# 1. Algoritmizace a programování

## Pojem algoritmus, způsoby zápisu, metoda top-down, ladicí cyklus

**Algoritmus** je přesně definovaný postup či sada instrukcí, které slouží k vyřešení daného typu úlohy. V informatice označuje algoritmus konečnou sekvenci kroků, které lze mechanicky provést, a které vedou od zadaných vstupů k požadovanému výsledku.

Aby byl postup považován za algoritmus, musí splňovat několik základních vlastností:

- **Konečnost** – Algoritmus vždy skončí po konečném počtu kroků.
- **Determinovanost** – Každý krok je přesně a jednoznačně popsán, bez možnosti nejednoznačného výkladu.
- **Vstup a výstup** – Algoritmus pracuje s definovanými vstupy a vrací výstupy.
- **Obecnost** – Řeší celou třídu problémů, nikoliv pouze jeden konkrétní případ.
- **Efektivita** – Jednotlivé kroky jsou natolik jednoduché, že je lze provést mechanicky (ručně nebo počítačem).

**Způsoby zápisu algoritmů:**

- Slovní popis: Podobně jako kuchařský recept, krok za krokem popisuje činnost.
- Vývojový diagram: Grafické schéma postupu (bloky pro akce, větvení, cykly).
- Programovací jazyk: Implementace algoritmu ve zvoleném programovacím jazyce.

**Návrh algoritmu metodou top-down** znamená rozdělení složitého problému na menší, jednodušší podproblémy, které jsou dále rozpracovány až na úroveň konkrétních kroků. Tato metoda se často pojí s využitím rekurze.

**Ladicí (debug) cyklus programu** je proces, při kterém opakovaně spouštíme a testujeme program, vyhledáváme a odstraňujeme chyby (tzv. bugy), často za použití speciálních nástrojů (debuggerů) a výpisů stavu proměnných.

# Řídicí a datové struktury

**Řídicí struktury** určují pořadí provádění příkazů v programu:

- **Sekvenční struktura**: Příkazy jsou vykonávány postupně za sebou.
- **Podmíněná struktura**: Na základě podmínky může být vykonán jeden z několika bloků kódu (např. `if`, `else`, `switch`).
- **Cyklická struktura**: Umožňuje opakování části kódu do splnění určité podmínky (např. `for`, `while`, `do-while`).

**Datové struktury** jsou způsoby organizace a ukládání dat v paměti:

- **Pole (array)**: Kolekce prvků stejného typu, uložených na po sobě jdoucích místech v paměti, přístup přes index.
- **Seznam (list)**: Dynamická datová struktura umožňující růst i zkracování, typicky spojový seznam.
- **Ukazatel (pointer)**: Proměnná obsahující adresu v paměti, umožňuje dynamickou práci s pamětí.
- **Struktura (structure)**: Sdružuje více různých prvků pod jedním názvem (např. `struct` v C).

# Proměnné, identifikátory, datové typy, příkazy a operace

**Proměnná** je pojmenované místo v paměti, kde je uložená nějaká hodnota, kterou lze během běhu programu měnit.

**Identifikátor** je název proměnné, funkce nebo jiné entity v programu (např. `pocet`, `vypocetSoucet`).

**Datové typy** určují, jaký druh hodnot může proměnná obsahovat (např. `int` pro celá čísla, `float` pro desetinná čísla, `char` pro znak, `bool` pro logickou hodnotu).

**Přiřazovací příkaz** umožňuje nastavit hodnotu proměnné (`a = 5;`).

**Aritmetické operace** zahrnují sčítání, odčítání, násobení, dělení, zbytek po dělení.

**Logické operace** pracují s pravdivostními hodnotami (`AND`, `OR`, `NOT`, `==`, `!=` atd.).

# Podmíněné příkazy, větvení, cykly

**Podmíněné příkazy** jako `if`, `else`, `switch` umožňují provádět různé části kódu v závislosti na splnění podmínky.

**Větvení** je rozhodování, kterou cestou se v algoritmu pokračuje (např. podle hodnoty proměnné).

**Cykly** (`for`, `while`, `do-while`) opakují blok kódu, dokud je splněna určitá podmínka.

# Číslicová reprezentace datových typů, číselné soustavy

**Číslicová reprezentace** znamená, jak jsou hodnoty uloženy v počítači:

- **Celá čísla (integer)**: Uložena jako binární čísla v pevné šířce (např. 32 bitů).
- **Desetinná čísla (float, double)**: Uložena jako čísla s plovoucí desetinnou čárkou, určena mantisou a exponentem.
- **Řetězce (string)**: Sekvence znaků, znaky jsou reprezentovány kódem (např. ASCII, Unicode).
- **Logické hodnoty (boolean)**: Obvykle jeden bit (0 = false, 1 = true).

**Číselné soustavy**:

- **Desítková (decimální, základ 10)**: Číslice 0–9.
- **Binární (základ 2)**: Pouze 0 a 1. Klíčová pro práci počítače.
- **Osmičková (základ 8)**: Číslice 0–7, používána např. při zápisu práv v UNIXu.
- **Hexadecimální (základ 16)**: Číslice 0–9 a písmena A–F (pro 10–15), často využívaná pro zápis barev a adres.

# Rekurzivní a iterační postupy

**Rekurzivní postup** je takový, kdy funkce volá sama sebe. Rekurze se hodí pro řešení problémů, které lze rozdělit na podobné menší podproblémy (například faktoriál, Fibonacciho posloupnost, prohledávání stromů).

**Iterační postup** znamená opakování činnosti pomocí cyklu (např. výpočet faktoriálu přes `for` cyklus).

# Posuzování kvality algoritmu (časová složitost)

Kvalita algoritmu se hodnotí především podle **časové složitosti** – tj. jak rychle algoritmus zpracuje data v závislosti na jejich velikosti. Časová složitost se značí tzv. **velkým O (O-notation)**.

Základní složitosti:

- **O(1)**: Konstantní – doba běhu je nezávislá na velikosti vstupu.
- **O(log n)**: Logaritmická – roste velmi pomalu s velikostí vstupu.
- **O(n)**: Lineární – čas roste úměrně vstupu.
- **O(n log n)**: Typické pro efektivní třídící algoritmy.
- **O($n^2$), O($n^3$), ...**: Kvadratická, kubická – výrazně pomalejší při větším vstupu.
- **O($2^n$), O(n!)**: Exponenciální, faktoriální – extrémně náročné pro větší vstupy.

# Strukturované a abstraktní datové typy

**Strukturované datové typy** umožňují sdružovat více hodnot:

- **Pole (array)**: Kolekce hodnot stejného typu, indexované.
- **Struktura (structure)**: Sdružení více různých typů pod jedním názvem (například záznam osoby s jménem, věkem a adresou).

**Abstraktní datové typy** jsou popsány z hlediska operací, které nad nimi lze provádět, ne konkrétní implementací:

- **Zásobník (stack)**: Princip LIFO – poslední vložený prvek je první vyjmutý (push, pop).
- **Fronta (queue)**: Princip FIFO – první vložený prvek je první vyjmutý (enqueue, dequeue).
- **Seznam (list)**: Umožňuje vkládání, mazání a procházení prvků.
- **Množina (set)**: Kolekce unikátních prvků, neřeší pořadí.
- **Strom (tree)**: Hierarchická struktura, každý uzel má rodiče a může mít potomky. Speciální případ je binární strom (každý uzel max. dva potomky).

# Princip tvorby uživatelských funkcí

Funkce umožňují rozdělit program na menší, opakovaně použitelné celky. Typický vývoj funkce zahrnuje:

1. Analýzu problému a definici cíle funkce.
2. Určení vstupů a výstupů.
3. Vytvoření hlavičky (definice jména, parametrů a návratového typu).
4. Implementaci těla funkce.
5. Testování správnosti a začlenění do programu.

# Princip práce se soubory

Práce se soubory zahrnuje základní operace:

1. **Otevření souboru** v požadovaném režimu (čtení, zápis, přidávání).
2. **Čtení nebo zápis dat** do/z souboru (po řádcích, po znacích nebo celé soubory).
3. **Uzavření souboru** pro uvolnění systémových prostředků.

Většinu těchto operací poskytují standardní knihovny jazyků (např. `open/read/write/close` v C, `with open()` v Pythonu).

# Přidělování paměti

Paměť lze přidělovat několika způsoby:

- **Staticky**: Při překladu programu, paměť je fixní (např. globální proměnné).
- **Dynamicky**: Za běhu programu na tzv. haldě (heap), typicky pomocí funkcí jako `malloc`/`free` v C nebo operátorů `new`/`delete` v C++.
- **Automaticky**: Lokální proměnné ve funkcích, které se alokují na zásobníku (stack) a po skončení funkce se automaticky uvolní.

# Základní algoritmy třídění a vyhledávání dat

**Třídicí algoritmy:**

- **Bubble sort**: Postupně porovnává sousední prvky a vyměňuje je, dokud není seznam setříděn. Jednoduchý, ale neefektivní pro velké množiny dat ($O(n^2)$).
- **Insertion sort**: Každý prvek je vložen na správné místo mezi již setříděné prvky. Efektivní pro malé nebo téměř setříděné seznamy.
- **Selection sort**: Vyhledá nejmenší prvek a vloží ho na začátek, opakuje pro zbytek.
- **Quicksort, Mergesort, Heapsort**: Efektivnější algoritmy vhodné pro velké datové sady (obvykle O(n log n)).

**Vyhledávací algoritmy:**

- **Lineární vyhledávání**: Prochází každý prvek sekvenčně, dokud nenajde hledaný.
- **Binární vyhledávání**: Funguje na setříděných datech, v každém kroku dělí interval na poloviny, hledá v odpovídající polovině.
- **Interpolační vyhledávání**: Speciální případ binárního hledání, využívá rozložení hodnot v poli (vhodné pro rovnoměrně rozložená data).

# Základní numerické algoritmy

**Numerická derivace**:

- Přibližuje derivaci v bodě pomocí tzv. konečných diferencí, například:
  - Vpřední diference: (f(x+h) - f(x)) / h
  - Vzadní diference: (f(x) - f(x-h)) / h

**Numerická integrace**:

- **Metoda lichoběžníků**: Plocha pod křivkou je aproximována lichoběžníky.
- **Metoda obdélníků**: Interval je rozdělen na malé obdélníky, jejichž součet dává přibližnou hodnotu integrálu.

**Aproximace metodou nejmenších čtverců**:

- Slouží k aproximaci (nejčastěji lineární) vztahu mezi proměnnými tak, aby součet druhých mocnin rozdílů mezi skutečnými a vypočtenými hodnotami byl minimální. Používá se v regresi a analýze dat.