocí operátoru [], složka struktury pomocí operátoru . nebo ->.
    - jde ji přečíst a jde do ní zapsat -- proto je proměnná!
  - V jazyce C
    - pojmenovaná paměťová buňka, která má přiřazen datový typ
    - Překladač hlídá programátora, aby se nepokusil zapsat hodnotu jiného datového typu. Říká se tomu **typová kontrola**.
      - Některé typy jsou částečně kompatibilní (char -> int -> float). U nich zápis povolí.
      - Jiné kompatibilní nejsou (int -> struktura). U nich zápis nepovolí a generuje chybové hlášení.
    - Toto dělají i jiné jazyky se **silnou typovou kontrolou**. Např. Pascal, C++, Java, C#. Typicky jde o kompilované programy.
      - Výhodou jsou kontroly během kompilace - zavčas odhalená chyba je lepší, než skrytá.
      - Výhodou je rychlejší běh výsledného programu.
  - Některé jiné jazyky
    - Jazyky se **slabou typovou kontrolou** (JavaScript, Python, Scheme/Racket, Scratch) toto nedělají. Proměnná je zde jakýsi kontejner, kam jde vložit cokoli -- číslo, za chvíli text, pak třeba obrázek. K chybě dojde až za běhu, když se programátor například pokusí sečíst číslo a obrázek...
    - Problém je, že kontrola se nedělá při překladu, ale až za běhu programu -- tyto jazyky jsou obvykle skriptovací (interpretované, ne kompilované). Chyby se pak hledají obtížněji.
- Datový typ
  - Je dán
    - název
      - Zabudované typy mají názvy jako klíčová slova: char, int, long, float, double aj.
      - Uživatel může tvořit vlastní typy (v jazyce C pomocí **typedef**).
        - `typedef popis_typu nazev_noveho_typu;`
    - množinou hodnot, kterých mohou nabývat proměnné
      - Ta je dána především velikostí paměti, kterou daný typ požaduje. Např. typ char zabírá jeden bajt, proto můžou proměnné typu char nabývat pouze 256 různých hodnot. Typ int zabírá 4 nebo 8 bajtů, proto je počet různých hodnot 2^32 = 4294967296 nebo 2^64 = 18446744073709551616.
    - množinou operací, které jde dělat s hodnotami tohoto typu
      - Zabudované typy mají zabudované operace jako +, -, *, /, <<, >> atd.
      - U uživatelských typů vytváří přípustné operace programátor pomocí podprogramů.
  - Zabudované typy v C
    - Jsou už připraveny k použití. Mají definovány operátory pro práci s jejich hodnotami. Typicky jde o jednoduché číselné typy.
    - Celé číslo
      - char, short, int, long, long long
      - unsigned
        - předpona pro neznaménkové
    - Desetinná čísla
      - float, double, long double
    - Znak
      - char
  - Uživatelské typy
    - Musí je blíže specifikovat uživatel. Jazyk poskytuje základní operátory pro přístup k uloženým hodnotám, ale složitější operace musí programátor naprogramovat sám. Typicky jde o složené typy, které mají hodnotu složenou z hodnot jiných typů.
    - Pole
      - bázový_typ identifikátor[rozměr];
    - Struktura
      - struct identifikátor {složky};
    - Ukazatel
      - bázový_typ *
- Deklarace proměnné
  - Syntax: `typ identifikátor`
    - Typem proměnné může být jakýkoli typ, kromě prázdného typu void.
  - Je to prohlášení o budoucí existenci proměnné.
    - Je to syntaktická záležitost: Říkáme překladači: "Připrav se, že budu pracovat s proměnnou tohoto jména a tohoto typu."
    - Je to "závazek", že někdy takovou proměnnou skutečně vyrobím a dosadím na toto místo.
  - Typicky používáme deklarace parametrů podprogramů.
    - V hlavičce jsou deklarovány parametry, se kterými se pracuje v těle funkce, ale v paměti se pro ně vyhradí místo až při zavolání funkce, kdy se sem nakopírují hodnoty použitých argumentů.
  - Deklarace lokální proměnné je zároveň definicí.
- Definice proměnné
  - Syntax: `typ identifikátor`
    - Typem proměnné může být jakýkoli typ, kromě prázdného typu void.
  - Je to **příkaz** k vyhrazení paměti pro proměnnou.
    - Důsledkem je skutečné vyhrazení místa v paměti, které velikostí odpovídá požadavkům použitého typu.
    - Příkaz končí středníkem **;**
  - Lokální proměnná
    - Je definována uvnitř těla nějaké funkce.
    - Je viditelná pouze od místa své deklarace (a definice) do konce bloku, v němž byla vyrobena. Poté zaniká - je z paměti automaticky uvolněna.
    - Je uložena v paměti zvané **zásobník**. Vzniká dynamicky, tj. až za běhu programu při zavolání funkce.
  - Globální proměnná
    - Je definována mimo tělo jakékoli funkce.
    - Je viditelná od místa své deklarace, až do konce modulu (resp. zdrojového souboru).
    - Je viditelná a použitelná i uvnitř funkcí - POZOR - způsobuje vedlejší efekty, tj. chování, které není viditelné na první pohled.
    - Je uložena v paměti zvané **datová oblast programu**. Je tedy součástí přeloženého programu (soubor .exe).
    - Její použití musí být velmi dobře odůvodněné, jinak je v rozporu se zásadami "dobrého programátorského stylu" a hlavně způsobuje těžko odhalitelné chyby.
      - Např. programy pro Arduino jsou zcela specifickým příkladem, kde je použití globálních proměnných odůvodněné: je to HW s velmi omezenými zdroji a programy jsou zde velmi malé.
- Inicializace proměnné
  - První přidělení hodnoty po definici proměnné.
  - Syntax: `typ identifikátor = hodnota;`
    - Přiřazení lze provést hned při definici proměnné.
  - V C mají některé typy zaveden speciální operátor - **inicializátor**.
    - Například u pole jde přiřadit první hodnota takto jednoduše pouze při definici, později to nejde, protože pole nemá definován přiřazovací operátor.

3 Typ pole

- Význam
  - Složený typ.
  - **Homogenní datový typ**: proměnné (prvky) stejného typu ležící vedle sebe v paměti.
  - Proměnná typu pole zastřešuje všechny své prvky.
    - Výhodné u parametrů funkcí.
    - Jde s nimi zacházet společně přes jediný identifikátor.
    - S každým prvkem jde zacházet jako se samostatnou proměnnou.
  - (Vnitřně je proměnná typu pole ukazatel na první prvek.)
- Deklarace proměnné
  - `bázový_typ identifikátor[rozměr]`
  - Bázový typ
    - Typ každého prvku pole
    - Jakýkoli typ kromě void.
  - Rozměr
    - Celé, kladné, nenulové číslo.
    - Při deklaraci (např. jako parametr funkce) není u jednorozměrných polí povinný. Délka pole se musí stejně zadávat pomocí dalšího parametru.
- Definice proměnné
  - Vyhradí místo v paměti
    - o velikosti délka*sizeof(bázový_typ)
  - `float pole0[10];`
    - Pole bez inicializátoru musí mít uveden rozměr.
    - Neinicializované pole má v prvcích neinicializované hodnoty (tj. není zaručena ani nula ani náhodná hodnota - nejde to použít jako generátor náhodných čísel).
  - Rozměr může být dosazen i proměnnou.
    - Pole na zásobníku ale nemůže být příliš velké!
    - U reálných programů je lepší použít dynamickou alokaci na hromadě (heap) pomocí funkce malloc.
- Inicializátor pole
  - `float pole1[]={1.2, 3.5, [5]=-0.5};`
    - V deklaraci lze vynechat rozměr dimenze - pak se rozměr dopočítá z inicializátoru.
    - Hodnoty v inicializátoru se oddělují čárkami.
    - Jde inicializovat prvek uprostřed pole zadáním jeho indexu v hranatých závorkách.
  - `float pole2[10]={1.4};`
    - Pokud je rozměr pole uveden, budou neinicializované prvky nastaveny na nulovou hodnotu (tj. alespoň jeden prvek musí být inicializovaný).
  - ❌ ~~`pole0 = {1.4};`~~
    - Tohle nejde.
    - Na již vytvořené pole nelze použít přiřazovací operátor.
- Práce s polem
  - Přístup k prvkům: operátorem indexování **[]**
  - ❗ V jazyce C se pole indexuje vždy od nuly!
  - `pole1[0] = 5.7;`
    - Indexovat jde konstantou.
  - `for (int i = 1; i < 6; ++i) pole1[i] = pole1[i-1]*2;`
    - Indexovat jde i celočíselnou proměnnou nebo výrazem.
  - ❌ ~~`pole1 = pole0;`~~
    - Tohle nejde.
    - Pole nejde kopírovat přiřazením!
    - Musí se použít cyklus a kopírovat po prvcích.
    - Pole si v jazyce C nepamatuje svou délku, proto jej nejde takto kopírovat.
      - 🙂 Má to dobré důvody: sice to není pohodlné, ale C tak může být ďábelsky rychlé.
      - 🙂 Funkce nemusí vždy pracovat s celým polem. Můžou pracovat jen s částí pole. Dokonce jde do funkce předat ukazatel doprostřed pole, aniž by bylo třeba jej dělit a kopírovat.
  - Argument funkce
    - Pole je vnitřně ukazatel.
    - Nikdy ne s operátorem &.
    - Pole se při předání do funkce nekopíruje! Funkce pracuje s originálním polem. Pokud do něj něco zapíše, vždy se to projeví v původním poli.
- Vícerozměrná pole
  - Pole polí.
    - Pro reálnou práci jde uvažovat o dvou a třírozměrných polích. Více rozměrů je možných, ale je to nepraktické. Obvykle je to známkou nevhodného návrhu programu.
  - `float matice[][3]={ {1.2, 3.4, 5.6}, {0.3, 1.5, 2.8}, };`
    - V deklaraci (s inicializací nebo v parametru funkce) lze vynechat rozměr první dimenze, další jsou ale povinné.
    - U matic: 1. rozměr - počet řádků, 2. rozměr - počet sloupců.
  - `pole1[r][s] = 2.3;`
    - Indexuje se v pořadí: řádek, sloupec

4 Typ struktura

- Význam
  - Složený typ.
  - **Heterogenní datový typ**: proměnné (složky) libovolných typů ležící vedle sebe v paměti.
  - Proměnná typu struktura zastřešuje všechny své prvky.
    - Výhodné u parametrů funkcí.
    - Jde s nimi zacházet společně přes jediný identifikátor.
    - S každým prvkem jde zacházet jako se samostatnou proměnnou.
  - Typicky se zavádí pomocí specifikátoru **typedef**.
- Specifikace typu struktura
  - `typedef struct { char jmeno[21]; int vyska; } Tosoba;`
- Deklarace proměnné
  - Např. jako parametr funkce.
  - `TypStruktura identifikátor`
  - TypStruktura je typicky specifikován pomocí **typedef**.
- Definice proměnné
  - `Tosoba c;`
    - Vyrobí neinicializovanou proměnnou typu Tosoba.
- Inicializace struktury
  - `Tosoba a={"Adam", 152};`
    - Struktura se inicializuje po složkách.
    - Složky musí být uvedeny ve stejném pořadí jako při specifikaci typu.
    - Hodnoty v inicializátoru se oddělují čárkami.
  - `Tosoba b={.vyska=175, .jmeno="Eva"};`
    - Preferovaný způsob v nových verzích jazyka C.
    - Složky jde inicializovat pomocí tečkové notace. Pak není nutné dodržovat stejné pořadí jako při specifikaci typu.
    - Neinicializované složky struktury se automaticky nastaví na nulovou hodnotu, pokud byla inicializována alespoň jedna složka.
  - `Tosoba c=a;`
    - Jde kopírovat hodnotu celé struktury pomocí přiřazení.
  - `c.vyska = 170;`
    - Hodnoty jde přiřazovat také po složkách
  - `strcpy(c.jmeno, "Alena");`
    - ❗ Pozor! Pole nejde přiřazovat - řetězec pomocí funkce strcmp.
  - ❌ ~~`c = {"Adam", 152};`~~
    - Mimo definici už inicializátor nejde použít.
- Práce se strukturou
  - Přístup ke složkám proměnné: tečkovým operátorem **.**
  - `c.vyska = 163; printf("%s meri %d cm.", c.jmeno, c.vyska);`
    - Složky proměnné c jsou také proměnné.
  - ❌ `Tosoba.vyska = 180;`
    - ❗ Nesmysl! Tosoba je typ, ne proměnná.
    - Častá začátečnická chyba, když někdo nechápe rozdíl mezi typem a proměnnou.
- Ukazatel na strukturu
  - `Tosoba *d;`
    - Vyrobí neinicializovanou proměnnou typu ukazatel na proměnnou typu Tosoba.
    - Nevyrobí proměnnou typu struktura!
  - `d = &a;`
    - Nastaví do d adresu proměnné a, což je proměnná typu Tosoba.
  - `d->vyska = 167; printf("%s meri %d cm.", d->jmeno, d->vyska);`
    - Přístup ke složkám struktury (proměnné) odkazované přes ukazatel: pomocí operátoru ->.
  - `(*d).vyska = 167; printf("%s meri %d cm.", (*d).jmeno, (*d).vyska);`
    - Operátor šipka -> je zkratkou za tohle.
  - `Tosoba *e = malloc(sizeof(Tosoba));`
    - Vyrobí proměnnou typu ukazatel, která je inicializována adresou proměnné dynamicky alokované na hromadě (heap).

5 Typ ukazatel

- Význam
  - Proměnná typu ukazatel obsahuje adresu paměťové buňky.
  - S odkazovanou proměnnou jde pracovat nepřímo přes její adresu.
  - Má význam u dynamicky alokovaných proměnných - s proměnnými na hromadě jde pracovat jen přes ukazatel.
  - Používá se pro parametry funkcí předávané odkazem.
  - Pro tvorbu pokročilých datových typů: seznam, strom, graf, ...
- Deklarace proměnné
  - `bázový_typ *identifikátor`
    - ❗ Zde je * součástí jména typu. Nejde o operátor *!
  - Bázový typ
    - Typ odkazované proměnné
    - Jakýkoli typ včetně void.
      - Proměnná typu void* se musí před použitím přetypovat na typ ukazatel na konkrétní typ.
- Definice proměnné
  - `float *pcislo;`
    - Ukončeno středníkem.
    - Vyrobí neinicializovanou proměnnou typu ukazatel.
    - Takto obsahuje náhodnou adresu, proto ji ještě nejde použít! Nejprve se do ní musí vložit smysluplná adresa na známou proměnnou, nebo hodnota NULL.
    - Celý typ je float*.
- Inicializace proměnné
  - `float *pcislo = NULL;`
    - Nastavení na prázdný ukazatel (konstanta NULL);
    - Hodnotu NULL je možné testovat. Náhodnou adresu od smysluplné nijak nepoznáme.
  - `float x; float *px = &x;`
    - Operátor & získá adresu proměnné x. Ta se nastaví do proměnné px.
- Práce s ukazatelem
  - Operátor &
    - Referenční operátor
    - Vrací adresu proměnné.
      - Může to být i prvek pole.
      - Může to být i složka struktury.
  - Operátor *
    - Dereferenční operátor
    - Má jako parametr ukazatel (adresu) a vrací odkazovanou hodnotu, tj. hodnotu proměnné, na kterou adresa ukazuje.
    - Výsledný výraz je proměnnou - může být použit i na levé straně přiřazovacího operátoru.
  - `float x; float *px; px = &x; *px = 15;`
    - Při deklaraci+definici proměnné px to není operátor, ale jméno typu. Operátory jsou vyznačeny modře.
    - Do proměnné px se uloží adresa proměnné x.
    - Do proměnné *px se uloží hodnota 15. Prakticky se tak hodnota 15 ocitne v proměnné x.

6 Podprogram

- Pojem podprogram
  - Uživatelsky vytvořený příkaz nebo funkce.
  - Relativně nezávislý kus programu, který s okolím komunikuje přes definované rozhraní (hlavičku).
  - Rozlišujeme podprogramy typu **funkce** a **procedura**.
  - V jazyce C jsou všechny podprogramy formálně funkcemi.
- Funkce
  - Podprogram, který vrací funkční hodnotu (návratovou hodnotu).
    - v kódu funkce: `return hodnota;`
  - Lze ji používat ve výrazech, nejčastěji při přiřazení.
- Procedura
  - Podprogram, který má charakter příkazu.
  - Nelze ji použít ve výrazech.
  - V jazyce C jde o funkci, která má návratový typ **void**.
    - může končit pomocí `return;`
- Vytváření
  - Hlavička = prototyp
    - Syntax: `návratový_typ identifikátor(parametry)`
    - Popisuje rozhraní podprogamu.
    - Za hlavičkou je středník **;**
      - Deklarace funkce
      - Po deklaraci už jde funkci používat.
      - Implementace funkce musí být někde dále v souboru nebo modulu.
    - Za hlavičkou je **{ tělo }**
      - Implementace funkce.
      - V těle je zapsán algoritmus funkce, který pracuje s hodnotami parametrů a případně vrací výsledek.
  - Návratový typ
    - ❗ Povinný. Musí být uveden na začátku hlavičky.
    - Říká, jakého typu bude výsledek funkce.
    - Jde použít jakýkoli typ, jehož hodnota jde přiřadit operátorem =.
      - Proto nejde vracet hodnotu typu pole! (Pouze ukazatel na dynamicky alokované pole. To je ale typ ukazatel...)
    - Procedury mají návratový typ **void**.
  - Deklarace parametrů
    - Seznam deklarací, oddělených čárkami.
    - Seznam může být i prázdný.
      - Ve starších verzích C to znamenalo funkci s *libovolným* počtem parametrů. Proto v moderních verzích C používáme místo prázdného seznamu klíčové slovo **void**.
    - Syntax: `typ identifikátor`
      - Typem proměnné může být jakýkoli typ, kromě prázdného typu void.
    - Někdy mluvíme o **formálních parametrech**.
    - Identifikátor parametru je viditelný jen uvnitř funkce. Při volání na něm vůbec nezáleží.
  - Definice funkce
    - Znamená implementaci funkce
    - Hlavička + { tělo }
    - Obsahuje kód popisující, co má dělat.
    - Chování funkce závisí na hodnotách parametrů získaných při volání, tj. argumentech.
      - Parametry, které nemají na chování funkce vliv jsou zbytečné.
      - Častá začátečnická chyba: deklarovat nesmyslné parametry místo lokálních proměnných.
    - Ukončení
      - Příkaz `return hodnota;`
        - U funkcí povinný.
        - Vrací výsledek.
        - Může jich být více a na různých místech.
      - Příkaz `return;`
        - Nepovinný u procedur.
        - Ukončí proceduru.
        - Může jich být více a na různých místech.
        - Procedury bez příkazu return končí na uzavírací závorce těla.
  - Rozhraní funkce
    - Je dáno hlavičkou (prototypem).
    - Návratová hodnota slouží pro přenos dat (výsledku) směrem **ven z funkce**.
    - Parametry slouží primárně pro přenos dat směrem **do funkce**. Jde o parametry **předávané hodnotou**.
      - Funkce pracuje s kopií hodnoty použité při volání.
    - Parametry **předávané odkazem** slouží pro přenos dat směrem **ven z funkce**.
      - V jazyce C musí jít doslova o odkaz, tedy typ ukazatel.
      - Při volání se musí předat odkaz na proměnnou (tedy &proměnná).
      - Uvnitř funkce se pak s parametrem pracuje přes operátor *. V důsledku pracujeme s obsahem originální, odkazované proměnné skrze její adresu.
- Volání
  - Syntax: `identifikátor(argumenty)`
    - Samotný identifikátor funkce je *ukazatel na funkci*.
    - Kulaté závorky jsou operátor volání funkce.
    - Seznam argumentů může být prázdný.
    - Více argumentů se odděluje čárkami.
    - Volá se stejně, jako knihovní funkce.
  - Procedura
    - Syntax: `identifikátor(argumenty);`
    - ❗ NE kopie hlavičky!
    - Součástí volání je ukončovací středník.
    - Je to příkaz. Nemá funkční (návratovou) hodnotu. Nejde použít ve výrazu.
  - Funkce
    - Syntax: proměnná = `identifikátor(argumenty);`
    - ❗ NE kopie hlavičky!
    - Funkce jde použít ve výrazech.
    - Středník ukončuje přiřazovací výraz.
    - `double y = sin(x) * cos(ln(x));`
    - Další příklad použití funkce. `if (sin(x) > 0) { return 1; }`
  - Argumenty
    - Někdy také nazývané **skutečné parametry**.
    - Hodnoty dosazované při volání.
    - Pořadí odpovídá pořadí deklarovaných parametrů.
    - Za předávané hodnotou lze dosadit
      - proměnnou
      - konstantu
        - printf("Ahoj");
      - výraz
        - sin(a+b)
    - Za předávané odkazem lze dosadit
      - Ukazatel na proměnnou
        - `float x; scanf("%f", &x);`
        - Operátor & vrací adresu proměnné.
        - `void nacti(float *x) { scanf("%f", x); }`
        - x už je adresa