GYMNASIUM JANA KEPLERA

Parléˇrova 2/118, 169 00 Praha 6

JOP

Jazyk Otakara Doubravy s úsporným syntaxem pro programování gpio na Rasbberry Pi

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 trvorbě programovacího jazyka pro programování GPIO pinů na jednočipovém počítači Raspberry Pi a jiných počítačích s přístupnámi GPIO. Cílem tohoto textu je předstvit průběh tvorby jazyka a jeho fungování.

Klíčová slova

programovací jazyk, GPIO, vývoj

Obsah

1.Teoretická část

Zabýval jsem se tvorbou programovacího jazyka uzpůsobrného pro práci s GPIO (general purpose input output) piny minipočítače Raspberry Pi a s maximálně úsporným syntaxm 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.

Rozhodl jsem se, že jazyk bude interpretovaný, protože napsat použitelný kompilátor by bylo nad mé síly.

2. Implementace

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, terý by jistě šlo nahradit již existujícími nástroji.
Posledním krokem bylo implementovat skript pro výkon jednotlivých parsovaných statementů. V tuto chvíli jsem již neřešil žádné světoborné problémy.

3. Dokumentace

Princip fungování

Jazyk JOP je interpretovaným jazykem. To znamená, že napsaný program se při každém spuštění předloží interpreteru, který příkazy vyhodnotí a vykoná. Syntaktické prvky jazyka byly často přejaty z již existujících jazyků.

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

Návod na použití:

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

Není třeba dělat nic dalšího.

Syntax

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	
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

Názvy operátorů

< >	menší/větší než	|	or
<= >=	menší/větší rovná se	||	xor
= !=	rovná se, nerovná se	!	negace
* /	krát, děleno	&	and
+ -	Plus, mínus	!&	nand

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

Závěr

Řekl bych, že jsem zadání splnil. Jazyk je sice velmi primitivní, ale splňuje všechna kritéria imperativního jazyka. Interpreter není úplně dokonalý a mnoho by se dalo zlepšit. Vše ale funguje, jak má.

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.