Kompilace

Jaké znáte překladače jazyka C?

- GCC - GNU compiler collection, nejpoužívanější, open-source, pro Windows v balíčku MinGW
- Clang
- MSVC - Microsoft Visual C, součást VS, Windows

Popište proces vytvoření spustitelného programu ze zdrojových souborů jazyka C.

1. Preprocessing
   - zpracování direktiv (#include, #define ...)
   - výsledkem je rozbalený zdrojový kód bez direktiv
2. Kompilace
   - preprocesovaný kód se přeloží do strojového kódu pro daný procesor, ale ještě není spustitelný
   - vzniká objektový soubor (.o nebo .obj)
3. Linkování
   - linker spojí všechny objektové soubory a potřebné knihovny (např. stdio, math) do jednoho celku
   - výsledkem je spustitelný soubor (.exe, .out apod.)
4. Loading (nahrání do paměti)
   - po spuštění programu ho operační systém nahraje do paměti a předá řízení funkci `main()`

Jaký rozdíl mezi použitím #include `<soubor.h>` a #include `"soubor.h"`?

Jedná se o hlavičkové soubory.

- #include `<soubor.h>` - hledá v systémových adresářích, používá se pro standardní knihovny
- #include `"soubor.h"` - nejprve aktuální adresář, poté systémové, dá se zadat i cesta, jedná se o vlastní hlavičkové soubory

---

Co je úkolem preprocesoru?

Zpracovává direktiva (příkazy začínající #).

- `#def` nahrazuje makra
- `#include` vkládá hlavičky
- `#ifdef`, `#ifndef` podmíněná kompilace částí kódu

Jaký je rozdíl mezi produktem kompilátoru a produktem linkeru?

- kompilátor - objektové soubory (.o nebo .obj), což je strojový kód jednotlivých funkcí, není zatím spustitelný
- linker - spustitelný program (.exe, .out, .elf), jelikož spojí všechny části programu a knihovny

---

Datové typy

Jak zjistíme velikost datové reprezentace základních celočíselných typů v jazyce C?

Pomocí operátoru `sizeof()`, který vrací velikost typu nebo proměnné v bajtech, např. `sizeof(int)`.

Co je to endianita?

Endianita určuje pořadí bajtů v paměti pro vícebajtová čísla; například číslo `0x12345678` bude v paměti uloženo jako:

- **Big-endian:** 12 34 56 78 (nejvýznamnější bajt první)
- **Little-endian:** 78 56 34 12 (nejméně významný bajt první)

Jak je v paměti počítače reprezentován datový typ float (IEEE 754)?

Datový typ `float` je reprezentován 32 bity: 1 bit pro znaménko, 8 bitů pro exponent a 23 bitů pro mantisu (hodnotu čísla).

Jak v C přistupujeme k datovým položkám složeného typu (struct)?

```
struct Point { int x; int y; };
struct Point p;
p.x = 10;
p.y = 20;

struct Point *ptr = &p;
ptr->x = 30;
ptr->y = 40;
```

K položkám struktury přistupujeme pomocí tečky (`.`) u proměnné typu struct a pomocí šipky (`->`) u ukazatele na strukturu.

Deklarujte ukazatel na proměnnou datového typu int.

```
int *x;
```

Vysvětlete, jaký je v C rozdíl mezi proměnnou typu ukazatel, proměnnou a polem z hlediska uložení v paměti.

- **Proměnná** – obsahuje přímo hodnotu, velikost závisí na typu (např. int, float), uložena na konkrétní adrese v paměti.

- **Ukazatel** – obsahuje **adresu** jiné proměnné, velikost je obvykle 4 B (32bit) nebo 8 B (64bit), hodnota cílové proměnné se získá dereferencí (*ptr).
- **Pole** – sekvence souvisle uložených proměnných stejného typu, velikost = typ × počet prvků, název pole v některých výrazech degraduje na ukazatel na první prvek, přístup přes index (arr[i]).

Deklarujte pole variabilní délky n, kterou načtete ze standardního vstupu.

```c
#include <stdio.h>

int main(void) {
    int n;
    scanf("%d", &n);
    int array[n];
    return 0;
}
```

Jaký je rozdíl mezi konstantním ukazatelem a ukazatelem na konstantní hodnotu?

- konstantní ukazatel - nelze měnit adresu, na kterou ukazuje, `int * const ptr;`
- ukazatel na konstantní hodnotu - může ukazovat kamkoli, ale hodnotu, na kterou ukazuje, nelze měnit přes tento ukazatel, `const int *ptr;`

Jak jsou v C reprezentovány textové řetězce?

Jako pole znaků (`char`), `char str[] = "Hello";`

Stručně popište typ union používaný v jazyce C.

Typ `union` umožňuje uložit různé datové typy na stejné místo v paměti, takže vždy zabírá tolik paměti, kolik má největší člen, a v daném okamžiku lze používat jen jeden z jeho členů.

Jak přistoupit na položku `number` proměnné data typu struktura?

```c
struct Data {
    int number;
    char letter;
};
d.number = 42;
```

Pomocí operátoru tečka: pokud máme `struct Data d;`, pak se na položku přistoupí jako `d.number`.

Jak přistoupit na položku `number` proměnné data typu ukazatel na strukturu?

```c
struct Data {
    int number;
    char letter;
};
```

```c
struct Data d = {42, 'A'};
struct Data *p = &d;
p->number = 99;
```

Pomocí operátoru šipka `->`: pokud máme ukazatel `struct Data *p;`, pak se na položku přistoupí jako `p->number`.

Jak lze rozlišit literál datového typu `double` a `float`?

Literál typu `float` se označuje příponou `f` nebo `F` (např. `3.14f`), zatímco literál typu `double` se příponou nepíše (např. `3.14`).

Jak lze rozlišit literál datového typu `long` a `int`?

Literál typu `long` se označuje příponou `l` nebo `L` (např. `1000L`), zatímco literál typu `int` žádnou příponu nemá (např. `1000`).

Inicializujte 2D pole velikosti 3×3 čísly 1 - 9 jdoucími vzestupně.

```c
int array[3][3] = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};
```

Co je hodnotou proměnné typu ukazatel?

Hodnota proměnné typu ukazatel je adresa paměťového místa, kam ukazatel odkazuje, tedy místo, kde je uložen obsah jiné proměnné.

Jakým způsobem lze v C zjistit délku textového řetězce?

Délku textového řetězce lze zjistit pomocí funkce `strlen()`, která vrací počet znaků až po nulovém znaku `'\0'`.

Co ovlivňuje u celočíselných typů modifikátor `unsigned`?

Modifikátor `unsigned` určuje, že proměnná může nabývat pouze nezáporných hodnot, čímž se zdvojnásobí kladný rozsah oproti stejnému typu bez `unsigned`.

K čemu slouží operátor `typedef`?

```c
typedef unsigned int uint;
```

Operátor `typedef` slouží k vytvoření nového aliasu pro existující datový typ, čímž lze psát čitelnější a kratší deklarace.

Standardní vstup a výstup

Patří funkce printf a scanf mezi příkazy programovacího jazyka C?

Ne, nepatří. Jsou to funkce standardní knihovny z hlavičkového souboru `<stdio.h>`.

Jaké jsou možné návratové hodnoty funkce scanf?

Vrací počet úspěšně načtených a přiřazených položek.

- >0 - načetlo se úspěšně n položek
- 0 - nepodařilo se načíst žádnou hodnotu
- EOF (obvykle -1) - konec souboru nebo chyba vstupu

Jak v C zajistíte načtení textového řetězce ze souboru aniž byste překročili alokovanou paměť určenou pro uložení řetězce?

Pomocí funkce `fgets()`.

```c
#include <stdio.h>

int main(void)
{
    char text[100];
    FILE *soubor = fopen("data.txt", "r");

    fgets(text, sizeof(text), soubor);
    printf("Text ze souboru: %s\n", text);

    fclose(soubor);
    return 0;
}
```

Jak v C zajistíte načtení řetězce, který bude obsahovat bílý znak?

```c
#include <stdio.h>

int main(void) {
    char buffer[100];

    printf("Zadejte text s mezerami: ");
    if (fgets(buffer, sizeof(buffer), stdin) != NULL) {
        printf("Nacteno: %s", buffer);
    }

    return 0;
}
```

Jak lze vynutit u funkce printf vypsání znaménka u číselných datových typů?

Použitím znaku + ve formátovacím řetězci, například %+d nebo %+f, což způsobí, že printf vypíše znaménko i u kladných čísel.

Co je to bílý znak? Je možné ho načíst pomocí funkce scanf?

Bílý znak je libovolný znak, který odděluje text – typicky mezera, tabulátor (`\t`) nebo nový řádek (`\n`). `Scanf()` ho nenačte, je vhodné použít `fgets()`.

Vypište funkce pro vypsání dat na standardní výstupu a stručně popište jejich činnost (max. jedna věta u každé funkce).

- `printf()` – vypisuje formátovaný text nebo hodnoty proměnných na standardní výstup.
- `putchar()` – vypíše jeden znak na standardní výstup.
- `puts()` – vypíše řetězec znaků a automaticky přidá nový řádek.
- `fprintf(stdout, ...)` – vypisuje formátovaný text na libovolný soubor, typicky standardní výstup.
- `fputs(..., stdout)` – vypíše řetězec na standardní výstup bez automatického nového řádku.

Vypište funkce pro načtení dat ze standardního vstupu a stručně popište jejich činnost (max. jedna věta u každé funkce).

- `scanf()` – načítá formátovaná data ze standardního vstupu do proměnných.
- `getchar()` – načte jeden znak ze standardního vstupu.
- `gets()` – načte řetězec znaků až po nový řádek (bez kontroly délky, nebezpečné).
- `fgets(..., stdin)` – bezpečně načte řetězec znaků až po nový řádek, včetně kontroly maximální délky.
- `sscanf()` – čte formátovaná data z řetězce (nikoli přímo ze standardního vstupu).

Jak lze poznat, že při čtení standardního vstupu nepřijdou již další data (tj. bylo např. dosaženo konce souboru přesměrovaného na standardní vstup)?
To lze zjistit pomocí návratové hodnoty funkcí pro čtení (např. `scanf`, `fgets`, `getchar`), která signalizuje `EOF` (end-of-file), nebo funkcí `feof(stdin)`, která vrací nenulovou hodnotu, pokud byl dosažen konec vstupu.

Jaký je rozdíl mezi zápisem na standardní výstup a na standardní chybový výstup?
Standardní výstup (`stdout`) se používá pro běžná data programu a je obvykle směrován na terminál nebo do souboru, zatímco standardní chybový výstup (`stderr`) slouží pro chybová hlášení a upozornění a bývá často směrován odděleně, aby bylo možné chybové zprávy zachovat i při přesměrování běžného výstupu.

Má funkce printf návratovou hodnotu? Pokud ano, jaký je její význam?
Ano, funkce `printf()` vrací počet úspěšně vypsaných znaků, nebo zápornou hodnotu v případě chyby při zápisu na standardní výstup.

Bude fungovat následující zápis čtení `scanf("%i", a)`? Pokud ano, jak vypadá deklarace proměnné a tisk načteného obsahu?
Ne, zápis fungovat nebude, jelikož funkce `scanf()` vyžaduje ukazatele na proměnnou.

Jak lze bezpečně načíst pomocí `scanf` jeden odstavec textu obecné délky? Uveďte příklad.

```
fgets(text, sizeof(text), stdin);
```

Řídicí struktury

Jakého datového typu může být řídící proměnná cyklu for?

Řídicí proměnná cyklu for může být libovolného celočíselného nebo reálného datového typu – nejčastěji int, ale může to být i char, long, float nebo double.

Jaký je rozdíl mezi cykly while a do-while?

While nejprve kontroluje podmínku a tělo se nemusí provést ani jednou, do-while nejprve provede tělo a pak kontroluje podmínku, takže se provede alespoň jednou.

Co způsobí příkaz continue?

Příkaz `continue` přeskočí zbytek aktuální iterace cyklu a pokračuje další iterací.

Co způsobí příkaz break?

Příkaz `break` okamžitě ukončí nejbližší obalující cyklus nebo `switch` a pokračuje za něj.

Popište funkci ternárního operátoru.

```
int a = 5, b = 10;
int max = (a > b) ? a : b; // pokud je a > b, max = a, jinak max = b
```

Ternární operátor `?:` vyhodnotí podmínku a podle její pravdivosti vrátí jednu ze dvou hodnot; zápis: `podmínka ? hodnota_true : hodnota_false`.

Jak je v příkazu větvení switch ošetřena varianta, která nevyhovuje podmínce žádné z větví?

Pomocí volitelné větve `default`, která se provede, pokud žádná z hodnot `case` neodpovídá.

Jaký je rozdíl mezi větvením programu pomocí příkazu if a pomocí příkazu switch?

Příkaz `if` umožňuje vyhodnocovat libovolné logické podmínky, zatímco `switch` porovnává hodnotu jedné proměnné s konstantami jednotlivých větví.

Co je to blok a jaký je jeho význam v jazyce C?

Blok je část kódu uzavřená do složených závorek `{}`, která umožňuje seskupit více příkazů a definovat lokální rozsah proměnných.

Nesetříděno

Jak lze v jazyce C realizovat předání parametru funkci odkazem?

V jazyce C se předání odkazem realizuje pomocí ukazatele – do funkce se předává adresa proměnné a uvnitř funkce se na ni přistupuje dereferencí.

Co je to literál a co tímto pojmem označujeme?

Literál je přímo zadaná hodnota v kódu programu, například číslo `42`, znak `'A'` nebo řetězec `"Hello"`.

Co znamená klíčové slovo `volatile`?

Klíčové slovo `volatile` označuje proměnnou, jejíž hodnota se může měnit mimo kontrolu programu, např. hardwarem nebo jiným vláknem, a kompilátor proto nesmí optimalizovat její čtení/zápis.

K čemu slouží modifikátor `const`?

Modifikátor `const` označuje proměnnou jako neměnnou, tj. její hodnotu nelze po inicializaci měnit.

Jaký význam má klíčové slovo `extern` v závislosti na kontextu?

Klíčové slovo `extern` označuje, že proměnná nebo funkce je definována jinde (v jiném souboru nebo později v programu) a umožňuje její sdílení mezi soubory.

- Deklarace proměnné bez inicializace: `extern int x;` – říká kompilátoru, že `x` existuje někde jinde.
- Deklarace funkce: `extern void foo();` – označuje, že definice funkce je mimo aktuální soubor.

Jaký význam má klíčové slovo `static` v závislosti na kontextu?

Klíčové slovo `static` mění dobu života a viditelnost proměnných a funkcí.

- **U proměnných v funkcích:** Proměnná si zachovává hodnotu mezi voláními funkce (*trvalá lokální proměnná*).
- **U globálních proměnných:** Omezuje viditelnost proměnné jen na aktuální soubor (*file scope*).
- **U funkcí:** Funkce je viditelná pouze v aktuálním souboru.

Jaké typy paměti dle způsobu alokace rozlišujeme v jazyce C?

V jazyce C rozlišujeme paměť podle způsobu alokace na:

- **Statickou paměť** – alokovaná při kompilaci, existuje po celou dobu běhu programu (globální a statické proměnné).
- **Automatickou paměť** – alokovaná při vstupu do bloku/funkce, uvolněná při jeho opuštění (lokální proměnné bez `static`).
- **Dynamickou paměť** – alokovaná za běhu programu pomocí funkcí `malloc`, `calloc`, `realloc` a uvolněná funkcí `free`.

Vyjmenujte čtyři specifikátory paměťových tříd.

V jazyce C rozlišujeme čtyři základní specifikátory paměťových tříd:

- `auto` – standardní lokální proměnné (implicitní, většinou se nepíše)
- `register` – doporučení překladači uložit proměnnou v registru CPU pro rychlejší přístup
- `static` – zachovává hodnotu proměnné mezi voláními funkcí, lokální proměnné mají statickou životnost
- `extern` – deklaruje proměnnou, která je definovaná jinde (globálně)

Jak v C alokujete dynamickou paměť pro uložení 20 hodnot typu int a později zvětšíte pole pro 10 dalších položek?

```c
#include <stdlib.h>

int *pole = malloc(20 * sizeof(int));
pole = realloc(pole, 30 * sizeof(int));

free(pole);
```

Jak se v jazyce C předává pole funkcím?

```c
#include <stdio.h>

void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main(void) {
    int numbers[] = {1, 2, 3, 4, 5};
    int len = sizeof(numbers) / sizeof(numbers[0]);

    printArray(numbers, len);
    return 0;
}
```

Pole se v jazyce C předává funkcím **vždy jako ukazatel na první prvek** (adresu `&arr[0]`), nikoli jako kopie celého pole. Proto musí být délka pole předána zvlášť, obvykle jako další argument. Deklarace `void func(int arr[])` a `void func(int *arr)` jsou ekvivalentní.

Jakého datového typu je návratová hodnota funkce `malloc`?

```c
#include <stdlib.h>

int *arr = malloc(5 * sizeof(int));
if (arr == NULL) {
    // alokace selhala
}
```

Funkce `malloc` vrací **ukazatel typu** `void *` (neboli „neznámý ukazatel“). To znamená, že ukazuje na blok paměti bez určení konkrétního typu. V jazyce C se tento ukazatel může **automaticky přetypovat** na jakýkoli jiný ukazatel (např. `int *`, `char *` apod.). V C++ je však nutné `void *` přetypovat explicitně pomocí `(typ *)malloc(...)`.

Jaký je rozdíl mezi funkcemi `malloc` a `calloc`?

```c
#include <stdlib.h>

int *a = malloc(5 * sizeof(int));
int *b = calloc(5, sizeof(int));
```

Obě funkce slouží k dynamické alokaci paměti, ale liší se ve dvou věcech:

- `malloc(size)` — alokuje blok paměti o dané velikosti (v bajtech), ale **neinicializuje** ho (obsahuje náhodné hodnoty).
- `calloc(n, size)` — alokuje paměť pro `n` prvků po `size` bajtech a **inicializuje všechny bajty na nulu**.

Obě funkce vrací ukazatel typu `void *`, který je potřeba uvolnit funkcí `free()`.

Jak lze v programu v jazyce C zjistit jeho vlastní jméno?

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
    printf("Jmeno programu: %s\n", argv[0]);
    return 0;
}
```

V jazyce C je jméno programu dostupné přes první argument funkce `main`, tedy `argv[0]`. Pole `argv` obsahuje všechny parametry příkazového řádku, přičemž `argv[0]` je vždy cesta nebo jméno spouštěného programu. Tento způsob funguje standardně na většině operačních systémů.

Jak jsou v programu zpracovány argumenty příkazové řádky?

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
    printf("Program dostal %d argumentu:\n", argc);
    for (int i = 0; i < argc; i++) {
        printf("argv[%d] = %s\n", i, argv[i]);
    }
    return 0;
}
```

Argumenty příkazové řádky jsou předávány do funkce `main` přes parametry `argc` a `argv`:

- `argc` – počet argumentů (včetně názvu programu).
- `argv` – pole ukazatelů na řetězce (`char *`), kde každý řetězec je jeden argument.

Pomocí smyčky lze projít všechny argumenty a zpracovat je podle potřeby programu.

Charakterizujte stručně paměťové třídy v jazyce C.

- **auto** – standardní lokální proměnné funkcí, alokované při vstupu do bloku a uvolněné při jeho opuštění.
- **register** – doporučení překladači uložit proměnnou v CPU registru pro rychlejší přístup.
- **static** – proměnná si uchovává hodnotu mezi voláními funkce nebo omezuje viditelnost globální proměnné na soubor.
- **extern** – deklaruje proměnnou nebo funkci definovanou v jiném souboru, umožňuje její sdílení mezi soubory.

Popište, jak v C probíhá volání funkce `int doit(int r)` a kam jsou hodnoty ukládány.

Při volání funkce `doit(r)` se stane následující:

- Hodnota parametru `r` se **zkopíruje** do lokální proměnné funkce (tzv. předání hodnotou).
- Prostor pro lokální proměnné a parametr se alokuje na **zásobníku (stack)**.
- Po dokončení funkce je její návratová hodnota typu `int` vrácena obvykle přes **registr CPU (např. EAX/RAX)**.
- Po návratu z funkce se zásobník upraví a lokální proměnné jsou uvolněny.

Stručně: parametry a lokální proměnné = stack, návratová hodnota = registr, volání probíhá přes kopii argumentů.

Jakými dvěma způsoby lze v C vytvářet konstanty?

- **Použitím klíčového slova** `const` – proměnná označená jako `const` nelze po inicializaci měnit, např. `const int x = 10;`.
- **Pomocí direktivy** `#define` – preprocesor nahradí všechna výskyty symbolu hodnotou před kompilací, např. `#define PI 3.14159`.

Jakým způsobem otevřete soubor pro čtení? Napište krátký kód.

```c
#include <stdio.h>

FILE *soubor = fopen("data.txt", "r");
if (soubor == NULL) {

}
```

Soubor se otevře funkcí `fopen` s režimem `"r"` pro čtení, a vrací ukazatel typu `FILE *`, který se používá pro další operace nad souborem.

Jakým způsobem otevřete soubor pro zápis? Napište krátký kód.

```c
#include <stdio.h>

FILE *soubor = fopen("vystup.txt", "w");
if (soubor == NULL) {

}
```

Soubor se otevře funkcí `fopen` s režimem `"w"` pro zápis; pokud soubor neexistuje, vytvoří se, pokud existuje, přepíše se, a funkce vrací ukazatel typu `FILE *`.

Jaké znáte logické operátory jazyka C a jak se zapisují?

- **AND (logické a)** – `&&`, výsledkem je 1 (true), pokud jsou oba operandy pravdivé.
- **OR (logické nebo)** – `||`, výsledkem je 1, pokud je alespoň jeden operand pravdivý.
- **NOT (negace)** – `!`, převrací hodnotu operandu ($0 \rightarrow 1$, nenulové $\rightarrow 0$).

Jaké znáte bitové operátory jazyka C a jak se zapisují?

- **AND (bitové a)** – `&`, každý bit výsledku je 1, pokud jsou oba odpovídající bity operandů 1.
- **OR (bitové nebo)** – `|`, každý bit výsledku je 1, pokud je alespoň jeden odpovídající bit operandů 1.
- **XOR (bitový exkluzivní nebo)** – `^`, každý bit výsledku je 1, pokud se odpovídající bity operandů liší.
- **NOT (bitová negace)** – `~`, převrací všechny bity operandu ($0 \rightarrow 1$, $1 \rightarrow 0$).
- **Posun vlevo** – `«`, posune bity vlevo o zadaný počet pozic (nuly se doplní zprava).
- **Posun vpravo** – `»`, posune bity vpravo o zadaný počet pozic (u unsigned se doplňují nuly, u signed záleží na implementaci).

Jak v C realizujete dělení a násobení dvěma s využitím operátorů bitového posunu?

```c
int x = 8;
int y;

y = x << 1;  // posun bitu doleva o 1, ekvivalent x * 2

y = x >> 1;  // posun bitu doprava o 1, ekvivalent x / 2 (u kladnych
   cisel)
```

Pomocí bitového posunu doleva (`«`) lze číslo násobit dvěma a pomocí bitového posunu doprava (`»`) lze číslo dělit dvěma (pozor u záporných čísel, může se lišit chování dělení).

Co je v jazyce C pointerová (ukazatelová) aritmetika a jak se používá?

```c
#include <stdio.h>

int main(void) {
    int array[5] = {10, 20, 30, 40, 50};
    int *p = array;  // ukazatel na prvni prvek pole

    printf("%d\n", *p);   // 10
    p++;                   // posun na dalsi prvek pole
    printf("%d\n", *p);   // 20

    p += 2;                // posun o 2 prvky vpred
    printf("%d\n", *p);   // 40

    return 0;
}
```

Pointerová aritmetika umožňuje provádět operace s ukazateli, jako je **přičítání nebo odečítání čísel**, aby ukazatel posunul svou adresu o několik prvků daného typu, a **odečítání ukazatelů** pro zjištění vzdálenosti mezi dvěma adresami; výsledkem dereference (`*p`) získáváme hodnotu na nové adrese.

Jak se v jazyce C pouziva operator pretipovani?

```c
#include <stdio.h>

int main(void) {
    double x = 3.14;
    int y;

    y = (int)x;   // pretipovani double na int
    printf("%d\n", y);  // 3

    return 0;
}
```

Operator přetypování (cast) umožňuje dočasně změnit datový typ výrazu, zápis: `(typ) vyraz`, např. `(int)x` převede `x` z typu `double` na `int`.

Jak v C zapiste konstantni ukazatel na konstantni hodnotu typu double?

```c
#include <stdio.h>

int main(void) {
    const double val = 3.14;
    const double * const ptr = &val;  // konstantni ukazatel na
       konstantni hodnotu

    printf("%f\n", *ptr);  // 3.140000

    return 0;
}
```

Deklarace `const double * const ptr` vytvori ukazatel, ktery nemuze menit adresu ani hodnotu, na kterou ukazuje.

Co je v C ukazatel na funkci? K cemu slouzi a jak definujete promenou typu ukazatel na funkci?

```c
#include <stdio.h>

// definice funkce
int add(int a, int b) {
    return a + b;
}

int main(void) {
    // deklarace ukazatele na funkci, ktera bere 2 int a vraci int
    int (*func_ptr)(int, int) = add;

    int result = func_ptr(2, 3);  // volani funkce pres ukazatel
    printf("%d\n", result);       // 5

    return 0;
}
```

Ukazatel na funkci je proměnná, která uchovává adresu funkce, umožňuje volat funkci přes ukazatel a používat funkce dynamicky; deklarace: `typ_navratovy (*nazev_ukazatele)(parametry) = funkce;`.

Můžeme v C při definici proměnné typu pole přímo inicializovat? Pokud ano, jak?

```c
#include <stdio.h>

int main(void) {
    int array[5] = {1, 2, 3, 4, 5};  // inicializace pri definici

    for(int i = 0; i < 5; i++) {
        printf("%d ", array[i]);
    }
    printf("\n");

    return 0;
}
```

Ano, při definici pole lze prvky inicializovat uvedením hodnot ve složených závorkách, například `int array[5] = {1,2,3,4,5};`.

Můžeme v C při definici proměnné typu struct inicializovat pouze určitou položku?

```c
#include <stdio.h>

struct Point {
    int x;
    int y;
};

int main(void) {
    struct Point p = {.y = 10};  // inicializace pouze y, x bude 0
```

```c

    printf("x = %d, y = %d\n", p.x, p.y);  // x = 0, y = 10

    return 0;
}
```

Ano, lze inicializovat jen určité položky pomocí označené inicializace (`.nazev_polozky = hodnota`); ostatní členy se nastaví na nulu nebo NULL.

Co vraci operator sizeof?

```c
#include <stdio.h>

int main(void) {
    int x;
    double y;
    char array[10];

    printf("Velikost int: %zu\n", sizeof(x));
    printf("Velikost double: %zu\n", sizeof(y));
    printf("Velikost pole char[10]: %zu\n", sizeof(array));

    return 0;
}
```

Operátor `sizeof` vrací velikost typu nebo proměnné v bajtech, například `sizeof(int)` vrací počet bajtů potřebných pro uložení celočíselného typu int.

Charakterizujte rozdíl mezi polem a spojovým seznamem.

- **Pole** – sekvence prvků stejného typu uložených souvisle v paměti, rychlý přístup přes index, pevná velikost.
- **Spojový seznam** – prvky (uzly) uložené kdekoli v paměti, každý uzel obsahuje hodnotu a ukazatel na další prvek, snadné vkládání a mazání, pomalejší náhodný přístup.

Navrhněte datovou strukturu pro vytvoreni spojoveho seznamu.

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

// definice uzlu spojoveho seznamu
struct Node {
    int data;
    struct Node *next;  // ukazatel na dalsi prvek
};

int main(void) {
    // vytvoreni prvniho uzlu
    struct Node *head = malloc(sizeof(struct Node));
    head->data = 10;
    head->next = NULL;

```

```c
    // pridani druheho uzlu
    struct Node *second = malloc(sizeof(struct Node));
    second->data = 20;
    second->next = NULL;

    head->next = second;

    // vypis hodnot
    struct Node *current = head;
    while(current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");

    // uvolneni pameti
    free(second);
    free(head);

    return 0;
}
```

Spojový seznam lze reprezentovat pomocí struktury, kde každý uzel obsahuje hodnotu (např. `int data`) a ukazatel na další uzel (`struct Node *next`); hlava seznamu ukazuje na první prvek.

Charakterizujte abstraktni datovy typ. Co se pod timto pojmem mysli?

Abstraktní datový typ (ADT) je koncept, který definuje datovou strukturu a množinu operací nad ní, aniž by bylo specifikováno, jak jsou tyto operace implementovány; uživatel pracuje s ADT pouze přes jeho rozhraní a neřeší vnitřní reprezentaci dat.

Charakterizujte zakladni rozdily mezi zasobnikem a frontou.

- **Zásobník (stack)** – princip LIFO (Last In, First Out), poslední vložený prvek se vyjme jako první, základní operace: `push` a `pop`.
- **Fronta (queue)** – princip FIFO (First In, First Out), první vložený prvek se vyjme jako první, základní operace: `enqueue` a `dequeue`.

Vysvetlete princip rekurze napr. na vypoctu faktorialu. Charakterizujte hlavni rozdily mezi rekurzivnim a iterativnim vypoctem.

```c
#include <stdio.h>

// rekurzivni funkce pro faktorial
int factorial(int n) {
    if(n <= 1)
        return 1;
    else
        return n * factorial(n - 1);
}

```

```c
int main(void) {
    int n = 5;
    printf("Faktorial %d je %d\n", n, factorial(n));
    return 0;
}
```

Rekurze je metoda, kdy se funkce volá sama sebe, například faktoriál n! = n * (n-1)!; hlavní rozdíly:

- **Rekurzivní výpočet** – kód je často kratší a přehlednější, využívá zásobník volání funkcí, může způsobit přetečení zásobníku při velkých hodnotách.
- **Iterativní výpočet** – využívá smyčky (for, while), efektivnější z hlediska paměti, nižší režie volání funkcí, někdy méně intuitivní zápis.
