**Breve história da linguagem C:**  
 Em 1972, no laboratório da empresa Bell Telephone Labs, foi projetada a linguagem de programação de computadores C por Dennis Richied e Brian Kerbugh, que a desenvolveram com o objetivo de utilizá-la na codificação da segunda versão do sistema operacional Unix para a equipe de trabalho chefiada por Ken Thompson.  
 A primeira versão do sistema operacional Unix foi escrita em linguagem Assembly (Assembler). A partir da segunda versão até hoje o Unix é escrito com linguagem C. A linguagem C é derivada da Algol e foi baseada na linguagem B de Ken Thompson que era uma evolução da linguagem BCPL. Alguns autores acharam que possivelmente a próxima linguagem baseada na estrutura C seria chamada de P, mas não ocorreu, pois a seguinte à linguagem C foi chamada de C++ (Linguagem C com suporte à Programação Orientada a Objetos), desenvolvida por Bjarne Stroustrup.  
 A linguagem C nasceu da necessidade de escrever programas que utilizem os recursos internos da máquina de uma forma mais fácil que a linguagem de montagem Assembly. C permite a integração direta entre alto nível e baixo nível, permitindo a escrita de códigos Assembly dentro do código em alto nível.  
 A grande aceitação da linguagem decorre da elegância em conciliar seu poder de programação em baixo nível com o seu alto grau de portabilidade, ou seja, um programa escrito em C pode teoricamente ser rodado em qualquer plataforma. C é uma linguagem de propósito geral, estruturada do tipo imperativa com suporte a programação estruturada, por possibilitar o trabalho de programação com base no paradigma procedural.  
**Instruções:**  
 Para desenvolver um programa que seja executável em um computador, é necessário ter em mente que o trabalho do gerenciamento de dados ocorre me três níveis:  
 1. Entrada de dados;  
 2. Processamento (transformação) dos dados em informação;  
 3. Saída de informação.  
  
 O trabalho de gerenciamento de dados em um computador executa instruções que devem estar ordenadas de forma lógica para realizar uma determinada ação e atender certa necessidade.  
 As instruções de operação na linguagem C são obtidas a partir de comandos e funções. A seguir apresentam-se alguns comandos e funções utilizados em C:

* Os comandos são definidos por palavras reservadas da linguagem escritas de forma simples, podendo-se destacar break, case, char, default, do, else, float, if, include, int, switch, while e outros;
* As funções são definidas por palavras reservadas da linguagem seguidas de parênteses.  
  Podendo-se destacar for(), printf(), puts(), scanf() e outras.

Tanto comandos como funções da linguagem C são escritos com caracteres em formato minúsculo, e assim devem ser definidos, pois C é uma linguagem considerada case sensitive, ou seja, suas instruções levam em conta a diferenciação dos caracteres. Caso escreva comandos ou funções em maiúsculo, obterá erro de compilação.

**Programa** É um algoritmo descrito em uma linguagem de programação, ou seja, em uma linguagem que o computador seja capaz de interpretar e executar.

Em linguagem C:

* Comentários: são ignorados pelo computador e servem apenas para introduzir lembretes para o próprio programador; são opcionais em um programa, porém sem uso é fortemente recomendado.

Em C há dois tipos de comentários

-os delimitadores por /\* e \*/, que podem se estender por várias linhas;

-os iniciados por //, que terminam no final da linha.

* Diretivas: Do tipo #include servem para incluir bibliotecas de comandos pré-definidos da linguagem: A inclusão da biblioteca de entrada e saída padrão stdio.h (standard.input/output header) é necessária sempre que precisamos usar o comando de entrada padrão scanf(), que lê dados digitados no teclado e os armazena em variáveis, e o comando de saída padrão printf(), que exibe dados em vídeo;

#include <stdio.h>

main()

{

bloco comandos

}

* A lógica do programa: É implementada por uma função principal, denominada main(). Todo programa em C deve ter uma função main(), por onde se inicia a sua execução. Os comandos da função são delimitados por um par de chaves;
* Variáveis: Devem ser declaradas antes de serm usadas. A declaração de uma variável consiste na indicação do tipo de dados que será armazenado por ela, bem como do nome usado para identificá-lá.
* A função printf(): é usada para exibição de dados formatados no vídeo;
* A função scanf(): é usada para exibição de dados formatados no vídeo;
* Expressões aritméricas: são compostas por variáveis, operadores aritméticos (+, -, \* e /); o operador de atribuição (=) avalia a expressão à sua direita e armazena o resultado desta na variável indicanda à sua esquerda;
* Todas as palavras reservadas: com significado predefinido na linguagem, devem ser escritas em letras minúsculas (ex: main, if...)

**Etapas básicas para a criação de um programa**

* **Análise**: nessa etapa, precisamos compreender o problema em questão, definindo que dados são fornecidos com entrada, que processamento deve ser efetuado r que informações devem ser apresentadas como saída (esta é a etapa em que especificamos as telas de execução).
* **Projeto**: nessa etapa, precisamos elaborar um algoritmo que descreva, passo a passo, como o computador deve proceder para obter os dados de entrada, processá-los e exibir as informações de saída, de acordo com o que foi definido na etapa de análise (esta é a etapa em que construímos o fluxograma).
* **Implementação**: messa etapa, precisamos codificar um programa correspondente ao algoritmo elaborado na etapa de projeto (esta é a etapa em que usamos uma linguagem de programação)
* **Testes**: nessa etapa, precisamos executar o programa num computador e verificar como ele se comporta para diversos dados de entrada (esta é a etapa em que usamos um compilador);

Um compilador é um programa que interpreta os comandos escritos em uma linguagem de programação e os converte em uma forma que o computador seja capaz de executar.

**Tipos de dados**

Os dados são representados por elementos brutos a serem processados por um computador, a fim de se transformarem em informação. A linguagem C tem a capacidade de dados básicos utilizados por um computador são numéricos (inteiros ou reais), caracteres e lógicos:

* Dados inteiros: são inteiros os dados numéricos positivos ou negativos, excluindo qualquer número fracionário (números do tipo real). Na maior parte dos programas será utilizados o tipo de identificador **int.**
* Dados reais: são reias os dados numéricos positivos, negativos, números fracionários e também inteiros. Em uma linguagem C, esse tipo de dado pode ser referenciado pelo idenrificador **float**. O tipo de dado real permite trabalhar com uma representação de valores em ponto flutuante que consiste em uma parte inteira e uma mantissa (parte fracionária).
* Dados caracteres: são classificados como caracteres os dados inteiros que representam números e símbolos delimitados com aspas simples (‘’). Esse tipo de dados é referenciados pelo identificador **char.**
* Dados de controle:
  + %i
  + %f
  + %c 🡪 1 byte
  + %s 🡪 mais de um byte

Tipos:

Int: números inteiros

Float: “ fracionários

Char: letras, nºs e caracteres especiais

**O uso de variáveis**

Variáveis do ponto de vista de programação, é uma região de memória preciamente identificada que tem por finalidade armazenar os dados ou informações de um programa por um determinado tempo. Uma variável limita-se a armazenar aprenas um valor por vez. Sendo considerado como valor o conteúdo de uma variável, desta forma, um valor está relacionado ao tipo de dado de uma variável, que pode ser numérico, lógico ou caractere.

O nome de uma variável é utilizado para sua identificação e posterior uso em um programa. Sendo assim, é necessário estabelecer algumas regras de utilização de variáveis:

* Nomes de usuários podem ser atribuídos com um ou mais caracateres;
* O primeiro caractere do nome de uma variável não pode ser, em hipótese nenhuma, um número; sempre deve ser letra ou, no caso da linguagem C, pode-se iniciar o nome de uma variável com o caractere underline “\_”,
* O nome de uma variável não pode ter espaços em branco;
* Não pode ser nome de um variável uma palavra reservada a uma instrução ou identidicador de uma linguagem de programação;
* Não podem ser utilizados outros caracateres em nomes de variáveis a não ser letras, números e o caractere underline, utilizado normalmente para simular a separação de palavras compostas como nomes de variáveis, como, por exemplo, NOME\_ALUNO;
* Outros detalhes a ser considerado na linguagem C é a diferença entre caracteres maiúsculos e minúsculos. Sendo assim, as variáveis NOME, nome, Nome, nome são diferentes.

**Operadores aritméticos**

|  |  |  |
| --- | --- | --- |
| **Operador** | **Operação** | **Exemplos** |
| + | Adição | Soma = A + B |
| - | Subtração | Subtracao = A – B |
| % | Restro da divisão | Resto = A%B |
| / | Divisão | Divisão = A/B |
| \* | Multiplicação | Multiplicacao = A\*B |
| pow(base,expoente) | Expoenciação | Exponenciacao = pow(A,B) |
| sqrt(valor numérico) | Raiz quadrada | Raiz = sqrt(A) |
| = | Atribuição |  |

**OBS:** Para utilizar as funções pow() e sqrt(), é necessário que a biblioteca math.h seja inicialmente incluída no programa com o comando #include antes da função principal main(), utilizando a sintaxe #include <math.h>

**Estrutura de um programa C**

Declaração de bibliorecas

Main()

{

Declaração de variáveis;

Entrada de dados;

Cálculos;

Exibição de resultados;

}

**Exemplo de um programa em C**

//programa que faz a soma de dois números inteiros

#include<stdio.h>

main()

{

Int A, B, SOMA;

printf(“Digite o primeiro número:”)

scanf(“%i”,&A);

printf(“Digite o segundo número”);

scanf(“%i”,&B);

SOMA=A+B

Printf(“A soma dos números é %i”,SOMA);

}

//estrutura scanf🡪 scanf(“string de controle”,&variável)

string de controle🡪 int %i

float %f (“%.2f” 🡪 indica quantas casas decimais o float terá

char %c // apenas um caractere

%s //mais de um caractere

**Acentuação de caracteres em C com locale.h**

#include <locale.h> // necessário para utilizar setlocate

Main()

{

setlocate(LC\_ALL, “Portuguese”);

//utilizamos a função setlocale() para fazer a adaptação do programa ao idioma desejado

}

8/4/24 **Condições e Decisões**

Para que seja possível um programa de computador tomar decisões, é necessário primeiramente imputar uma condição.

**Decisão Simples:**

Para solucionar o problema, é necessário trabalhar com uma nova instrução, denominada **if**, que tem por finalidade tomar uma decisão e efetuar um desvio no processamento do programa, dependendo, da dondição atribuída ser Verdadeira ou Falsa.

Sendo a condição Verdadeira, será executada a instrução que estiver após a instrução **if**. Caso seja necessário executar mais de uma instrução para uma condição verdadeira, elas devem estar escritas dentro de um bloco. Um bloco é definido com os símbolos de chaves “{“ e ”}”.

Sendo a condição Falsa, executam-se as instruções que estejam após a instrução considerada Verdadeira, ou as instruções após o bloco considerado Verdadeiro.

**Sintaxe** (forma de escrever o comando):

If (condição)

Instrução para a condição verdadeira;

Instrução para a condição falsa;

Caso exista a necessidade de processar mais de uma instrução para uma mesma condição Verdadeira, elas devem estar inseridas em um bloco, como indicado a seguir:

If (condição)

{

instrucao1 para condição Verdadeira;

instrucao2 para condição Verdadeira;

instrucao3 para condição Verdadeira;

}

instrucao para a condição Falsa;

**Opedaroes Racionais:**

**Simbolo Signigicado**

== Igual a

!= Diferente de

> maior que

< menor que

>= maior ou igual que

<= menor ou igual que

**Decisão Composta:**

Utiliza a instrução if..else. Sendo a condição Verdadeira, será executada a instrução que estiver posicionada entre as instruções if e else. Sendo que a condição Falsa, será executada a instrução que estiver posicionada logo após a instrução else.

Caso seja necessário considerar mais de uma instrução para as condições Verdadeira ou Falsa, usam-se blocos delimitados por chaves.

**Sintaxe:**

if (condição)

Instrução para condição verdadeira;

else

instrução para condição falsa;

15/4/24

**Operadores lógicos**

Existem ocasiões em que é necessário trabalhar com o relacionamento de duas ou mais condições a o mesmo tempo na mesma instrução if, efetuando testes múltiplos. Para estes casos é necessário trabalhar com os operadores lógicos, também conhecidos como operadores booleanos.

Os operadores lógicos comumente usados na atividade de programação de computadores são quatro:

* **E** (and);
* **OU inclusive** (or);
* OU exclusivo(xor); -> Utilizado para oprações de nível binário
* **Não** (not)

**Operador lógico de conjunção**

O operador lógico **&& (e)** é utilizado quando dois ou mais relacionamentos lógicos de uma determinada condição necessite ser verdadeiros.

|  |  |  |
| --- | --- | --- |
| **Condição 1** | **Condição 2** | **Resultado I** |
| 0 (falsa) | 0 (falsa) | 0 (falso) |
| 1 (verdadeira) | 0 (falsa | 0 (falso) |
| 0 (falsa) | 1 (verdadeira) | 0 (falso) |
| 1 (verdadeira) | 1 (verdadeira) | 1 (verdadeiro) |

O operador **&&** faz com que somente seja executada uma determinada operação se todas as condições forem verdadeiras, gerando um resultado lógico verdadeiro.

**Operadores lógicos de disjunção inclusiva**

O operador lógico **||** (sentido de OU – or em inglês) é utilizado quando pelo menos um dos relacionamentos lógicos (quando houver mais de um relacionamento) de uma condição necessita ser verdadeiro.

|  |  |  |
| --- | --- | --- |
| **Condição 1** | **Condição 2** | **Resultado lógico** |
| 0 (falsa) | 0 (falsa) | 0 (falso) |
| 1 (verdadeira) | 0 (falsa) | 1 (verdadeira) |
| 0 (falsa) | 1 (verdadeira) | 1 (verdadeira) |
| 1 (verdadeira) | 1 (verdadeira) | 1 (verdadeira) |

Operador **||** faz com que seja executada uma determinada operação, se pelo menos uma das condições for verdadeira.

29/4/24 **Seleção múltipla**

Estrutura de seleção encadeadas

Dizemos que duas estruturas de seleção simples estão encadeadas se uma delas está encaixada no lado falso da outra.

Estruturas de seleção encadeadas são úteis quando precisamos escolhae **apenas um** entre vários comandos altesrnativos.

Em uma estrutura de seleção encadeada, cada **comando** é associado a uma **condição,** que deve ser verdadeira para que ele seja executado; exceto o último comando à direita, que é executado sempre que nenhuma das condições anteriores é verdadeira.

3º Bimestre

29/7/29

**Repetição contada**

**Acumuladores**

Um acumulador é uma variável que ocorre em ambos os lados de uma atribuição e que, antes de ser usada pela primeira vez, é iniciada com um valor específico. Por exemplo, efetuando a atribuição **a=3**, iniciamos a variável **a** com o **valor 3**; então quando a atribuição **a=a+2** é executada, essa variãvel passa a ter o **valor 5**. Note que, se o valor inicial de **a** não é definido pela primeira atribuição, não é possível determinar o seu valor após a execução da segunda atribuição. Pelo fato de operações com acumuladores serem tão comuns em programação, C oferece um conjunto de operadores **aritmérico de atribuição**, que permite escrever expressões com a cumuladores de uma forma mais compacta. Por exemplo, se usando o operador aritimético de atribuição **+=**, a operação **a=a+2** pode ser codificada como **a+=2** e a operação de atribuição **a=a+2\*b** pode ser codificada como **a+=2\*b**.

|  |  |  |
| --- | --- | --- |
| **Operação** | **Fluxograma** | **Programa em C** |
| Soma | a=a+expressão | a+=expressão |
| Subtração | a=a-expressão | a-=expressão |
| Multiplicação | a=a\*expressão | a\*=expressão |
| Divisão | a=a/expressão | a/=expressão |
| Resto | a=a%expressão | a%=expressão |

Um contador é um tipo de acumulador cujo valor aumento, ou diminui, de 1 em 1. A tabela abaixo mostra os operadores em C para lidar com contadores.

|  |  |  |
| --- | --- | --- |
| **Operação** | **Fluxograma** | **Programa em C** |
| Incremeto | c=c+1 | c++ |
| Descremento | c=c+1 | c-- |

**Laços de repetição**

Existem situações em que é necessário repetir o trecho de um programa de um determinado número de vezes, o que pode ser conseguido de duas formas. Na primeira será escrito o mesmo trecho tantas vezes quanto necesário, bastante trabalhoso; na segunda forma podem ser utilizados laços de repetição, conhecidos também como loopings ou malhas de repetição.

A vantagem em utilizar laços de repetição (loopings) é que o programa passa a ser menor e seu processamento aumenta sem alterar o tamanho do código de programação. Desta forma é possível determinar repetições com números variados de vezes.

**Repetição com Precondição**

A estrutura de repetição com precondição serve para executar um comando, repedidamente, enquanto uma determinada condição for verdadeira. Observe que, como a condição é avaliada antes do comando ser executado, se ela for inicialmente falsa, o comando dentro da repetição jamais é executado.

**Sintaxe**

While (condição)

{

Instruções para condição verdadeira;

}

/\*Elaborar um programa que solicite cinco vezes dois números para o cálculo e apresentação de uma adição \*/

#include <stdio.h>

main()

{

Int A, B, R, I,;

I=1;

While (I<=5)

{

Printf(“\n Entre com um valor A: “);

Scanf(“%i”,&A);

Printf(“\n Entre com um valor B: “);

Scanf(“%i”,&B);

R=A+B

Printf(“\n O resultado corresponde a: %i”, R);

I+=1;

}

}

/\*Faça um programa que entre com nome e idade de três pessoas, exibindo “Fulano tem xx anos de idade”\*/

5/8/24

/\* Em um banco, as contas são identificadas por um número de conta com dígito verificador. Esse dígito verificador é calculado do seguinte modo: primeiramente somamos todos os dígitos do número da conta, depois dividimos a soma por 10 e tomamos o resto. Por exempli, se o número da conta for 5713, temos a soma 3+1+7+5=16; dividindo 16 por 10, temos resto 6, que é seu dígito verificador. Dado um número de conta, informe o dígito verificador correspondente\*/

#include<stdio.h>

main(){

int n, s, t, d;

printf(“número:”);

scanf(“%i”,&n);

s=0;

while(n>0){

r=n%10;

n/=10; //equivale a n=n/10

s+=r; //equivale a s=s+r

}

d=s%10;

printf(“DV = %i\n”, d);

}

Imagine uma outra situação na qual o usuário deseja executar a rotina do programa várias vezes, mas não sabe quantas vezes ao certo deve executar o trecho de promagra. Neste caso, seria melhor que o programa fizesse uma pergunta, solicitando se ele deseja ou não continuar.

#include <stdio.h>

main(){

int A,B,R, RESP=1;

while(RESP==1){

printf(“\n\n Digite um valor para A: ”);

scanf(“%i”,&A);

printf(“\n\n Digite um valor para B: “);

scanf(“%i”,&B);

R=A+B;1

printf(“\n O resultado corresponde a %i”, R);

printf(“\n Deseja Continuar?”);

printf(“\n Tecle [1] para sim / [2] para não: “);

scanf(“%i”,&RESP);

}

}

1-/\* Faça um programa que entre com nome e salário de um funcionário, exiba “O funcionário xxx ganha xxx reais”.Faça o programa rodar 3 vezes. \*/

2-/\* Faça o mesmo programa acima rodando o número de vezes que o usuário desejar \*/

3-// Ler 3 números e apresentar o maior deles.

4-//Faça um programa que calcule a tabuada de um nº. (utilizando o while)

5-//Faça um programa que exiba o menor de 3 números. Faça com while.

6-Calcular a médiade LPI de 5 alunos utilizando while.

7-//Faça um programa que leira o nome e a altura de 5 pessoas. Calcule a média das alturas.

8-/\*Faça um programa que leia peso e o sexo de 4 pessoas. Calcule e exiba a média dos pesos das pessoas, e, quantas mulheres tem no grupo.\*/

2/9/24 Laço condicional pós-teste

Essa estrutura realiza um teste lógico no final do laço de repetição, verificando se é permitido exercutar o trecho de instruções subordinado a ele até certa condição ser satisfeita.

A estrutura **do...while** tem seu funcionamento controlado por condição, portanto executará determinado conjunto de instruções até que a condição seja verdadeira. No momento em que a condição se torna verdadeira, o processamento da rotina é desvidado para fora do laço de repetição. Uma característica deste laço é que essa estrutura faz um teste lógico no final do laço de repetição, permitindo que a ação subordinada ao laço seja executada no mínimo uma vez

Sintexe:

do

{

Faça instrução enquanto condição verdadeira;

}

while(condição)**;**

Uma característica de uso do laço **do...while** é o fato dele ser uma instrução de laço pós-teste verdadeiro.

/\*Ler dois valores, efetuar adição e apresentar o resultado Cinco vezes \*/

#include<stdio.h>

main(){

int A, B, R, i=1;

do{

printf(“Digite um valor para A: “);

scanf(“%i”,&A);

printf(“Digite um valor para B: “);

scanf(“%i”,&B);

R=A+B;

printf(“O resultado corresponde a: %i\n”,R);

i++;

}

while (i<=5);

}

/\*Ler dois números, calcular e exibir a multiplicação enquanto o usuário digitar “S”\*/

#include<stdio.h>

Main(){

int n1,n2,multiplicacao;

char cont=’S’;

do{

printf(“Digite o primeiro número: “);

scanf(“%i”,&n1);

printf(“Digite segundo número: “);

scanf(“%i”,&n2);

multiplicacao=n1\*n2;

printf(“A multiplicação é: %i”,multiplicacao);

scanf(“%c”,&cont);

}

while(cont==’S’);

}

Exercícios

1- Dado um número real não negativo, informe sua raiz quadrada.

2- Dadas as duas notas de um aluno, informe a sua média.

Seu programa deve forçar o usuário a digitar notas na faixa de 0 a 10.

/\*Programa que simula o funcionamento de um caixa eletronico\*/

#include <stdio.h>

#include <locale.h>

main(){

setlocale(LC\_ALL, “Portuguese”);

float S=1000,V;

int op;

do {

puts(“\n 1-Depósito);

puts(“2 – Saque);

puts(“3 – Saldo”);

puts(“4 – Sair”);

scanf(“%i”,&op);

switch(op) {

case 1: printf(“\n Valor do depósito: “);

Scanf(“%f”,&V);

S=S-V; break;

case 2: printf(“\n Valor do saque: “);

Scanf(“%f”,&V);

S=S-V; break;

case 3: printf(“\n Saldo atual = R$ %.2f\n”,S);

break;

default: if (op!=4) puts(“\n Opção inválida!”);

}

} while (op!=4);

puts(“fim das transações”);

}

4º Bimestre

# Laço com variável de controle

Foram apresentadas anteriormente duas formas de elaborar laços de repetição. Com as técnicas mostradas é possível elaborar rotinas que exacutem um laço, um determinado número de vezes, com a utilização de um contador ou mesmo de uma variável que aguarde a resposta do usuário. Independentemente da forma de tratamento, ela é denominada variável de controle.

Existe outra forma que facilita o uso de contadores finitos sem usar as estruturas de laço anteriormente apresentadas. Os laços de repetição com **while** e **do...while** passam a ser utilizados quando não se conhece de antemão o número de vezes que uma determinada sequência de instruções deve ser executada. Os laços de repetição que possuem um número finito de execuções podem ser processados por meio do laço **for**.

Esse laço de repetição tem o funcionamento controlado por uma variável contador, que pode ser crescente ou decrescente, tendo como sintaxe:

for(início;fim;incremento){

instruções;

}

Caso exista mais de uma instrução dentro do laço, elas devem estar inseridas em um bloco, delimitado pelos símbolos de chaves.

**Início:** É uma instrução de atribuição com o valor inicial do laço

**Fim**: É uma instrução de condição com o valor final do laço

**Incremento:** expressão com o incremento do laço

/\* Elaborar um programa que escreva na tela os valores 1,2,3,4,5,6,7,8,9,9 e 10 \*/

#include<iostream>

using namespace std;

int main(){

int I;

for(i=1;i<=10;I++){

cout << i << “\n”;

}

return 0;

}

//Programa que escreve na tela os valores 1,3,5,7,9

#include<iostream>

Using namespace std;

int main(){

int n;

for(n=1;n<=10;n+=2){

cout << n;

}

return0;

}

21/10/24 Vetores

Um **vetor** é um tipo de variável capaz de armazenar uma coleção de dados do mesmo tipo. Cada um dos dados armazenados num vetor, denominado **item**, é identificado por um número natural, a partir de 0, denominado **índice**.

Para indicar que uma variável é um vetor, devemos declará-la com o sufixo [n], sendo n um valor inteiro positivo que estabelece o tamanho do vertor.

Por exemplo, a declaração:

char v[3];

cria um vetor com três posições, cada uma delas capaz de armazenar um caractere. Na verdade, com esta única declaração, criamos as variáveis v[0], v[1] e v[2]. Note que, com a indexação inicia-se em 0, o úlitmo item de um vetor de tamanho n é armazenado na posição n-1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| C | E | M | E | P |

Índice->0 1 2 3 4

\*represetação de um vetor na memória do computador.\*