Programování

1. Definice

- **Programování** je proces navrhování a vytváření sekvence instrukcí, které počítač vykonává.
- Cílem programování je řešit konkrétní problémy, automatizovat úkoly nebo vytvářet aplikace, které vykonávají specifické operace.

2. Programovací jazyky

2.1 **Dělení programovacích jazyků**:

- **Nízké jazyky** (strojový kód, assembler):
  - Strojový kód je jediný jazyk, který počítač přímo rozumí.
  - Assembler je nízkoúrovňový jazyk, který se používá k přímé komunikaci s hardwarem. Kód je psán v binárních instrukcích nebo v podobě mnemonických symbolů.
- **Vysoké jazyky**:
  - **Imperativní jazyky**: Programování na základě definování kroků a sekvencí příkazů (např. C, Java, Python).
  - **Deklarativní jazyky**: Programování, kde se specifikuje, co má být provedeno, bez nutnosti specifikovat, jak (např. SQL, HTML, Prolog).
  - **Objektově orientované jazyky (OOP)**: Jazyky, které používají objekty a třídy jako základní stavební bloky programů (např. Java, Python, C++).
  - **Funkcionální jazyky**: Jazyky, které se zaměřují na funkce jako základní stavební bloky programu (např. Haskell, Lisp).
  - **Skriptovací jazyky**: Vysoké jazyky určené pro automatizaci úkolů, například JavaScript, PHP, Ruby, Python.

2.2 **Příklady nejběžnějších jazyků**:

- **C**: Nízká úroveň abstrakce, používá se pro systémové programování.
- **Python**: Vysoce čitelný jazyk, oblíbený pro skriptování a vývoj aplikací.
- **Java**: Objektově orientovaný jazyk, široce používaný ve firemních aplikacích.
- **JavaScript**: Jazyk pro webový vývoj, běží na straně klienta v prohlížeči.
- **C++**: Rozšíření jazyka C, používá se v oblasti vývoje her a náročných aplikací.
- **Swift**: Jazyk pro vývoj aplikací pro Apple zařízení.

3. Programovací paradigma

3.1 **Imperativní programování**:

- Programovací paradigma, kde se specifikuje postup, jakým způsobem má být úkol vykonán.
- Používá se pro popis sekvence operací, jako jsou přiřazení hodnot, cykly, podmínky.
- Jazyk: C, Java, Python.

3.2 **Objektově orientované programování (OOP)**:

- Zaměřuje se na objekty, které obsahují data (atributy) a metody (funkce).

- Hlavní principy:
    - **Encapsulace**: Ukrytí implementace a expozice pouze potřebných částí.
    - **Dědičnost**: Vytváření nových tříd na základě existujících tříd.
    - **Polymorfismus**: Možnost mít různé implementace stejné funkce/metody.
    - **Abstrakce**: Skrývání složitosti a zaměření se na základní koncepty.
- Jazyk: Java, C++, Python.

3.3 **Funkcionální programování**:

- Zaměřuje se na používání funkcí jako základní stavební bloky.
- **Čisté funkce**: Funkce, které mají stejné výstupy pro stejné vstupy a nemění žádný stav.
- **Nevyjadřování stavu**: Funkcionální jazyky nepracují s proměnnými, ale místo toho používají konstanty a funkce.
- **Laziness**: Evaluace výrazů až ve chvíli, kdy jsou opravdu potřeba.
- Jazyk: Haskell, Lisp, Scala.

3.4 **Deklarativní programování**:

- Zaměřuje se na to, **co** má být provedeno, bez nutnosti specifikovat **jak**.
- Příklady:
    - **SQL**: Používá se pro práci s databázemi.
    - **HTML/CSS**: Používají se k definici vzhledu a struktury webových stránek.

3.5 **Logické programování**:

- Programy jsou vyjádřeny jako soubor logických pravidel a faktů, které jsou využívány k vyhledávání řešení.
- Jazyk: Prolog.

4. Struktura programu

4.1 **Algoritmy**:

- Algoritmus je přesně definovaná posloupnost kroků, které vedou k řešení problému.
- Klíčové vlastnosti algoritmů:
    - **Konečnost**: Algoritmus musí skončit po konečném počtu kroků.
    - **Přesnost**: Každý krok musí být jasně definován.
    - **Efektivita**: Algoritmus by měl být co nejrychlejší a co nejméně náročný na prostředky.

4.2 **Řízení toku programu**:

- **Sekvenční řízení**: Příkazy jsou prováděny jeden po druhém.
- **Podmínky (if-else)**: Rozhodování o tom, který blok kódu se vykoná.
- **Cykly (for, while)**: Opakování kódu, dokud není splněna podmínka.
- **Skoky (break, continue)**: Ovlivňují průběh vykonávání cyklů a podmínek.

4.3 **Funkce a procedury**:

- Funkce je blok kódu, který provádí určitou operaci a vrací výsledek.
- **Parametry**: Vstupy, které funkce přijímá.
- **Návratová hodnota**: Výsledek, který funkce vrací.

4.4 **Datové struktury**:

- **Pole (Array)**: Sekvenční uspořádání dat.
- **Seznamy (List)**: Dynamická struktura pro ukládání dat.
- **Zásobníky (Stack)**: Pracuje na principu LIFO (Last In First Out).
- **Fronty (Queue)**: Pracuje na principu FIFO (First In First Out).
- **Stromy**: Hierarchická struktura dat (např. binární stromy).
- **Grafy**: Skládají se z uzlů a hran mezi nimi.

5. Práce s pamětí

5.1 **Paměťové modely**:

- **Stack**: Paměťová oblast pro ukládání lokálních proměnných a návratových adres funkcí.
- **Heap**: Dynamická paměť, která se alokuje během běhu programu.

5.2 **Alokace paměti**:

- **Dynamická alokace**: Alokace paměti během běhu programu (např. `malloc` v C).
- **Statická alokace**: Alokace paměti během kompilace programu.

5.3 **Řízení paměti**:

- **Garbage Collection**: Automatické uvolňování nepoužívané paměti (např. v Javě a Pythonu).
- **Manuální uvolňování**: Programátor musí explicitně uvolnit paměť (např. v C++).

6. Ladění a optimalizace kódu

6.1 **Ladění (Debugging)**:

- Proces identifikace a opravy chyb v programu.
- Používání ladících nástrojů (GDB, IntelliJ IDEA, Visual Studio).
- **Tiskové výstupy**: Pomocí `print` nebo logovacích nástrojů k identifikaci chyb.

6.2 **Optimalizace**:

- **Optimalizace výkonu**: Zlepšení doby vykonávání programu.
- **Optimalizace paměti**: Zajištění efektivního využívání paměti.
- **Zlepšení čitelnosti**: Udržování kódu tak, aby byl snadno pochopitelný a udržovatelný.

7. Fáze vývoje softwaru

- **Analýza požadavků**: Shromáždění informací o potřebách uživatele.
- **Návrh**: Vytváření architektury systému.
- **Implementace**: Programování samotného softwaru.
- **Testování**: Zajištění, že software funguje podle požadavků.
- **Údržba**: Opravy chyb a vylepšování softwaru.