Malý domácí úkol ze cvičení 3

Interpret je program (v našem případě funkce), který na vstupu vezme zdrojový kód programu napsaném v nějakém programovacím jazyce, tento program odsimuluje a na výstup vrátí to, co by vrátil onen program. V tomto úkolu budete implementovat interpret pro jednoduchý programovací jazyk, více si o něm řekneme vzápětí. Kdybyste implementovali interpret Pythonu, programovali byste funkci, která na vstup vezme kód v Pythonu ve formátu textového řetězce a na výstup vrátí to, co by vrátil tento kód. Pro vstup “print(‘ahoj’)” by tedy vaše funkce vytiskla řetězec “ahoj”, pro vstup
“a=2
b=a+1
print(b)”
by vaše funkce na výstup vytiskla řetězec “3”. Vy ale nebudete intepretovat Python, ale jednoduchý jazyk, který se skládá z těchto osmi instrukcí: + - > < . , [ ]
Každá z těchto instrukcí má danou sémantiku, která je popsána níže, vaše funkce tedy na vstupu vezme program, např řetězec “>+[-].”, a odsimuluje běh tohoto programu. Abyste mohli program odsimulovat, budete potřebovat paměť, nad kterou program pracuje (ta bude reprezentována jako pole čísel, jehož hodnoty budete během simulace měnit) a vstup, ktery program čte (ten bude reprezentován jako pole znaků, po němž se během simulace posouváte), pokud během simulace program něco vytiskne, vypíšete to na standartní výstup (tzn vytisknete to pomocí funkce print()).

Implementujte interpret jednoduchého programovacího jazyka. Definujte funkci interpreter(code, textInput) typu (string → string) → void, která bude interpretovat program code nad vstupem textInput. Interpret (a interpret je v tomto případě naše funkce) bude mít následující vnitřní stav:

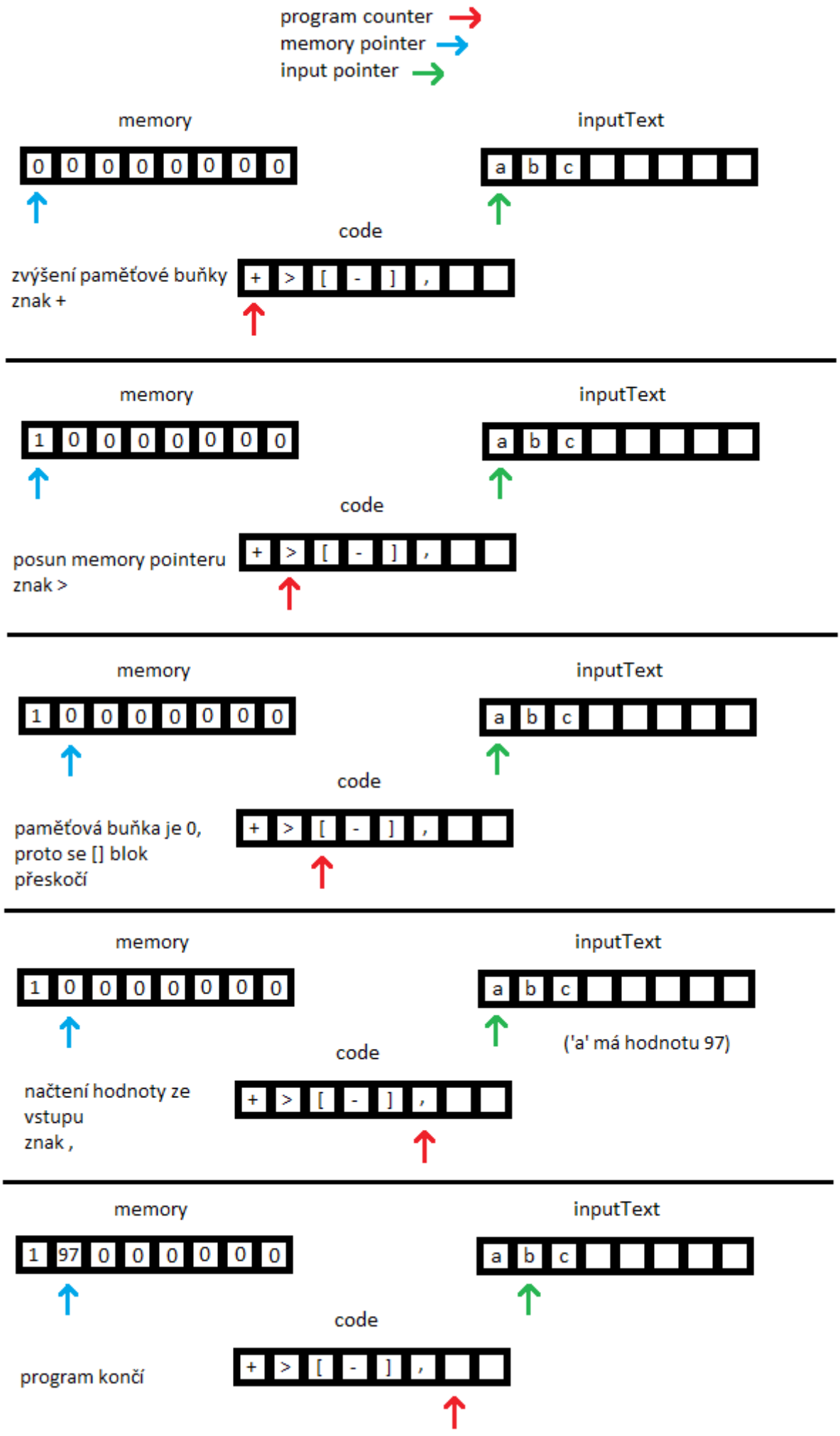
-paměť ([int] délky 30); na začátku mají všechny buňky hodnotu 0
-ukazatel na aktuální buňku paměti (memory pointer) (int), na začátku má hodnotu 0
-ukazatel na aktuální polohu v programu code (program counter) (int), na začátku má hodnotu 0
-ukazatel na aktuální hodnotu na vstupu (input pointer) (int), který představuje textInput, na začátku má hodnotu 0

Intepret bude postupně číst vstupní program code a jeho znaky bude vyhodnocovat s takto:

‘>’ posuň memory pointer o jednu buňku doprava, skoč na další znak programu
‘<’ posuň memory pointer o jednu buňku doleva, skoč na další znak programu
‘+’ zvyš hodnotu, na kterou ukazuje memory pointer o 1, skoč na další znak programu
‘+’ sniž hodnotu, na kterou ukazuje memory pointer o 1, skoč na další znak programu
‘.’ vypiš na výstup hodnotu aktuální buňky paměti jako znak (pomůže funkce chr()), skoč na další znak progamu
‘,’ převeď aktuální hodnotu na vstupu (v našem případě v poli textInput) na číslo (pomůže funkce ord()), ulož ji do aktuální buňky paměti, přesuň ukazatel na další znak na vstupu, skoč na další znak programu
‘]’ posouvej program counter dozadu tak dlouho, dokud nebude ukazovat na znak ‘[‘
‘[‘ pokud je hodnota aktuální buňky 0, posouvej program counter dopředu tak dlouho, dokud nebude ukazovat na první znak za znakem ‘]’, jinak skoč na další znak programu

Předpokládejte, že code žádné jiné znaky než tyto obsahovat nebude. Pro jednoduchost neřešte vnořené cykly (v programu se nikdy nebojeví konstrukce typu ‘[[]]’, vždy to bude pouze ‘[]’).

Na další stránce je ukázka, jak interpret vypadá.


program counter
memory pointer
input pointer
memory
0 0 0 0 0 0 0 0
inputText
a b c
code
+ > [-] ,
zvýšení paměťové buňky
znak +
memory
1 0 0 0 0 0 0 0
inputText
a b c
code
+ > [-] ,
posun memory pointeru
znak >
memory
1 0 0 0 0 0 0 0
inputText
a b c
code
+ > [-] ,
paměťová buňka je 0,
proto se [] blok
přeskočí
memory
1 0 0 0 0 0 0 0
inputText
a b c
('a' má hodnotu 97)
code
+ > [-] ,
načtení hodnoty ze
vstupu
znak ,
memory
1 97 0 0 0 0 0 0
inputText
a b c
code
+ > [-] ,
program končí