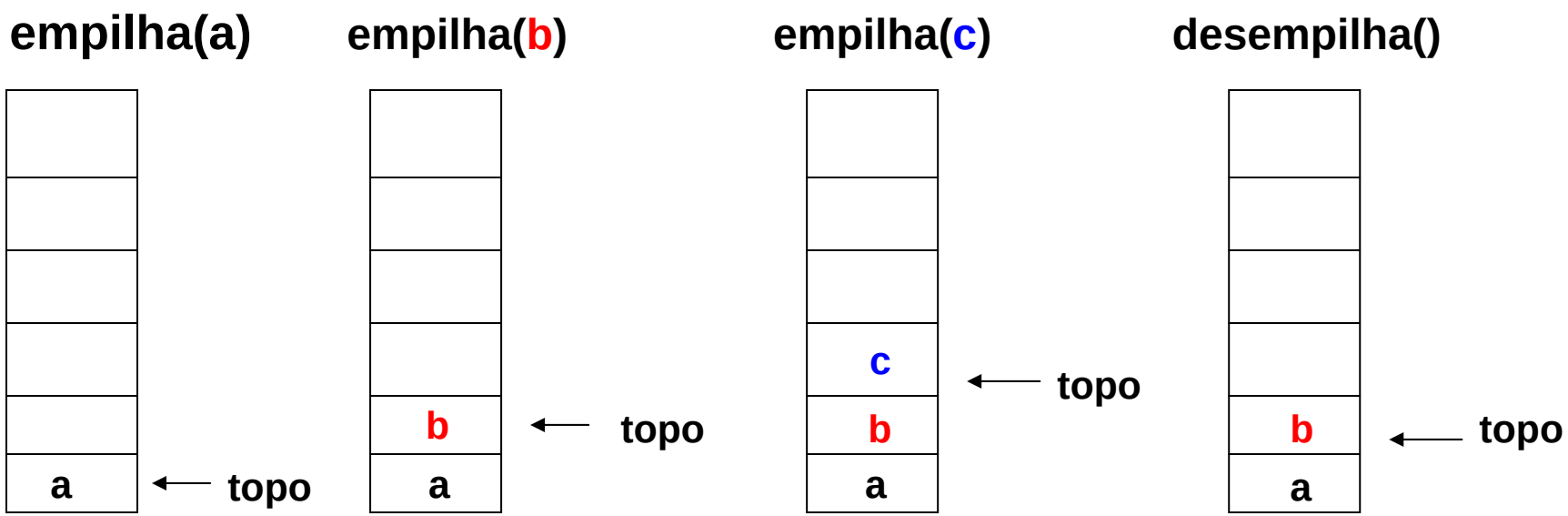
Métodos Computacionais

Pilha

Definição de Pilha

- Pilha é uma estrutura de dados dinâmica onde:
  - Inserção e remoção de elementos no topo da pilha
  - O primeiro elemento que sai é o último que entrou (LIFO)
  - Operações básicas: push (empilhar) e pop (desempilhar)

Funcionamento da pilha

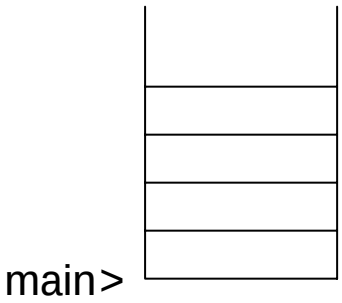


Exemplo de Uso: Pilha de Execução de Funções

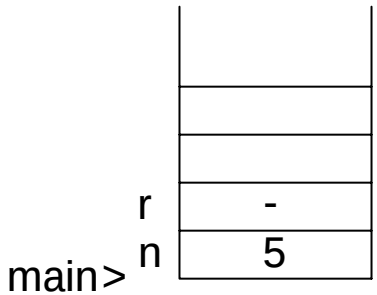
```
#include “stdio.h”
int  fat ( int  n ) ;
int main () {
  int  n = 5,  r ;
  r = fat(n) ;
  printf ( “ Fatorial  = %d \n ”,  r ) ;
  return 0 ;
}
int  fat  ( int  n ){
  int  f=1 ;
  while(n != 0) {
      f  *=  n ;
      n-- ;
  }
  return f ;
}
```

Pilha de Execução

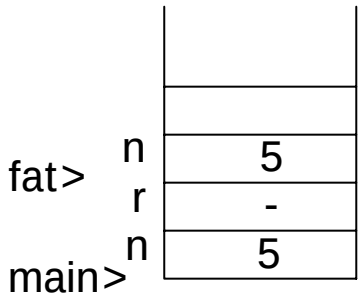
1 – Início do programa: pilha vazia



2 – Declaração de variáveis: n,r

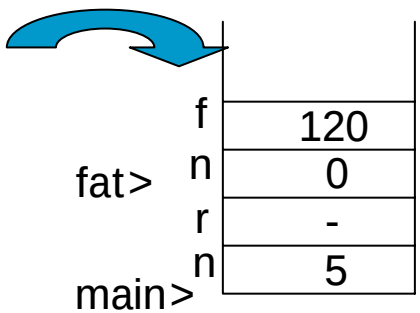


3 – Chamada da função: empilha variáveis

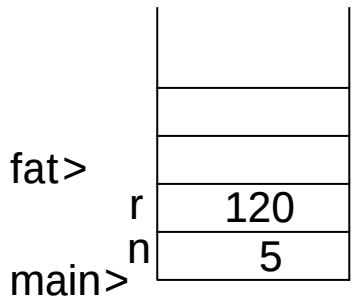


Acesso às variáveis que estão na função do topo

4 – Final do laço



5 – Retorno da função: desempilha



Interface do tipo Pilha

- Criar pilha vazia;
- Inserir elemento no topo (push)
- Remover elemento do topo (pop)
- Verificar se a pilha está vazia
- Liberar a estrutura de pilha

- Em C

```
/* pilha.h */
typedef struct pilha Pilha;
Pilha* pilha_cria();
void pilha_push(Pilha* p,float v);
float pilha_pop(Pilha* p);
int pilha_vazia(Pilha* p);
void pilha_libera(Pilha* p);
```

Implementação do tipo Pilha

- Existem várias implementações possíveis de uma pilha. Podem diferir da natureza dos elementos, maneira como são armazenados e operações disponíveis
- Iremos estudar 2 tipos de implementação:
  - Utilizando Vetor
  - Utilizando Lista Encadeada

Implementando Pilha com Vetor

- Estrutura que representa pilha deve ser composta por um vetor para armazenar os elementos e o número de elementos armazenados
- Vantagem
  - Simplicidade
- Desvantagens
  - Deve-se saber de antemão o número máximo de elementos
  - Desperdício de espaço de memória

```
#define N 50
struct pilha {
  int n;
  float vet[N];
};
```

Implementando Pilha com Vetor

- Função de Criação

```
typedef struct pilha Pilha;

Pilha* pilha_cria ()
{
  Pilha* p = (Pilha*) malloc(sizeof(Pilha));

  p->n = 0;
  return p;
}
```

Aloca estrutura dinamicamente

Inicializa com 0 elementos

Implementando Pilha com Vetor

- Função de Inserção

```
void pilha_push (Pilha* p, float v) {
  if (p->n == N) { /* capacidade esgotada */
     printf("Capacidade da pilha estourou.\n");
     exit(1);
  }

  p->vet[p->n] = v;
  p->n++;
}
```

Verifica se tem espaço disponível

Insere na próxima posição livre e incrementa o número de elementos da pilha

Implementando Pilha com Vetor

- Função de Remoção

```
float pilha_pop (Pilha* p)
{
  float v;
  if (pilha_vazia(p)) {
     printf("Pilha vazia.\n");
     exit(1);
  }

  v = p->vet[p->n-1];

  p->n--;
  return v;
}
```

Verifica se a pilha está vazia

Topo está na posição n - 1

Remoção se dá pelo decremento do número de elementos (próximo push sobrescreverá antigo topo)

Implementando Pilha com Vetor

- Função que testa se pilha está vazia

```
int pilha_vazia (Pilha* p) {
  return (p->n==0);
}
```

- Função que libera memória alocada para a pilha

```
void pilha_libera (Pilha* p) {
  free(p);
}
```

Implementando Pilha com Vetor

- Outras Funções Utilitárias

```
/* Função que informa o elemento do topo */
float pilha_topo (Pilha* p) {
  return (p->vet[p->n – 1]);
}
```

```
/* Função que imprime os elementos da pilha
void pilha_imprime (Pilha* p) {
  int i;
  for (i=p->n-1; i>=0; i--)
     printf(“%f\n”, p->vet[i]);
}
```

Implementando Pilha com Lista

- Estrutura que representa pilha deve ser composta por um ponteiro para o primeiro nó da lista que armazena os elementos
- Vantagens
  - Otimização da memória
  - Tamanho irrestrito
- Desvantagem
  - Complexidade na manipulação de listas

```
typedef struct lista{
  float info;
  struct lista *prox;
} Lista;

struct pilha {
  Lista *topo;
};
```

Implementando Pilha com Lista

- Função de Criação

```
typedef struct pilha Pilha;

Pilha* pilha_cria () {
  Pilha* p = (Pilha*) malloc(sizeof(Pilha));
  p->topo = NULL;
  return p;
}
```

Lista(topo) é inicializada com NULL

Implementando Pilha com Lista

- Função de Inserção

```
void pilha_push (Pilha* p, float v) {
  Lista* novo = (Lista*) malloc(sizeof(Lista));
  novo->info = v;
  novo->prox = p->topo;
  p->topo = novo;
}
```

Elemento é inserido no começo da lista – topo aponta para o começo da lista

Implementando Pilha com Lista

- Função de Remoção

```
float pilha_pop (Pilha* p) {
  Lista* t;
  float v;
  if (pilha_vazia(p)) {
     printf("Pilha vazia.\n");
     exit(1);     /* aborta programa */
  }
  t = p->topo;
  v = t->info;
  p->topo = t->prox;
  free(t);
  return v;
}
```

Elemento é realmente removido, topo é atualizado

Implementando Pilha com Lista

- Função que testa se pilha está vazia

```
int pilha_vazia (Pilha* p) {
  return (p->topo == NULL);
}
```

- Função que libera memória alocada para a pilha

```
void pilha_libera (Pilha* p) {
  Lista* q = p->topo;
  while (q!=NULL) {
     Lista* t = q->prox;
     free(q);
     q = t;
  }
  free(p);
}
```

Deve-se liberar todos os elementos da lista primeiro

Implementando Pilha com Lista

- Outras Funções Utilitárias

```
/* Função que informa o elemento do topo */
float pilha_topo (Pilha* p) {
  return (p->topo->info);
}
```

```
/* Função que imprime os elementos da pilha
void pilha_imprime (Pilha* p) {
  Lista* q;
  for (q=p->topo; q!= NULL;  q = q->prox)
     printf("%f\n", q->info);
}
```

Exemplo de uso: Calculadora Pós-Fixada

- Em algumas calculadoras HP, expressões do tipo `(1-2)*(4+5)` são escritas na forma:

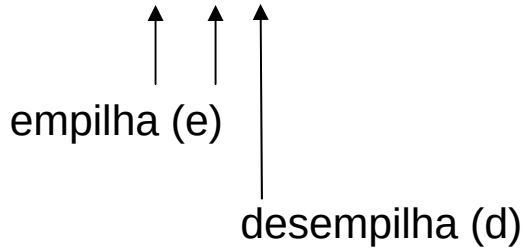
  `1 2 - 4 5 + *`

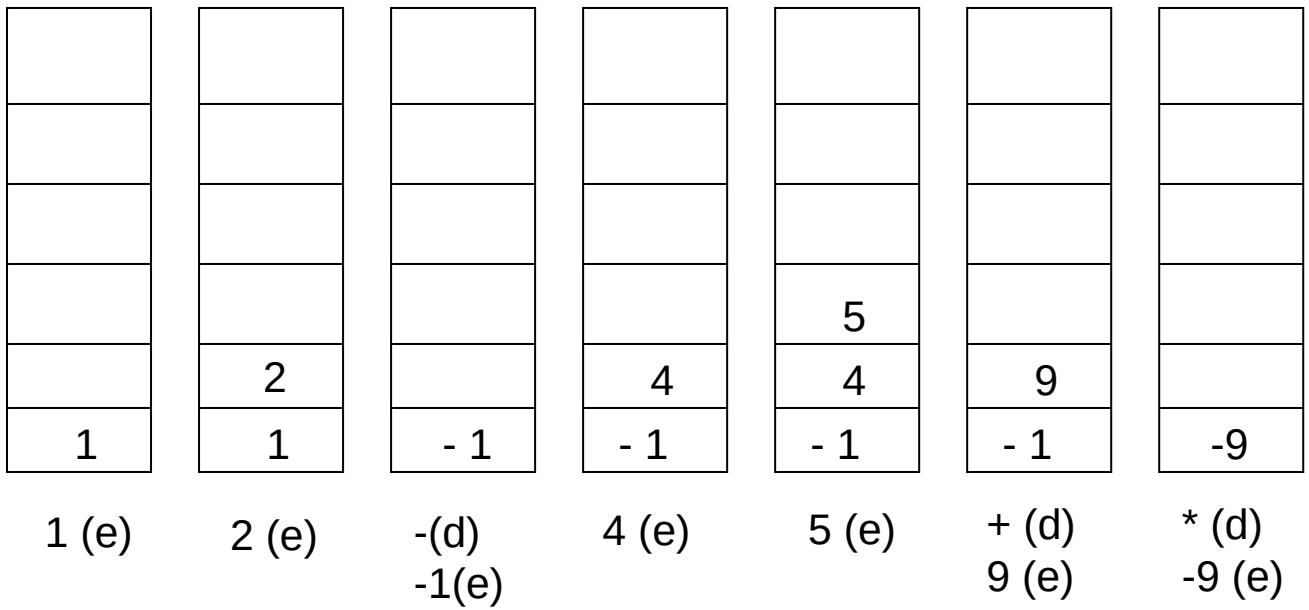
  - Notação pos-fixada ou polonesa
- Cada operando é empilhado em uma pilha de valores
- Ao se encontrar um operador, desempilha-se o número apropriado de operandos (2 para operadores binários e um para operadores unários), calcula-se o valor da expressão e empilha-o novamente
- Este processo segue até o final da expressão

Exemplo de uso: Calculadora Pós-Fixada

- Supondo que o usuário digita 1 2 - 4 5 + *

≡ (1-2)*(4+5)





Implementando Calculadora Pós-Fixada

- Interface da Calculadora

```
/* arquivo calc.h: define a interface da calculadora
   */

typedef struct calc Calc;

/* funções exportadas */
Calc* calc_cria (char* f); /* Formato do resultado */
void calc_operando (Calc* c, float v);
void calc_operador (Calc* c, char op);
void calc_libera (Calc* c);
```

Exemplo de uso: Calculadora Pós-Fixada

- Implementação da Calculadora usando Pilha (independente da implementação de Pilha)

```
/* arquivo calc.c*/

struct calc {
  char f[21]; /* formato para impressão */
  Pilha* p;
}
```

- Função de Criação da calculadora

```
Calc* calc_cria (char* formato) {
  Calc* c = (Calc*) malloc(sizeof(Calc));
  strcpy (c->f, formato);
  c->p = pilha_cria(); /* cria pilha vazia */
  return c;
}
```

Exemplo de uso: Calculadora Pós-Fixada

- Função para empilhar operando

```
void calc_operando (Calc* c, float v) {
  /* empilha operando */
  pilha_push (c->p, v);
  /* imprime topo da pilha */
  printf (c->f, v);
}
```

- Função que libera a pilha de operandos e estrutura da calculadora

```
void calc_libera (Calc* c) {
  pilha_libera (c->p);
  free (c);
}
```

Exemplo de uso: Calculadora Pós-Fixada

- Função que desempilha operandos e calcula

```
void calc_operador (Calc* c, char op) {
  float v,v1, v2;
  /* desempilha operandos */
  if (pilha_vazia (c->p))          /* assume o valor do
   operando como zero */
      v2 = 0.0;
   else
      v2 = pilha_pop (c->p);
  if (pilha_vazia (c->p))        /* assume o valor do
   operando como zero */
     v1 = 0.0;
  else
     v1 = pilha_pop (c->p));


...
```

Exemplo de uso: Calculadora Pós-Fixada

```
/* faz operação */
  switch (op) {
     case ‘+’: v = v1+v2; break;
     case ‘-’:   v = v1-v2; break;
     case ‘*’:  v = v1*v2; break;
     case ‘/’:   v = v1/v2; break;
  }

 /* empilha resultado */
  pilha_push (c->p, v);

  /* imprime topo da pilha */
  printf (c->f, v);
}
```

Exemplo de uso: Calculadora Pós-Fixada

- Programa para ler expressão e chamar funções da calculadora

```
#include <stdio.h>
#include “calc.h”

int main (void) {
   char c; /* caracter a ser lido no buffer da
   calculadora */
   float v; /* valor associado a um operando */
   Calc* calc;

  /* cria calculadora com formato para impressão de
   duas casas decimais */
  calc = calc_cria(“%.2f\n”);
...
```

Exemplo de uso: Calculadora Pós-Fixada

```
do {
   scanf("%c", &c); /* lê próximo caracter não branco */
   /* verifica se é operador válido */
   if (c=='+' || c =='-' || c=='*'  || c=='/') {
       calc_operador (calc, c); /* calcula o resultado da
 aplicação do operador */
   } else {
       ungetc(c, stdin); /* devolve o caractere lido e
 tenta ler número */
       if (scanf("%f", &v) == 1)
          calc_operando (calc, v);
   }
 } while (c!='q'); /* a execução é finalizada ao digitar q
 */
 calc_libera (calc);
 return 0;
}
```

Exemplo de uso: Execução do programa

```
3 5 8 * +     /* digitado pelo usuário (equivale a (3 + (5*8)) */
3.00          /* impresso pelo programa */
5.00          /* impresso pelo programa */
8.00          /* impresso pelo programa */
40.00         /* impresso pelo programa (5*8) */
43.00         /* impresso pelo programa (3 + (5*8)) */
7 /           /* digitado pelo usuário */
7.00          /* impresso pelo programa */
6.14          /* impresso pelo programa (43/7) */
```