https://github.com/MichalLebeda/it/blob/master/otazky/3.md

Podprogramy a jejich využití

- Podprogramy (někdy nazývané také funkce, procedury nebo metody) jsou oddělené části kódu, které provádějí určité úkoly nebo operace. Tyto úkoly mohou být prováděny opakovaně na různých místech ve vašem programu.
- Funkce se skládají ze 2 hlavních částí: hlavička (její definice a název) a tělo(kód)
- **Výhody:**
    - **Modularita:** Podprogramy umožňují rozdělit kód do menších a lépe spravovatelných částí. To usnadňuje čtení, údržbu a rozšiřování kódu.
    - **Opakovatelnost:** Můžete opakovaně použít stejný kód na různých místech ve vašem programu nebo dokonce v různých programech.
    - **Zvýšení přehlednosti:** Použití podprogramů může zvýšit přehlednost kódu tím, že oddělí různé části funkcionality do samostatných bloků, což usnadňuje porozumění kódu.
    - **Zmenšení redundance:** Díky opakovanému použití podprogramů můžete snížit redundanci kódu a snížit možnost chyb v kódu.
    - **Testovatelnost:** Oddělené podprogramy lze snadněji testovat nezávisle na zbytku kódu, což usnadňuje detekci a opravu chyb.
    -
- **Nevýhody:**
    - **Časová náročnost:** Volání podprogramů přináší určitou režii kvůli přenosu řízení mezi různými částmi programu. Při použití velkého množství podprogramů může být výkonově náročné.
    - **Správa paměti:** Pokud podprogramy pracují s velkými datovými strukturami nebo vytvářejí mnoho nových objektů, může být náročné spravovat paměť a zabránit úniku paměti.
    - **Komplexita kódu:** Příliš mnoho podprogramů, každý s mnoha parametry, může vést k složitějšímu kódu a zmatenější architektuře programu.
    - **Nároky na debugging:** Při ladění programů může být obtížné sledovat tok řízení mezi různými podprogramy a identifikovat, kde a proč vznikají chyby.
    - **Návrhové problémy:** Pokud nejsou podprogramy správně navrženy a organizovány, mohou vést k problémům s vytvářením a údržbou kódu.
    -

Funkce a procedury

♦ **Funkce**

– Ve světě programování je funkce blok kódu, který má určenou funkcionalitu a může být zavolán z jiné části programu. Funkce obvykle přijímá vstupní hodnoty (argumenty), provádí určité operace a může vrátit výstupní hodnotu.
– Funkce jsou užitečné, protože umožňují znovupoužitelnost kódu, zlepšují čitelnost kódu a umožňují rozdělení programu na menší, lépe spravovatelné části
– Při její volání program přeskočí do funkce a po jejím provedení se vrátí na místo volání.

♦ **Procedura**

– je zvláštním případem funkce – nemá návratovou hodnotu a nemusí mít ani vstupní parametry.
– Používají se často při dávkovém zpracování – např. každou hodinu zavoláme proceduru, která zpracuje objednávky, které se nashromáždily v databázi, a předá je do jiného systému.
– Procedura formálně v jazyku C neexistuje.

Rekurze

♦ Rekurze v programování je technika, při které funkce volá sama sebe, buď přímo nebo nepřímo, k řešení problémů. To znamená, že funkce během svého vykonávání provádí stejný kód, ale s různými vstupy, až je splněna určitá podmínka ukončení, která zabraňuje nekonečnému volání sama sebe.
♦ Data, která jsou funkci předávána, se ukládají do takzvaného **zásobníku.**
=> To je nějaké vyhrazené místo v paměti. **=>**Po skončení funkce se data ze zásobníku zase odstraní.
♦ Pokud však funkce během svého vykonávání zavolá samu sebe, pak se do zásobníku umístí další data a tak stále dokola => Může se zacyklit => To je samozřejmě velice náročné na paměť a může vést až ke zhroucení programu. Použití rekurze je sice efektní, ale ne vždy efektivní.
♦ Klasickým příkladem na rekurzi je výpočet faktoriálu nebo třídění (avšak i faktoriál lze vypočítat elegantně bez rekurze).

♦Rekurze může být mocný nástroj pro řešení určitých problémů, ale je důležité si uvědomit, že nezvládnutá rekurze může vést k přetečení zásobníku (stack overflow) a výkonovým problémům. Je také důležité zajistit, aby byla rekurze ukončena, což může vyžadovat pečlivé plánování a kontrolu vstupních hodnot.