**Imagem digital fictícia de personagem de desenho animado

Descrição gerada automaticamente com confiança média**

**UNIVERSIDADE ESTADUAL DE SANTA CRUZ - UESC**

**LUIZ AUGUSTO BELLO MARQUES DOS ANJOS**

**RELATORIO PARA TRABALHO PROJ1B PARA A DISCIPLINA CET087 – CONCEITOS DE LINGUAGEM DE PROGRAMAÇÃO**

**ILHÉUS – BAHIA**

**2024**

**SUMÁRIO:**

1. **INTRODUÇÃO**
2. **INTERPRETADOR P-CODE EM C**
   1. **Explicação do código**
   2. **Código fonte**
   3. **Linhas de comando para compilar e executar**
3. **QUESTÕES RESOLVIDAS**
   1. **Questão 1**
   2. **Questão 2**
   3. **Questão 3**
4. **SAIDAS DAS QUESTÕES**
   1. **1**
   2. **2**
   3. **3**
5. **LINKS PARA DOWNLOAD**
6. **REFERÊNCIAS**
7. **INTRODUÇÃO:**

Neste relatório, compartilho os detalhes da minha implementação de uma máquina de p-código em linguagem C. A máquina de p-código é uma construção fundamental em ciência da computação, e esta implementação foi inspirada nas ideias de Niklaus Wirth, conforme apresentadas em seu influente livro "Algorithms + Data Structures = Programs", publicado em 1976.

O objetivo principal deste projeto foi desenvolver um simulador da máquina de p-código, capaz de interpretar um conjunto limitado de instruções e executá-las de forma eficiente. Esta implementação envolveu a criação de um ambiente simulado que incluiu registradores específicos e uma pilha de dados, conforme especificado por Wirth.

Ao longo deste relatório, descrevo o processo de desenvolvimento, desde a concepção até a implementação final do código em C. Exploro em detalhes as instruções suportadas, a estrutura do código, bem como os desafios enfrentados durante o desenvolvimento.

Em resumo, o código implementa um simulador de máquina de p-código em C, capaz de ler um conjunto de instruções de um arquivo de entrada, interpretá-las e executá-las, registrando em um arquivo de saída o estado interno da máquina em cada passo da execução. Isso permite uma análise detalhada do comportamento do programa durante a execução.

**2.a INTERPRETADOR P-CODE EM C:**

Este código implementa uma máquina de p-código em linguagem C, inspirada na máquina de p-código original proposta por Niklaus Wirth em Pascal. A máquina de p-código é uma representação simplificada de um ambiente de execução de programas, composta por um conjunto limitado de instruções e registradores.

**Modelo de Entrada:**

O programa lê as instruções de um arquivo de texto chamado "instrucoes.txt". Cada linha deste arquivo contém uma instrução no formato “op arg1 arg2”, onde “op” é o código da operação, "argl" é um parâmetro para o nível lexical , e "arg2" é outro parâmetro que varia dependendo da operação, podendo ser um valor inteiro, um endereço do programa, a identidade de um operador etc.

**Modelo de Saída:**

A saída do programa é registrada no arquivo "resposta.txt". Cada linha deste arquivo representa o estado interno da máquina após a execução de uma instrução, incluindo os valores dos registradores p, b, t e o conteúdo da pilha de dados.

**Funcionamento do Código:**

* 1. **Definições e Estruturas de Dados:**
     1. O código define algumas constantes, como o tamanho máximo da pilha (stacksize), o tamanho máximo da tabela de código (cxmax), e os códigos de operação (fct).
     2. Define também uma estrutura de dados “instrucao”, que representa uma instrução com seu código de operação (f), nível léxico (l), e argumento (a).
  2. **Função parse\_instrucao:** 
     1. Esta função analisa uma linha do arquivo de entrada e extrai os componentes da instrução, atribuindo-os à estrutura "instrucao”.
  3. **Função interpret:** 
     1. Esta é a principal função do programa, responsável por interpretar e executar as instruções lidas do arquivo.
     2. Utiliza um loop para percorrer todas as instruções lidas do arquivo.
     3. A cada iteração, a função verifica o código de operação da instrução e executa a operação correspondente.
     4. Mantém registros de estado (ponteiro de instrução p, base b, topo da pilha t) e manipula a pilha de dados de acordo com as instruções.
     5. A saída dessa função é direcionada para um arquivo de texto chamado "resposta.txt", registrando o estado interno da máquina após cada instrução.
  4. **Função main:**
     1. Esta função coordena a leitura das instruções do arquivo de entrada, chamando parse\_instrucao para processá-las e armazená-las em um array de instrucao.
     2. Em seguida, chama a função interpret para executar as instruções.
     3. Por fim, fecha os arquivos abertos e encerra o programa.

**2.b CÓDIGO DO INTERPRETADOR EM C:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define amax 2047       /\* maximum address \*/

#define levmax 3        /\* maximum depth of block nesting \*/

#define cxmax 200       /\* size of code array \*/

typedef enum {lit, opr, lod, sto, cal, inter, jmp, jpc} *fct*;

typedef struct {

*fct* f;

    int l;

    int a;

    char name[5]; // Aumento do tamanho para acomodar todos os nomes de instruções

} *instrucao*;

*instrucao* code[cxmax];

void interpret(*FILE* \**output*, int *instrucao\_count*) {

    const int stacksize = 500;

    int p, b, t;

*instrucao* i;

    int s[stacksize];

    p = 0; b = 1; t = 0;

    memset(s, 0, sizeof(int) \* stacksize);

    for (int index = 0; index < *instrucao\_count*; index++) {

        i = code[index];

        switch (i.f) {

            case lit:

                s[++t] = i.a;

                fprintf(*output*, "%s %d %d %d ", i.name, p, b, t);

                for (int j = 1; j <= t; j++) {

                    fprintf(*output*, "%d ", s[j]);

                }

                fprintf(*output*, "\n");

                break;

            case opr:

                switch (i.a) {

                    case 0: /\* return \*/

                        t = b - 1;

                        p = s[t + 2];

                        b = s[t + 1];

                        break;

                    case 1:

                        s[t] = -s[t];

                        break;

                    case 2:

                        t--;

                        s[t] = s[t] + s[t + 1];

                        break;

                    case 3:

                        t--;

                        s[t] = s[t] - s[t + 1];

                        break;

                    case 4:

                        t--;

                        s[t] = s[t] \* s[t + 1];

                        break;

                    case 5:

                        t--;

                        s[t] = s[t] / s[t + 1];

                        break;

                    case 6:

                        s[t] = (s[t] % 2);

                        break;

                    case 8:

                        t--;

                        s[t] = (s[t] == s[t + 1]);

                        break;

                    case 9:

                        t--;

                        s[t] = (s[t] != s[t + 1]);

                        break;

                    case 10:

                        t--;

                        s[t] = (s[t] < s[t + 1]);

                        break;

                    case 11:

                        t--;

                        s[t] = (s[t] >= s[t + 1]);

                        break;

                    case 12:

                        t--;

                        s[t] = (s[t] > s[t + 1]);

                        break;

                    case 13:

                        t--;

                        s[t] = (s[t] <= s[t + 1]);

                        break;

                }

                fprintf(*output*, "%s %d %d %d ", i.name, p, b, t);

                for (int j = 1; j <= t; j++) {

                    fprintf(*output*, "%d ", s[j]);

                }

                fprintf(*output*, "\n");

                break;

            case lod:

                s[++t] = s[b + i.a];

                fprintf(*output*, "%s %d %d %d ", i.name, p, b, t);

                for (int j = 1; j <= t; j++) {

                    fprintf(*output*, "%d ", s[j]);

                }

                fprintf(*output*, "\n");

                break;

            case sto:

                s[b + i.a] = s[t];

                fprintf(*output*, "%s %d %d %d ", i.name, p, b, t);

                for (int j = 1; j <= t; j++) {

                    fprintf(*output*, "%d ", s[j]);

                }

                fprintf(*output*, "\n");

                t--;

                break;

            case cal:

                s[t + 1] = b;

                s[t + 2] = p;

                s[t + 3] = i.a;

                b = t + 1;

                p = i.a;

                break;

            case inter:

                t += i.a;

                fprintf(*output*, "%s %d %d %d ", i.name, p, b, t);

                for (int j = 1; j <= t; j++) {

                    fprintf(*output*, "%d ", s[j]);

                }

                fprintf(*output*, "\n");

                break;

            case jmp:

                p = i.a;

                fprintf(*output*, "%s %d %d %d ", i.name, p, b, t);

                for (int j = 1; j <= t; j++) {

                    fprintf(*output*, "%d ", s[j]);

                }

                fprintf(*output*, "\n");

                break;

            case jpc:

                if (s[t] == 0)

                    p = i.a;

                t--;

                fprintf(*output*, "%s %d %d %d ", i.name, p, b, t);

                for (int j = 1; j <= t; j++) {

                    fprintf(*output*, "%d ", s[j]);

                }

                fprintf(*output*, "\n");

                break;

            default:

                printf("Invalid operation code.\n");

                break;

        }

    }

}

void parse\_instrucao(const char \**line*, *instrucao* \**ins*) {

    char op[5]; // Aumento do tamanho para acomodar todos os nomes de instruções

    sscanf(*line*, "%s %d %d", op, &(*ins*->l), &(*ins*->a));

    if (strcmp(op, "lit") == 0) {

*ins*->f = lit;

        strcpy(*ins*->name, "lit"); // Atribui o nome da instrução

    } else if (strcmp(op, "opr") == 0) {

*ins*->f = opr;

        strcpy(*ins*->name, "opr"); // Atribui o nome da instrução

    } else if (strcmp(op, "lod") == 0) {

*ins*->f = lod;

        strcpy(*ins*->name, "lod"); // Atribui o nome da instrução

    } else if (strcmp(op, "sto") == 0) {

*ins*->f = sto;

        strcpy(*ins*->name, "sto"); // Atribui o nome da instrução

    } else if (strcmp(op, "cal") == 0) {

*ins*->f = cal;

        strcpy(*ins*->name, "cal"); // Atribui o nome da instrução

    } else if (strcmp(op, "int") == 0) {

*ins*->f = inter;

        strcpy(*ins*->name, "int"); // Atribui o nome da instrução

    } else if (strcmp(op, "jmp") == 0) {

*ins*->f = jmp;

        strcpy(*ins*->name, "jmp"); // Atribui o nome da instrução

    } else if (strcmp(op, "jpc") == 0) {

*ins*->f = jpc;

        strcpy(*ins*->name, "jpc"); // Atribui o nome da instrução

    } else {

*ins*->f = -1; // indica que a instrucao e invalida

    }

}

int main() {

    char line[20];

    int index = 0;

*FILE* \*input\_file = fopen("instrucoes.txt", "r");

*FILE* \*output\_file = fopen("resposta.txt", "w");

    int instrucao\_count = 0; // Variável para contar o número de instrucoes lidas

    if (input\_file == NULL || output\_file == NULL) {

        printf("Error opening files.\n");

        return 1;

    }

    while (index < cxmax && fgets(line, sizeof(line), input\_file)) {

        parse\_instrucao(line, &code[index]);

        index++;

        instrucao\_count++; // Incrementa o número de instrucoes lidas

    }

    interpret(output\_file, instrucao\_count); // Passa o número de instrucoes lidas para a função interpret

    fclose(input\_file);

    fclose(output\_file);

    return 0;

}

**2.c Código para compilação e execução no terminal:**

- gcc -o Interpretador\_C Interpretador\_C.c

- ./Interpretador\_C.c

- Get-Content questao1.txt | Set-Content instrucoes.txt (para executar Questão 1)

- Get-Content questao2.txt | Set-Content instrucoes.txt (para executar Questão 2)

- Get-Content questao3.txt | Set-Content instrucoes.txt (para executar Questão 3)

1. **QUESTÕES RESOLVIDAS:**
   1. Soma dos números naturais de 1 até 10:

| **Op** | **Arg1** | **Arg2** |
| --- | --- | --- |
| lit | 0 | 1 |
| sto | 0 | 0 |
| lit | 0 | 10 |
| sto | 0 | 1 |
| lit | 0 | 0 |
| sto | 0 | 2 |
| lod | 0 | 2 |
| lod | 0 | 0 |
| opr | 0 | 2 |
| sto | 0 | 2 |
| lod | 0 | 0 |
| lit | 0 | 1 |
| opr | 0 | 2 |
| sto | 0 | 0 |
| lod | 0 | 0 |
| lod | 0 | 1 |
| opr | 0 | 10 |
| jpc | 0 | 22 |
| jmp | 0 | 7 |

**3.2** Soma dos quadrados dos números naturais de 1 até 100.

| **Op** | **arg1** | **arg2** |
| --- | --- | --- |
| lit | 0 | 1 |
| sto | 0 | 0 |
| lit | 0 | 100 |
| sto | 0 | 1 |
| lit | 0 | 0 |
| sto | 0 | 2 |
| lod | 0 | 2 |
| lod | 0 | 0 |
| opr | 0 | 2 |
| sto | 0 | 2 |
| lod | 0 | 0 |
| lit | 0 | 1 |
| opr | 0 | 2 |
| sto | 0 | 0 |
| lod | 0 | 0 |
| lod | 0 | 1 |
| opr | 0 | 4 |
| opr | 0 | 2 |
| sto | 0 | 2 |
| lod | 0 | 0 |
| lod | 0 | 1 |
| opr | 0 | 10 |
| jpc | 0 | 27 |
| jmp | 0 | 7 |

* 1. Soma dos cubos dos números naturais de 1 até 1000.

| **Op** | **arg1** | **arg2** |
| --- | --- | --- |
| lit | 0 | 1 |
| sto | 0 | 0 |
| lit | 0 | 1000 |
| sto | 0 | 1 |
| lit | 0 | 0 |
| sto | 0 | 2 |
| lod | 0 | 2 |
| lod | 0 | 0 |
| opr | 0 | 2 |
| sto | 0 | 2 |
| lod | 0 | 0 |
| lit | 0 | 1 |
| opr | 0 | 2 |
| sto | 0 | 0 |
| lod | 0 | 0 |
| lod | 0 | 1 |
| opr | 0 | 3 |
| opr | 0 | 2 |
| opr | 0 | 2 |
| sto | 0 | 2 |
| lod | 0 | 0 |
| lod | 0 | 1 |
| opr | 0 | 10 |
| jpc | 0 | 28 |
| jmp | 0 | 7 |

1. **SAIDA PARA EXECUÇÃO DAS QUESTÕES**

**4.1 Questão 1:**

lit 0 1 1 1

sto 0 1 1 1

lit 0 1 1 10

sto 0 1 1 10

lit 0 1 1 0

sto 0 1 1 0

lod 0 1 1 0

lod 0 1 2 0 0

opr 0 1 1 0

sto 0 1 1 0

lod 0 1 1 0

lit 0 1 2 0 1

opr 0 1 1 1

sto 0 1 1 1

lod 0 1 1 1

lod 0 1 2 1 1

opr 0 1 1 0

jpc 22 1 0

jmp 7 1 0

**4.2 Questão 2:**

lit 0 1 1 1

sto 0 1 1 1

lit 0 1 1 100

sto 0 1 1 100

lit 0 1 1 0

sto 0 1 1 0

lod 0 1 1 0

lod 0 1 2 0 0

opr 0 1 1 0

sto 0 1 1 0

lod 0 1 1 0

lit 0 1 2 0 1

opr 0 1 1 1

sto 0 1 1 1

lod 0 1 1 1

lod 0 1 2 1 1

opr 0 1 1 1

opr 0 1 0

sto 0 1 0

lod 0 1 0

lod 0 1 1 1

opr 0 1 0

jpc 27 1 -1

jmp 7 1 -1

**4.3 Questão 3:**

lit 0 1 1 1

sto 0 1 1 1

lit 0 1 1 1000

sto 0 1 1 1000

lit 0 1 1 0

sto 0 1 1 0

lod 0 1 1 0

lod 0 1 2 0 0

opr 0 1 1 0

sto 0 1 1 0

lod 0 1 1 0

lit 0 1 2 0 1

opr 0 1 1 1

sto 0 1 1 1

lod 0 1 1 1

lod 0 1 2 1 1

opr 0 1 1 0

opr 0 1 0

opr 0 1 -1

sto 0 1 -1

lod 0 1 -1

lod 0 1 0

opr 0 1 -1

jpc 0 1 -2

jmp 7 1 -2

1. **LINKS PARA DOWNLOAD**

- link para o GitHub: [https://github.com/UESC/tree/main/CLP/trabalho2](https://github.com/DRhiuky/UESC/tree/main/CLP/trabalho2)

1. **REFERÊNCIAS:**

- <https://en.wikipedia.org/wiki/P-code_machine>

- [https://prograd.uesc.br/MaterialApoio/Aula/VirtualMachine.pdf](https://trendspdf.prograd.uesc.br/MaterialApoio/Diario/Aula/2003059831/Lecture02VirtualMachinesUCF2010.pdf)