GYMNASIUM JANA KEPLERA

Parléřova 2/118, 160 00 Praha 6

Programovací jazyk pro nízká rozlišení

Třída: R8.A

Autor práce: Otakar Dourava

Školní rok: 2020/2021

Předmět: Informatika

Vedoucí práce: Šimon Schierreich

Praha, 2021

GYMNASIUM JANA KEPLERA
Kabinet informatiky

ZADÁNÍ MATURITNÍ PRÁCE

Student:	**Otakar Doubrava**
Třída:	**R8.A**
Školní rok:	**2020/2021**
Platnost zadání:	**30. 9. 2021**
Vedoucí práce:	**Šimon Schierreich**
Název práce:	**Programovací jazyk pro nízká rozlišení**

Pokyny pro vypracování:

Vytvořte programovací jazyk pro účely programování na jednočipovém počítači Rapsberry PI a dalších zařízeních s přístupnými GPIO piny. Jazyk bude možné využít při programování na alfanumerických LCD displejích a dalších podob- ných zobrazovačích s nízkým rozlišením. Součástí jazyka bude jeho kompilá- tor/interpret a řádná dokumentace.

Doporučená literatura:

[1] AHO, Alfred, Monica LAM, Ravi SETHI a Jeffrey ULLMAN. Compilers: Principles, Techniques and Tools. 2. vyd. Boston: Addison Wesley, 2006. ISBN 978- 0321486813.

[2] HALFACREE, Gareth. The Official Raspberry Pi Beginner's Guide: How to use your new computer. 3. vyd. Cambridge: Raspberry Pi Press, 2019. ISBN 9781912047581.

URL repozitáře:

https://github.com/otakardoubrava/gjk_maturitni_projekt

Prohlášení:

Prohlašuji, že jsem svou práci vypracoval samostatně a použil jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů. Nemám žádné námitky proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Praze dne 11. března 2021 Otakar Doubrava

Poděkování

Děkuji Šimonu Schierreichovi za vedení mé maturitní práce, za příhodné rady a vstřícnost při konzultacích práce.

Abstrakt:

Tato práce se věnuje tvorbě programovacího jazyka pro programování GPIO pinů na jednočipovém počítači Raspberry Pi a jiných počítačích s přístupnými GPIO. Cílem tohoto textu je představit průběh tvorby jazyka a jeho fungování.

Klíčová slova

programovací jazyk, GPIO, vývoj

Abstract:

This work deals with the creation of a programming language for programming GPIO pins on a single-chip computer Raspberry Pi and other computers with accessible GPIO. The aim of this text is to present the course of language creation and its functioning.

Key words:

programing language, GPIO, development

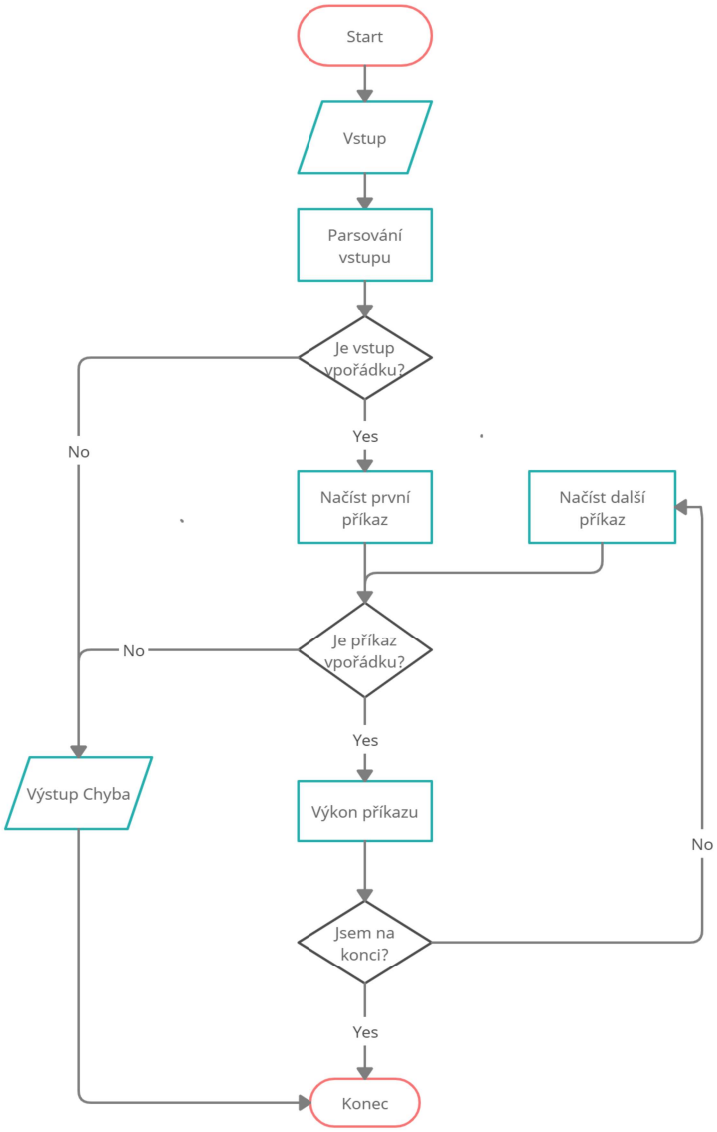
Obsah

1.Teoretická část

Zabýval jsem se tvorbou imperativního programovacího jazyka uzpůsobeného pro práci s GPIO (general purpose input output) piny minipočítače Raspberry Pi a s maximálně úsporným syntaxem jazyka. Jazyk by měl být použitelný při programování na zařízeních, které mohou na výstupu zobrazit pouze velmi omezené množství znaků.

Důvodem bylo, že v tuto chvíli není žádný nástroj, který by toto umožňoval. GPIO je sice možné ovládat pomocí jazyka Python nebo přímo z konsole operačního systému, ale obě z možností mají své nedostatky. Knihovna pro Python je zatím ve velmi začáteční podobě a pro složitější operace je téměř nepoužitelná. Pro ovládání GPIO z konsole je nutné mít přístup k uživateli root (na linuxových systémech administrátorský profil), protože při práci s GPIO přepisujeme obsah systémových souborů. Obě z těchto možností kladou vysoké nároky na množství zobrazených znaků.

Imperativní programování používá pro popis výpočtu sérii příkazů. Výsledkem je přesný algoritmus. Základními příkazy jsou: přiřazení hodnoty do proměnné, příkazy pro větvení a cykly.



Ilustrace 1: zjednodušené schéma fungování interpreteru

Raspberry Pi je jednodeskový minipočítač. Je osazen procesorem rodiny ARM má výkon srovnatelný s pomalejšími stolními počítači. Deska počítače je osazena kontakty pro vstup a výstup, které mohou být individuálně nastaveny. Velikost desky a počet kontaktů závisí na typu danného Pi.

Rozhodl jsem se, že jazyk bude interpretovaný, protože napsat použitelný kompilátor by bylo nad mé síly. Program v interpretovaném jazyce se při každém spuštění předloží interpreteru, který program rozparsuje a následně vyhodnotí a vykoná příkazy.

Syntaktické prvky jazyka jsem přejímal z již existujících jazyků, řádek končí středníkem, obsah cyklů a podmínek je uzavřen do složených závorek,...

2. Implementace

Pro implementaci jsem si zvolil jazyk Python. Co se rychlosti týče, to nebyla úplně nejlepší volba. Lepší by bylo zvolit jazyk C nebo C++, ale s těmi nemám žádnou zkušenost a optimalizace nebyla mým primárním záměrem. Python jsem zvolil také proto, že pro Raspberry Pi je primárním operačním systémen Raspbian (modifikovaný Debian), který, jako všechny ostatní operační systémy založené na linuxovém jádře, má Python nainstalován v základní sadě funkcí. Díky tomu nebude potřeba instalovat žádný doplňující software.

V průběhu celé implementace jsem se soustředil na co největší modularitu každého skriptu a na pojmenování prvků přesně podle toho, co dělají.

V první fázi jsem vytvořil skript, který vyhodnocuje matematické a logické operace a vrací jejich hodnotu. Řešil jsem, jak vyhodnotit libovolně dlouhý výraz bez zbytečně složitého postupu. Nakonec jsem se rozhodl vytvořit třídu pro každý operátor i operand. Skript sestaví z těchto objektů, podle priorit operací objektový strom. Každý objekt každé třídy má metodu *evaluate (vyhodnotit)*, která vrací hodnoty metod *evaluate* podřazených objektů. Po vytvoření stromu tedy stačí zavolat metodu *evaluate* nejvyššího objektu.

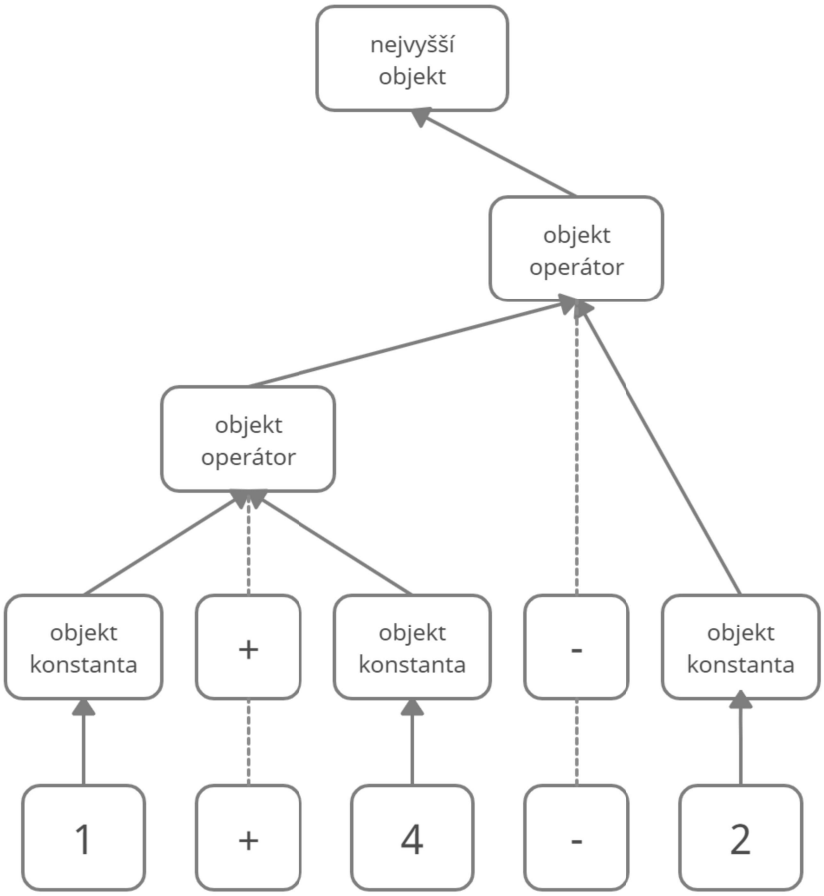
Tímto způsobem je možné přidat další operátor či operand bez nutnosti velikého zásahu do zbytku skriptu.

Dále jsem vytvořil parser, který by jistě šlo nahradit již existujícími nástroji.

Posledním krokem bylo implementovat skript pro výkon jednotlivých parsovaných statementů.

Při implementaci jsem kromě ručního testování použil testovací nástroj unittest. Unittest je Pythonový objektově orientovaný framework pro testování modulů. Obsahuje mnoho funkcí, ale pro mé účely, tedy pro testování, zda funkce vrací správný řetězec, byla nejpoužitelnější funkce *test case. Test case* je testovací jednotkou, zkoumá odpovědi na jisté vstupy. Poskytuje třídu TestCase, která má metody: assertEqual(a, b), assertNotEqual(a, b), assertTrue(x), assertFalse(x), assertIs(a, b,) assertIsNot(a, b), assertIsNone(x), assertIsNotNone(x), assertIn(a, b), assertNotIn(a, b), assertIsInstance(a, b) a assertNotIsInstance(a, b). Pokud test neprojde, unittest vrátí dopodrobna, co nevyšlo.

Pro mě byla nejdůležitější metoda assertEqual(a, b), která testuje jestli *a* je stejné jako *b.* Napsal jsem si testery na většínu funkcí, které pracují se stringy textu. Každý tento tester má vlastní soubor s testovanými stringy a očekávaným výstupem. Do budoucna je tedy velmi jednochuché dodělávat další části modulů a bude stačit vytvořit novou funkci v patřičném testeru a dodělat vstupy a výledky pro testování.



Ilustrace 2: jednoduchý objektový strom

3. Dokumentace

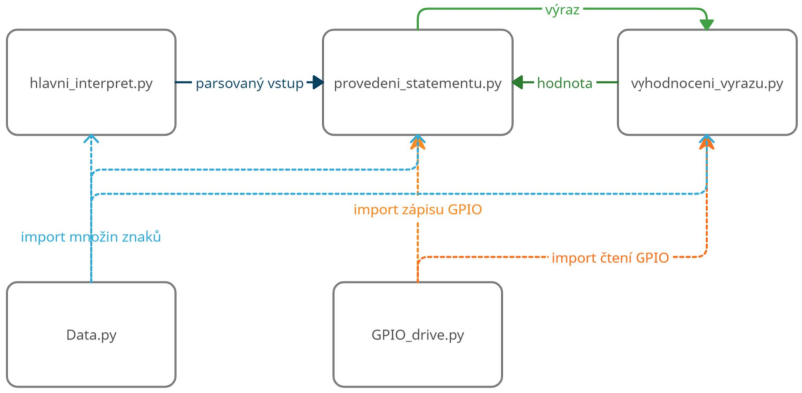
3.1 Princip fungování

Interpret se skládá z pěti modulů: *vyhodnoceni_vyrazu.py*, *hlavni_interpret.py*, *provedeni_statemenetu.py*, *GPIO_drive.py* a *Data.py*.

Jádrem interpreteru je skript *hlavni_interpret.py,* který parsuje vstupní řetězec a ve formě vnořených seznamů ho předává skriptu *provedeni_statemenetu.py*. Ten je zodpovědný za vykonání všech příkazů.

Modul *vyhodnoceni_vyrazu.py* vrací hodnotu matematického nebo logického výrazu. Je volán skriptem *provedeni_statemenetu.py* při nastavování proměnné a při zjišťování platnosti výrazu v podmínce.

GPIO_drive.py a *Data.py* jsou pomocné moduly. *GPIO_drive.py* obsahuje funkce pro čtení a nastavení hodnoty GPIO pinu. *Data.py* obsahuje pouze deklarace množin známých znaků, operátorů a příkazů. Jeho obsah se importuje do všech výkonných skriptů. Tímto způsobem není nutné, aby každý výkonný skript obsahoval všechny tyto množiny separátně.



Ilustrace 3: propojení modulů

Chyby v kódu jsou tištěny do konsole Pythonu se stručnou instrukcí, co je špatně.

3.2 Návod na použití

Interpreter je funkční pouze na systémech Raspberry Pi a je nutné mít všechny skripty ve stejné složce.

Je potřeba se ujistit, že na vašem systému je naistalována knihovna pro ovládání GPIO s názvem RPi.GPIO. Většinou je tato knihovna součástí základní instalace jazyka Python.

1. spusťte skript *hlavni_interpret.py*
2. vložte do konzole váš program jako souvislý řetězec znaků bez odřádkování.

Není třeba dělat nic dalšího.

3.3 Základní syntaxe

označení pojmu	povolená skladba	vysvětlivky
kód	statement ; statement ;	mezera za středníkem není povinná, slouží pouze pro zpřehlednění kódu
statement	deklarace	
	příkaz	
deklarace	jméno = hodnota	
hodnota	číslo	pouze kladná čísla, záporná čísla je třeba deklarovat jako 0-x, u GPIO pouze 0 a 1
	jméno	jméno dříve deklarované proměnné
jméno	text	proměnná s textovým jménem je klasická proměnná
	$ číslo	reprezentuje konkrétní GPIO PIN na desce Rasapberry Pi
číslo		číslo fyzického pinu, více v dokumentaci konkrétního RPi.
text		jakýkoli souvislý text, nesmí obsahovat speciální znaky
speciální znaky	& * / - + < = > | ! ( )	
příkaz	název_příkazu výraz	
název_příkazu	if lp pt whl brk	

výraz	operand operátor operand	jednoduchý výraz
	(výraz) operátor operand	složený výraz (libovolný operand nahrazen výrazem)
operand	jméno	názvy proměnných a GPIO
	číslo	konstanta, záporná čísla je potřeba prioritizovat (dát do závorky)
operátor	< <= != = > >= * / - +	číselné operátory
	| || ! &	logické operátory

3.4 Názvy operátorů

< >	menší/větší než	|	or
<= >=	menší/větší rovná se	||	xor
= !=	rovná se, nerovná se	!	negace
* /	krát, děleno	&	and
+ -	plus, mínus	!&	nand
Všechny operátory mají stejnou prioritu. Násobení a dělení tedy **nemá** přednost před sčítáním a odčítáním.			

3.5 Syntaxe příkazů

pt (print)	pt výraz	tisk do konsole
if (if)	if výraz { kód }	podmínka pokud, když platí výraz, vykoná se kód po posledním statementu kódu ve složených závorkách není třeba psát středník
lp (loop)	lp číslo { kód }	opakuj, číslo určuje počet opakování
	lp jméno ~ číslo { kód }	cyklus opakuj, ukládá počet opakovaní do proměnné od 0 do zadaného čísla
	lp jméno ~ číslo , číslo { kód }	cyklus opakuj, ukládá počet opakovaní do proměnné od zadaného čísla do zadaného čísla první číslo musí být menší než druhé
číslo	jakékoli přirozené číslo	
jméno	jméno proměnné, není nutné předem deklarovat	
whl (while)	whl výraz { kód }	cyklus dukud, opakuje kód, dokud výraz platí
brk (break)	brk číslo	čekej, pozastaví program na čas určený v milisekundách
číslo	jakékoli přirozené číslo	

3.6 Ukázkový kód

$20=0;whl $18!=1{if $19=1{lp 2{$20=1; brk 500; $20=0; brk 500}}}

Nastav GPIO 20 na 0, dokud GPIO 18 není 1 opakuj: pokud GPIO 19 je 1, opakuj dvakrát: nastav GPIO 20 na 1, čekej 500 ms, nastav GPIO 20 na 0, čekej 500 ms.

4. Závěr

Řekl bych, že jsem zadání splnil. Jazyk je sice velmi primitivní, ale splňuje všechna kritéria imperativního jazyka.

Mé provedení má jeden veliký nedostatek. Všechny operece mají stejnou prioritu. Zaprvé je to špatně z hlediska matematických konvencí a zadruhé to trochu popírá cíl celé práce, tedy vytvořit úsporný jazyk, protože nyní jsou potřeba zcela zbytečné závorky.

Do budoucna bych přidal možnost pracovat i s jinými datovými typy než je integer, primárně floating point.

Při vyhodnocování výrazů se každý objekt tvoří znovu. Bylo by lepší, kdyby se třeba objekty proměnných ukládaly, protože je logické předpokládat, že se s danou proměnnou bude pracovat vícekrát. Stejně tak by se mohly ukládat nějaké složitější objekty. Třeba pokud cyklu loop vždy zvedáme hodnotu proměnné *a* o *1*, uložit objekt *a+1*.

Pro parsování vstupního řetězce by se daly použít již existující nástroje, třeba *Flex (Fast Lexical Analyzer Generator).*

Kód interpreteru není nijak optimalizovaný pro rychlost a obsahuje mnoho částí, které by se daly napsat lépe.

Nemyslím si, že by nedokonalost měl být problém. Programovací jazyk většinou nevyvýjí jeden člověk a implementace trvá déle než půl roku. Nikdo asi nečekal, že student maturitního ročníku vytvoří jako maturitní projekt dokonalý, hotový programovací jazyk.

5. Použitá literatura

Python Software Foundation [online]. Poslední změna 7. 4. 2021. Dostupné z: https://docs.python.org/3/

HALFACREE, Gareth. The Official Raspberry Pi Beginner's Guide: How to use your new computer. 3. vyd. Cambridge: Raspberry Pi Press, 2019. ISBN 9781912047581.