Fluxograma - Calculadora HP-12C

Coluna 1 (main)

INÍCIO (início)

Preprocessamento:
```
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define STACK_CAPACITY 4
#define INPUT_CAPACITY 200
```

Definição de Tipos:
```
typedef enum { Ok, Error } Status

typedef enum {
    NumberOk,
    OperatorOk,
    InvalidTokenError,
    StackIsFullError,
    DivideByZeroError
} EvaluateTokenStatus

typedef struct {
    int inner[STACK_CAPACITY]
    int len
} Stack
```

main()

FIM

---

int main()

```
Stack stack = {0};
char input[INPUT_CAPACITY] = {0};
```

"Instruções:
- Digite a expressão na Notação Polonesa Inversa para obter o resultado desta;
- Digite "Q" para encerrar o programa."

Loop infinito

Texto a ser armazenado em input

`char* token = strtok(input, " \n")`

token == NULL → SIM

NÃO ↓

strcmp(token, "Q") == 0 → SIM

NÃO ↓

```
token != NULL;
token = strtok(NULL, " \n")
```

```
EvaluateTokenStatus
status =
evaluateToken(&stack,
token)
```

status

- NumberOk
- OperatorOk → Último valor da pilha
- InvalidTokenError → token " não é um token válido."
- StackIsFullError → "Não foi possível empilhar " token ", e portanto o processo foi interrompido"
- DivideByZeroError → "Não é possível dividir por zero."

"Obrigado por usar nossa Calculadora Fatec-HP12c"

return 0

Coluna 2 (evaluateToken)

```
EvaluateTokenStatus
evaluateToken(Stack* stack,
char* token)
```

`int parsedInt;`

```
int parseStatus
= parseInt(token,
&parsedInt)
```

parseStatus == 0 → SIM → `int pushStatus = stackPush(stack, parsedInt)` → **pushStatus == 0** → SIM → **return NumberOk**

NÃO ↓ (pushStatus) → **return StackIsFullError**

NÃO ↓ (parseStatus)

strlen(token) != 1 → SIM → **return InvalidTokenError**

NÃO ↓

```
int result, value1 = 0,
value2 = 0
```

token[0]

- '+' → `stackPop(stack, &value2)` `stackPop(stack, &value1)` → `result = value1 + value2`
- '-' → `stackPop(stack, &value2)` `stackPop(stack, &value1)` → `result = value1 - value2`
- '*' → `stackPop(stack, &value2)` `stackPop(stack, &value1)` → `result = value1 * value2`
- '/' → `stackPop(stack, &value2)` → **value2 == 0** → SIM → `stackPush(stack, value2)` → **return DivideByZeroError**
  - NÃO ↓ → `stackPop(stack, &value1)` → `result = value1 / value2`
- Outros casos → **return InvalidTokenError**

`stackPush(stack, result)`

return OperatorOk

Coluna 3 (parseInt)

```
Status parseInt
(char* s, int* target)
```

`*target = 0`

```
int i = 0;
s[i] != '\0';
i++
```

s[i] < '0' ou s[i] > '9' → SIM → **return Error**

NÃO ↓

```
*target = *target * 10
+ (s[i] - '0')
```

return Ok

Coluna 4 (stackPush)

```
Status stackPush(Stack*
stack, int value)
```

stack->len >= STACK_CAPACITY → SIM → **return Error**

NÃO ↓

```
stack->inner[stack->len] = value
stack->len++
```

return Ok

Coluna 5 (stackPop)

```
Status stackPop(Stack*
stack, int* target)
```

stack->len <= 0 → SIM → **return Error**

NÃO ↓

```
stack->len--
*target = stack->inner[stack->len]
```

return Ok

Disciplina: Programação Estruturada e Modular
Prof. Carlos Veríssimo

Objetivo: Diagramação do Código Calculadora HP-12C
Data: 20/09/2024
Autor: Vinicius Leon Melo Tavares