Úvod do programování

# Co je to program? Příklady a dělení programů

Program je sada uspořádaných instrukcí vytvořených za účelem provedení konkrétního úkolu. Počítačové programy mají mnoho různých podob. Operační systémy, firmware a aplikace jsou příklady programů nalezených v počítačích.

**Operační systém (OS)** je program, který umožňuje uživatelům komunikovat s počítači. Moderní operační systémy obsahují grafická rozhraní usnadňující interakci uživatele. Příklady OS jsou Windows, Linux, OS X, Apple iOS a Android.

**Firmware** jsou programy vytvořené tak, aby umožnily počítači provést konkrétní úkol. Firmware se často nachází v malých počítačích, jako jsou jednoduché digitální hodinky, moderní zařízení a různé subsystémy, které se nacházejí v moderních autech (např. protismykový brzdový systém (ABS) a airbagy). Kvůli omezené paměti a dalším omezením zdrojů IoT zařízení často spoléhají na firmware. Firmware a funkce OS se často překrývají. V některých případech může firmware fungovat také jako operační systém pro jednodušší počítače.

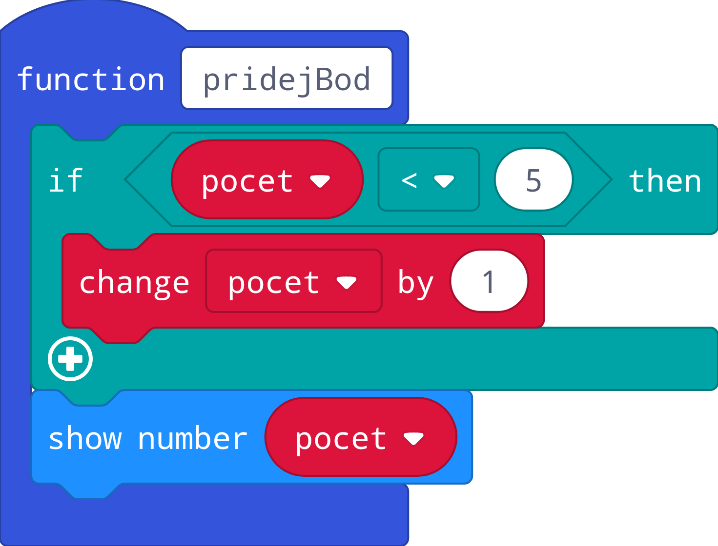
**Aplikace** jsou programy vytvořené za účelem pomoci uživateli při provádění konkrétního úkolu. Příklady aplikací jsou Microsoft Word, Mozilla Firefox a Kalkulačka.

## IF – THEN, FOR, WHILE (příklady)

**IF – THEN** Tato logická struktura umožňuje počítači učinit rozhodnutí na základě výsledku výrazu. Příkladem výrazu je X> 0 . Tento výraz je pravdivý, pokud je hodnota uložená v proměnné X větší než nula. Když interpret nebo kompilátor narazí na strukturu IF-THEN, vyhodnotí poskytnutý výraz. Pokud je výraz nepravdivý, počítač přejde k další struktuře a ignoruje obsah bloku IF-THEN. Pokud je výraz pravdivý, počítač provede související akci, než přejde k další instrukci v programu. (obr.1) 

Obrázek

**FOR** Používají se k provedení specifické sady instrukcí určitým počtem opakování na základě výrazu. Termín smyčka pochází ze skutečnosti, že sada instrukcí se provádí opakovaně. Zatímco syntaxe smyček FOR se jazyk od jazyka liší, koncept zůstává: proměnná je definována tak, aby fungovala jako čítač a jeho maximální hodnota. Při každém provedení smyčky se proměnná čítače zvýší. Když je čítač roven definované maximální hodnotě, smyčka se opustí a provádění se přesune k další instrukci. (obr.2)



Obrázek

**WHILE** Používají se k provedení konkrétní sady instrukcí, když je výraz pravdivý. Všimněte si, že často instrukce uvnitř smyčky nakonec způsobí, že výraz bude vyhodnocen jako nepravdivý. (obr.3)

**Obsah obrázku text

Popis byl vytvořen automaticky**

Obrázek

## Interpretované vs. Kompilované programovací jazyky

**Interpretované jazyky** se při analýze a spuštění kódu spoléhají na jiný program. Tento program se nazývá interpret. Kód zůstává v textu čitelném pro člověka a je předán interpretovi k provedení. Zatímco interpretované jazyky usnadňují údržbu kódu a odstraňování problémů, provádění je také pomalejší než kompilované jazyky. Příklady interpretovaných jazyků jsou Python, JavaScript a PHP.

**Kompilované jazyky** se spoléhají na kompilátor, jiný program, který přemění člověkem čitelný kód na binární spustitelný kód. Zkompilovaný kód je uložen v binárním formátu a lze jej kdykoli spustit bez nutnosti opětovné kompilace. Zkompilovaný kód také běží rychleji, protože je spouštěn přímo CPU, bez potřeby interpreta. Protože kompilovaný jazyk je spouštěn přímo CPU, kompilátor a binární kompilovaný kód, který generuje, jsou specifické pro platformu.  Příklady kompilovaných jazyků jsou C a C++.

# Programovací jazyky (Java Script, Python, C, Java, Blockly)

**JavaScript** je interpretovaný počítačový jazyk jenž je určen pro tvorbu webových aplikací. Pomocí Javascriptu může programátor vytvářet webové aplikace, které mohou komunikovat s uživateli a dalšími aplikacemi.

**Python** je další interpretovaný jazyk, který umožňuje jednodušší příkazy. Python je velmi snadno použitelný, výkonný a všestranný a stal se jazykem volby mnoha vývojářů IoT.

**C** je kompilovaný jazyk, je skvělý pro vytváření složitých a rychlých programů, ale jeho přísná pravidla a syntaxe ztěžují jeho vývoj. C bylo vytvořené na počátku 70. let a se stalo jedním z nejrozšířenějších programovacích jazyků všech dob. Operační systém Linux je napsán v C.

**Java** je kompilovaný jazyk „write once, run anywhere“ (WORA). Navzdory podobnému názvu spolu Java a JavaScript nesouvisí. Java je navržena tak, aby běžela na jakékoli platformě bez nutnosti rekompilace. Java aplikace vyžadují platformu JVM (Java Virtual Machine) nainstalovanou v počítači. JVM je prostředí, kde se spouští kód vyhovující Javě.

**Blockly** je vizuální programovací jazyk, který umožňuje uživatelům vytvářet programy spojováním bloků, které představují různé struktury logického jazyka, spíše než psaním skutečného kódu. Blockly běží ve webovém prohlížeči a dokáže přeložit vizuálně vytvořený program jako JavaScript, PHP nebo Python.

# API, REST API

**Aplikační programové rozhraní (API)** je sada rutin a softwarových nástrojů, které usnadňují komunikaci mezi programy. Díky API mohou aplikace běžící napříč sítěmi mezi sebou komunikovat, sdílet data a žádat o specifické služby od jiných aplikací, jako je dotazování externího API, zda je osoba na obrázku muž nebo žena.

K dispozici jsou různé typy rozhraní API, včetně rozhraní API operačního systému, rozhraní API aplikací a rozhraní API webových stránek. Při vytváření aplikace správce souborů využije vývojář softwaru výhody rozhraní API operačního systému ke zkopírování, přesunutí nebo odstranění souboru. Pokud vývojářská aplikace nemá potřebný kód k provádění těchto funkcí, může jednoduše požádat operační systém, aby je provedl prostřednictvím svého odhaleného rozhraní API operačního systému.

Aplikace IoT mohou mezi sebou komunikovat pomocí rozhraní API, ale mohou také používat rozhraní API pro komunikaci s různými cloudovými službami. Bez ohledu na typ API zůstává cíl stejný; umožnit ostatním programům vzájemnou interakci. Podrobnosti o tom, jak používat konkrétní rozhraní API a jaké úkoly jsou vystaveny, jsou zahrnuty v dokumentaci rozhraní API.

**Representational State Transfer (REST)**, neboli webové služby RESTful, je typ rozhraní API navržený tak, aby usnadnil programům interakci přes internet. REST API používají volání založená na HTTP mezi různými aplikacemi pro přístup a manipulaci s informacemi uloženými ve výkonných databázích.

Webové zdroje bývaly identifikovány výhradně adresou URL. Webové zdroje dnes zahrnují každou entitu nebo věc, kterou lze pojmenovat, identifikovat nebo oslovit. Entity mohou být cíl denního kroku, nastavení teploty z vašeho domu nebo srdeční frekvence z kardiostimulátoru. Každý zdroj má nyní jedinečný identifikátor URI (Uniform Resource Identifier), který obvykle začíná lomítkem, například /heartrate. Rozhraní REST API používají protokol HTTP a buď adresy URL, nebo URI k vyžádání webových služeb. Takové webové požadavky REST spouštějí odpovědi v dobře definovaných webových formátech, jako je XML a JSON (JavaScript Object Notation). Pomocí HTTP jako protokolu si mohou služby RESTful vypůjčit operace HTTP, včetně HTTP sloves GET, POST, PUT a DELETE. Příkladem může být kardiostimulátor, který odesílá aktualizace o vaší průměrné tepové frekvenci do lékařské databáze.

# Bezpečnost kódu

# Doporučené postupy pro zabezpečení připojených zařízení v IoT musí zahrnovat:

a) Zařízení by se měla chránit před útoky, které narušují jejich funkci nebo umožňují jejich neoprávněné použití k nezamýšleným účelům.

b) Zařízení by měla chránit soukromé autentizační údaje a klíčový materiál před prozrazením neoprávněným stranám.

c) Zařízení by měla chránit informace přijaté, přenášené nebo uložené lokálně v zařízení před nevhodným prozrazením neoprávněným stranám.

d) Zařízení by se měla chránit před použitím jako vektor k útoku na jiná zařízení nebo hostitele na internetu.

Aby bylo možné dosáhnout doporučených osvědčených postupů zabezpečení, je fyzické zabezpečení, schopnost důvěřovat integritě kódu a možnost upgradu na dálku zásadní pro návrh všech systémů a zařízení internetu věcí. Zařízení IoT se nacházejí na tradičních místech, jako jsou datová centra v cloudu, a také na mnoha netradičních místech, jako jsou sloupy osvětlení, parkoviště.

**Tradiční místa** lze chránit sledovacími kamerami, systémy kontroly přístupu, biometrickou identifikací a více autentizačními body.)

Zařízení na **netradičních místech** by měla být vyrobena tak, aby byla odolná proti neoprávněné manipulaci, a měla by být umístěna tak, aby nebyla zjevná a byla velmi obtížně přístupná.

## Dalším způsobem ochrany zařízení je:

Zajistit, aby kód na zařízení (jak firmware, tak aplikace) byl původní kód, který byl vytvořen. Toto bezpečnostní opatření může ochránit zařízení před pokusy o změnu jeho funkce, a to i v případě prolomení fyzického zabezpečení. To se provádí kryptografickým „podepsáním“ původního kódu a zajištěním, že se zařízení nespustí, pokud kód není podepsán. Jedná se o stejný přístup používaný k zajištění integrity a původu dokumentů. V oboru se objevuje několik přístupů, jak se vypořádat s různými úhly tohoto problému, jako je Trusted Computing, Cisco Trust Anchor Technologies a Cisco Secure Boot.

## Šifrování dat na cloudu

Data by měla být při příjmu nebo přenosu na místní a vzdálené servery šifrována. Vzdálený přístup k serverům nebo vzdáleným koncovým bodům by měl být také zabezpečen. To zajišťuje, že data si zachovají svou integritu, protože jsou chráněna před neoprávněným přístupem a manipulací. Šifrovací algoritmy a zabezpečené přenosové protokoly jsou techniky běžně používané k zabezpečení softwaru a dat.