

Úvod do programování

Obsah

1 Úvod	1
1.1 Co je to programování	1
1.2 Proč se učit programovat	1
1.3 Je programování pro každého	1
2 Zaklady algoritmizace	2
2.1 Co je to algoritmus	2
2.2 Vlastnosti algoritmu	2
2.3 Kontext/vnitřní stav algoritmu	2
2.4 Řídící konstrukce algoritmu	2
2.4.1 Sekvence	2
2.4.2 Selekcce	3
2.4.3 Iterace	3
2.5 Algoritmizace problému	4
2.5.1 Kroky algoritmizace	4
2.5.2 Dekompozice a zobecnění problému	4
3 Základy jazyka C	6
3.1 Příprava prostředí pro vývoj v jazyce C	6
3.1.1 MS Windows	6
3.2 Syntaxe jazyka C	9
3.2.1 Základní struktura projektu v jazyce C	9

1 Úvod

1.1 Co je to programování

Programování je proces, při kterém programátor píše zdrojový kód v programovacím jazyce, aby vytvořil instrukce, které počítač vykoná. Zdrojový kód je textový zápis programu, který obsahuje přesné kroky k řešení určitého úkolu. Programovací jazyk je prostředek, kterým programátor komunikuje s počítačem, a překládá své myšlenky do formy, které počítač rozumí. Programátor je člověk, který tyto instrukce tvoří a tím dává počítači schopnost vykonávat různé činnosti.

Programování je tvůrčí činnost, která umožňuje přetvořit nápady na realitu pomocí počítače. Díky tomu můžete automatizovat nudné činnosti, analyzovat velké množství dat, vytvářet užitečné aplikace nebo třeba i hry.

1.2 Proč se učit programovat

V dnešním digitálním světě je programování dovedností, která otevírá dveře k mnoha příležitostem, ať už profesním, nebo osobním. Možná si myslíte, že programování je určeno pouze pro IT odborníky, ale ve skutečnosti má široké využití v každodenním životě.

- **Schopnost řešit problémy** - Programování vás naučí logicky myslet a rozdělit složité problémy na menší, snadněji řešitelné části. Tato dovednost se hodí nejen při práci na počítači, ale i v běžném životě.
- **Zábava a kreativita** - Programování může být zábavné! Například vytvoření vlastní webové stránky, jednoduché hry nebo aplikace vám dá pocit uspokojení, když váš nápad ožije na obrazovce a dělá přesně to co chcete.
- **Příležitosti na pracovním trhu** - Dovednost programování je vysoce ceněná na pracovním trhu. I základní znalosti mohou být velkou výhodou, protože mnoho firem dnes hledá zaměstnance, kteří rozumí technologiím.

1.3 Je programování pro každého

Určitě ano! Možná jste slyšeli, že programování je obtížné nebo že k němu potřebujete být „matematický génius“. To není pravda. Moderní nástroje a jazyky jsou navrženy tak, aby byly přístupné každému, kdo má chuť učit se něco nového. Navíc začít programovat je dnes jednodušší než kdy dříve - existuje mnoho interaktivních tutoriálů, kurzů a nástrojů, které vás provedou prvními kroky. Nejdůležitější je však ochota zkoušet nové věci, nebát se chyb a učit se z nich.

2 Zaklady algoritmizace

2.1 Co je to algoritmus

Algoritmus je přesně definovaný postup skládající se z konečného počtu jasně popsanych kroků, které vedou k řešení určitého problému.

Příklady algoritmů můžeme najít všude kolem nás:

- **V běžném životě:** Recept na přípravu jídla je algoritmus - obsahuje přesné kroky, jak dosáhnout požadovaného výsledku (například upečení koláče).
- **V programování:** Počítačový algoritmus může vyhledat informace, seřadit data nebo vypočítat matematický problém.

Algoritmus je základem každého programu. Než začneme programovat, je důležité nejprve vymyslet a navrhnout algoritmus, protože právě on definuje, jakým způsobem se problém vyřeší.

2.2 Vlastnosti algoritmu

Algoritmus je složením dat a instrukcí, které říkají co se s těmito daty dělá. Každý algoritmus musí splňovat několik klíčových vlastností, aby byl považován za správný a funkční.

- **Vstupní bod** - Každý algoritmus musí mít jeden přesně definovaný vstupní bod, aby bylo jasné kde algoritmus začíná, tedy jakou instrukcí začít.
- **Výstupní bod** - Algoritmus musí mít minimálně jeden, ale i více výstupních bodů, tedy stavů kdy algoritmus již nepokračuje ve své činnosti. Algoritmus totiž může skončit různými způsoby, například úspěšně a nebo neúspěšně.
- **Konečnost** - Algoritmus musí vždy skončit po konečném počtu kroků. Nemůže pokračovat nekonečně, jinak by problém nevyřešil.
- **Vstup** - Algoritmus přijímá vstupní data, která zpracovává. Kdyby do algoritmu nebyly vloženy žádná data, algoritmus by neměl s čím pracovat.
- **Výstup** - Výsledkem algoritmu musí být jednoznačný výstup, který odpovídá zadanému problému. Tento výstup je závislý na vstupu. Kdyby výstupem algoritmu nebyl žádný výsledek, algoritmus by nedělal nic užitečného a byl by zbytečný.

2.3 Kontext/vnitřní stav algoritmu

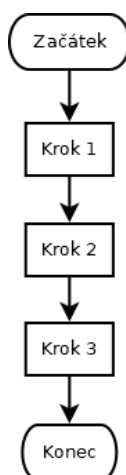
Algoritmus při svém průchodu vytváří tzv. **kontext** nebo také **vnitřní stav**. V reálném světě například v případě kuchařského receptu je kontext algoritmu stav v jakém se nachází vařené jídlo. V případě počítačového programu je kontext algoritmu tvořen hodnotami uloženými v paměti RAM. V průběhu vykonávání algoritmu se tento vnitřní stav algoritmu mění a je na něj možné reagovat pomocí **řídících konstrukcí**.

2.4 Řídící konstrukce algoritmu

Každý algoritmus je tvořen kombinací tří základních konstrukcí, které určují, jak jsou jednotlivé kroky algoritmu prováděny.

2.4.1 Sekvence

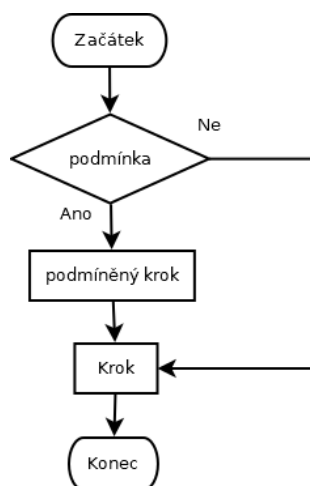
Sekvence je definovaná jako posloupnost kroků, pracující s daty, které v přesně stanoveném pořadí vedou k řešení problému.



Obrázek 2.4.1 Diagram sekvence

2.4.2 Selekcce

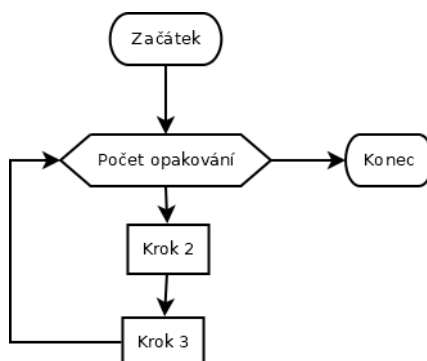
Selekcce umožňuje rozhodování na základě podmínky, která část algoritmu se vykoná. Díky tomu lze vytvořit flexibilní chování, které reaguje na aktuální vnitřní stav algoritmu.



Obrázek 2.4.2 Diagram selekcce (větvení)

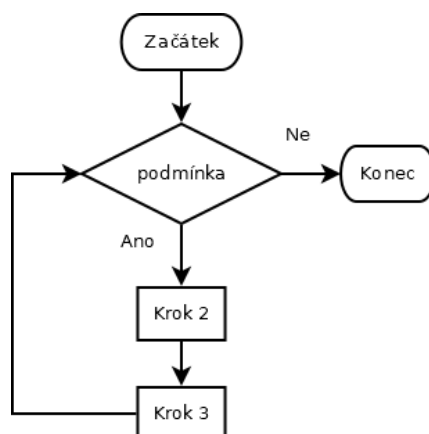
2.4.3 Iterace

Iterace, nebo také cyklus nebo smyčka umožňuje opakované vykonávání určité sekvence kódu, na základě platnosti nějaké podmínky. Je možné rozlišit dva případy iterace. Iterace s pevným počtem opakování se používá v případech kdy předem víme kolikrát je třeba vykonat určitou sekvenci kódu.



Obrázek 2.4.3 Diagram iterace s pevným počtem opakování

Podmíněná iterace je použita ve chvíli kdy je třeba vícekrát vykonat určitou sekvenci kódu, ale není předem možné určit kolikrát to bude třeba.



Obrázek 2.4.4 Diagram podmíněné iterace

2.5 Algoritmizace problému

Algoritmizace je proces vytváření algoritmu - tedy přesného a efektivního postupu, který vede k vyřešení konkrétního problému. Tento proces zahrnuje analýzu problému, návrh a testování algoritmu, který tento problém efektivně vyřeší.

2.5.1 Kroky algoritmizace

- **Porozumění problému** - Prvním krokem je důkladné porozumění tomu, jeho podstatu a co vlastně problém vyžaduje. To zahrnuje definování vstupních a výstupních dat a specifikaci požadavků.
- **Analýza problému** - V tomto kroku se zaměřujeme na rozložení problému na menší části a pochopení, jak jednotlivé části souvisejí. Identifikujeme podproblémy a zjistíme, jaký přístup bude nejvhodnější pro jejich řešení.
- **Návrh algoritmu** - Vytvoříme plán, jakým způsobem problém vyřešit. Zvolíme vhodné kroky a struktury, které pomohou dosáhnout správného výsledku. Tento krok se často realizuje pomocí pseudokódu, diagramů nebo popisu v přirozeném jazyce.
- **Implementace algoritmu** - Jakmile máme dobře navržený algoritmus, přistoupíme k jeho implementaci/realizaci v konkrétním programovacím jazyce. Tento krok zahrnuje přenos algoritmu do kódu, který bude vykonávat počítač.
- **Testování implementace** - Po napsání kódu algoritmus testujeme, abychom ověřili, že implementace funguje podle očekávání, bez chyb a že vykonává algoritmus efektivně.

2.5.2 Dekompozice a zobecnění problému

V praxi je složité a nepraktické řešit složité problémy jako celek, jednodušší a přehlednější je použít dekompozici a využít abstrahování informací k zobecnění problému. Rozložení složitějších problémů na jednodušší je jednou z nejdůležitějších technik v algoritmizaci. Tento proces, známý také jako dekompozice, spočívá v rozdělení velkého problému na menší, lépe zvládnutelné podproblémy, které jsou jednodušší k pochopení a řešení. Každý z těchto podproblémů je řešen samostatně, což následně vede k řešení celkového problému.

Zobecnění řešení znamená nalezení obecného postupu nebo vzoru, který lze aplikovat na více různých problémů, nejen na jeden konkrétní. Tato technika je velmi užitečná, protože umožňuje vytvářet flexibilní a univerzální algoritmy, které mohou být použity na širokou škálu problémů. Zobecnění často znamená zjednodušení problému tak, že se

zaměříme na jeho základní vlastnosti a ignorujeme detaily, které se mohou měnit mezi jednotlivými případy. Díky tomu vytvoříme algoritmy, které nejsou závislé na konkrétním zadání, ale jsou aplikovatelné na širokou paletu problémů.

3 Základy jazyka C

Jazyk C je jedním z nejstarších a nejvlivnějších programovacích jazyků, který byl vyvinut v 70. letech 20. století v Bellových laboratořích. Tento jazyk se stal základem pro mnoho moderních programovacích jazyků, jako je C++, JavaScript a dokonce i některé jazyky vyšších úrovní jako Java a Python.

Vlastnosti jazyka C:

- **Nízká úroveň abstrakce** - Jazyk C poskytuje abstrakci na použitém hardwarem (na úrovni jazyka programátora nezajímá jaký procesor programuje), ale zároveň umožňuje přímý přístup k paměti.
- **Vysoká optimalizace** - Díky své jednoduchosti je možné využít vysokou míru optimalizace výsledného programu. Díky tomu poskytují výsledné programy vysoký výkon.
- **Jednoduchost a přímočarost** - Syntaxe jazyka C je jednoduchá a přímočará (v porovnání s jazyky vyšší úrovně).

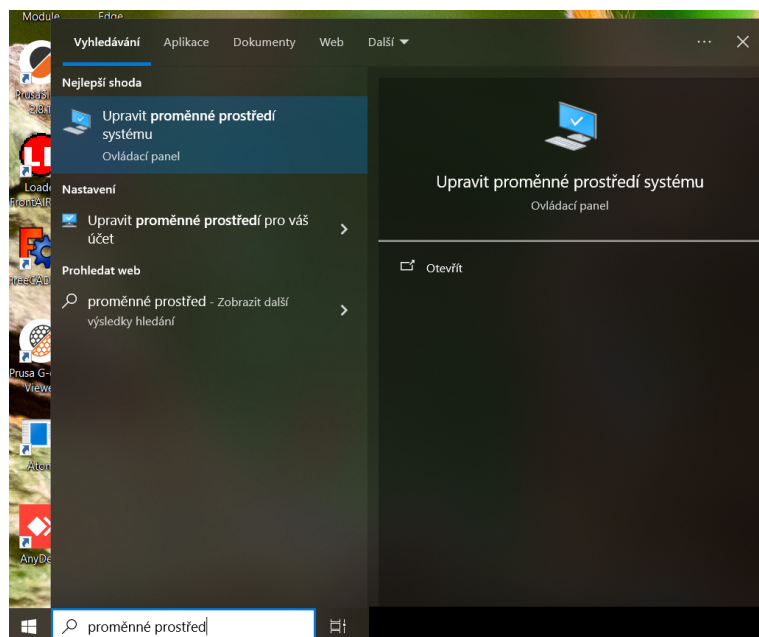
Použití jazyka C:

- **Systémové programování** - C je široce používán pro vývoj operačních systémů (například UNIX), ovladačů zařízení, vestavěných systémů a dalších nízkoúrovňových aplikací, kde je vyžadován přímý přístup k hardwaru.
- **Aplikace s vysokým výkonem** - Vzhledem k tomu, že C poskytuje velkou kontrolu nad výkonem a pamětí, je ideální pro vývoj aplikací, kde jsou kladeny vysoké nároky na výkon, jako jsou grafické programy, hry, simulace nebo vědecké výpočty.
- **Embedded systémy** - C je jazykem první volby pro vývoj embedded (vestavěných) systémů, kde je efektivní využívání paměti a výkonu klíčové. Příkladem vestavěných systémů je například chytrý domácí spotřebič (robotický vysavač, pračka, ...)

3.1 Příprava prostředí pro vývoj v jazyce C

3.1.1 MS Windows

Na systémech MS Windows je nejjednodušší způsob pro nastavení prostředí pro vývoj v jazyce C použít program **msys2** (<https://www.msys2.org/>). Tento program vytvoří strukturu nástrojů, které jsou k dispozici na systémech Linux. Po instalaci je třeba nejprve třeba nastavit systém, aby mohl vyhledat nástroje, které se do počítače nainstalovali. K tomu je nutné upravit **proměnnou prostředí PATH**, do které je třeba přidat dvě nové cesty. Proměnné prostředí je možné upravovat zadáním příkazu do nabýdky start: *upravit proměnné prostředí systému*



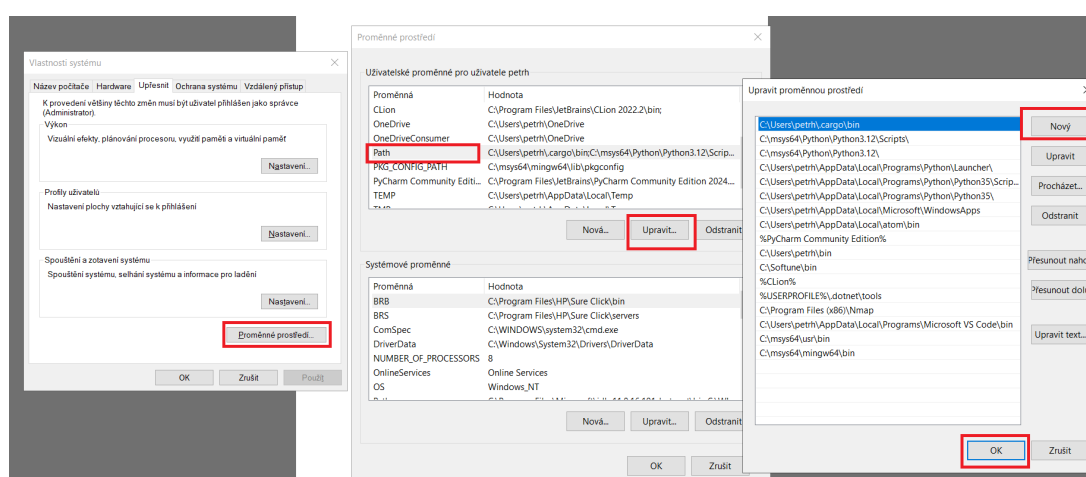
Obrázek 3.1.1 Nastavení proměnné prostředí Path

Poté se otevře okno pro nastavení proměnných prostředí ve kterém je tlačítko **Proměnné prostředí**, které otevře okno s proměnnými prostředí, které jsou v systému vytvořené. V sekce **Uživatelské proměnné** se nachází výčet proměnných prostředí pro aktuálně přihlášeného uživatele a mezi nimi se nachází proměnné **Path**. Po jejím označení je třeba stisknout tlačítko upravit. Následně by se mělo otevřít okno pro úpravu hodnot v této proměnné. Nyní je nutné přidat dvě nové cesty pomocí tlačítka **Nový** a nastavit jejich hodnotu na:

`C:\msys2\usr\bin`

`C:\msys2\mingw64\bin`

Následně stačí jen vše uložit stiskem tlačítka OK a cesty k nástrojům systému msys2 by měly být nastavené.



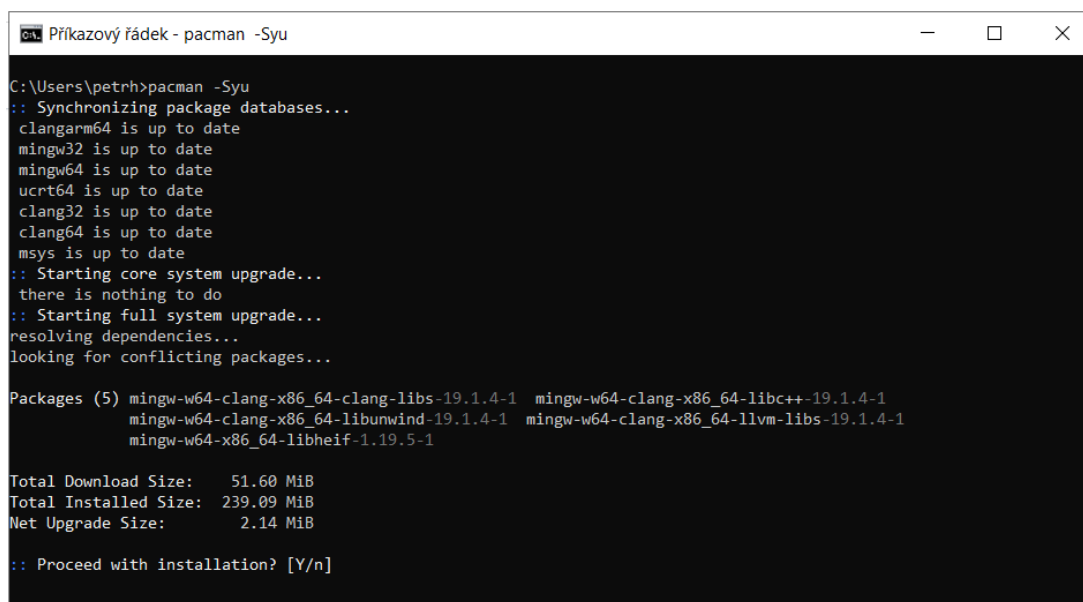
Obrázek 3.1.2 Okno pro nastavení proměnných prostředí

Nově nastavené cesty je možné věřit otevřením příkazové řádky pomocí zadání příkazu `cmd` do nabídky start a zadat příkaz, který by měl provést aktualizaci systému msys2:

`$ pacman -Syu`

(Prosím nekopírujte příkaz se znakem \$, ten značí že se jedná o terminálový vstup)

V případě, že jsou cesty správně nastavené by se měl spustit balíčkový správce *pacman*, který slouží k instalaci a aktualizaci programů do prostředí *msys2* a požádá vás o potvrzení spuštění instalace aktualizací:



```
Příkazový řádek - pacman -Syu

C:\Users\petrh>pacman -Syu
:: Synchronizing package databases...
clangarm64 is up to date
mingw32 is up to date
mingw64 is up to date
ucrt64 is up to date
clang32 is up to date
clang64 is up to date
msys is up to date
:: Starting core system upgrade...
there is nothing to do
:: Starting full system upgrade...
resolving dependencies...
looking for conflicting packages...

Packages (5) mingw-w64-clang-x86_64-clang-libs-19.1.4-1  mingw-w64-clang-x86_64-libc++-19.1.4-1
             mingw-w64-clang-x86_64-libunwind-19.1.4-1  mingw-w64-clang-x86_64-llvm-libs-19.1.4-1
             mingw-w64-x86_64-libheif-1.19.5-1

Total Download Size:    51.60 MiB
Total Installed Size:  239.09 MiB
Net Upgrade Size:       2.14 MiB

:: Proceed with installation? [Y/n]
```

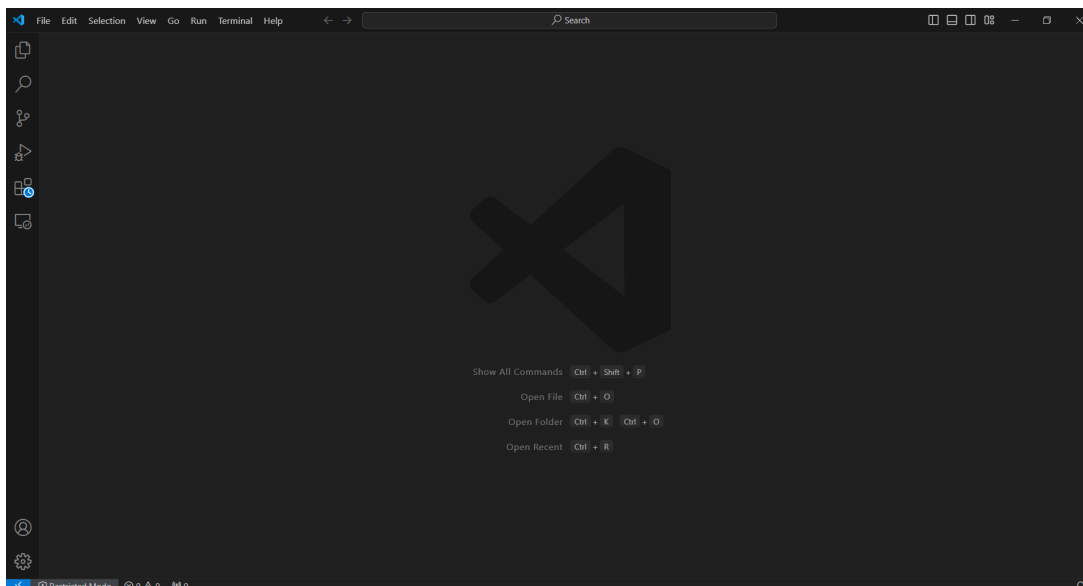
Obrázek 3.1.3 Okno pro nastavení proměnných prostředí

Aktualizace a jiné instalace se potvrdí prostým stiskem klávesy *Enter*.

Po aktualizaci prostředí *msys2* je potřeba nainstalovat kompilátor pro jazyk C **gcc**, který má za úkol převést zdrojový kód na spustitelný a nástroj pro automatizaci překladu **make**. To se provede zadáním příkazu:

```
$ pacman -S gcc make
```

Po této instalaci je prostředí pro vývoj programů v jazyce C na systému MS Windows připravené k použití, ale je potřeba nainstalovat tzv. IDE (Integrated development environment), zjednodušeně řečeno textový editor, který obsahuje nástroje pro zjednodušení psaní zdrojových kódů. Nejjednodušší IDE je **Visual studio code** (<https://code.visualstudio.com/>). Jeho instalace je jednoduchá a přímočará.



Obrázek 3.1.4 Visual studio code

3.2 Syntaxe jazyka C

Jazyk C je založen na několika základních stavebních blocích, mezi něž patří:

- Proměnné
- Podmínky
- Cykly
- Funkce
- Preprocesor

3.2.1 Základní struktura projektu v jazyce C