

# Sistemas Hardware-Software

## Aula 12 – Tipos Abstratos de Dados

2019 – Engenharia

Igor Montagner [<igorsm1@insper.edu.br>](mailto:igorsm1@insper.edu.br)

# malloc

```
#include <stdlib.h>  
void *malloc(size_t size)
```

**Se bem sucedido:** retorna ponteiro para bloco de memória com pelo menos **size** bytes reservados, e com alinhamento de 8 bytes em x86, ou 16 bytes em x86-64. Se **size** for zero, retorna **NULL**.

**Se falhou:** retorna **NULL** e preenche **errno**

# free

```
#include <stdlib.h>  
void free(void *p)
```

Devolve o bloco apontado por **p** para o *pool* de memória disponível

# Alocação dinâmica

- Vantagens
  - Controle feito em tempo de execução
  - Economia de memória
  - Expandir / diminuir / liberar conforme necessário
- Desvantagens
  - Riscos da gerência
    - Liberar espaços não mais necessários
    - Não acessar espaços já liberados
    - Acessar apenas a quantidade requisitada
    - Etc.

# Alocação dinâmica de memória

- Alocadores organizam o heap como uma coleção de blocos de memória que estão **alocados** ou **disponíveis**
- Tipos de alocadores
  - Explícitos: usuário é responsável por **alocar** e **deallocar** (ou liberar) a memória.  
Exemplo: malloc, new
  - Implícitos: usuário não precisa se preocupar com a liberação da memória.  
Exemplo: **garbage collector** em Java

# Exemplo

```
#include <stdio.h>
#include <stdlib.h>

void foo(int n) {
    int i, *p;

    /* Allocate a block of n ints */
    p = (int *) malloc(n * sizeof(int));
    if (p == NULL) {
        perror("malloc");
        exit(0);
    }

    /* Initialize allocated block */
    for (i = 0; i < n; i++) {
        p[i] = i;
    }

    /* Return allocated block to the heap */
    free(p);
}
```

# Tipos Abstratos de Dados



A troca entre postes funciona como uma pilha

# Tipos Abstratos de Dados

Operações que podem ser feitas com uma pilha:



# Tipos Abstratos de Dados

Operações que podem ser feitas com uma pilha:

1. Criar, Deletar
2. Empilhar
3. Desempilhar
4. Checar tamanho
5. Ver se está vazia

# Tipos Abstratos de Dados

```
struct _stack;
```

```
typedef struct _stack Stack;
```

```
Stack *_stack_new();
```

```
void stack_destroy();
```

```
void stack_push(Stack *p, int i);
```

```
int stack_pop(Stack *p);
```

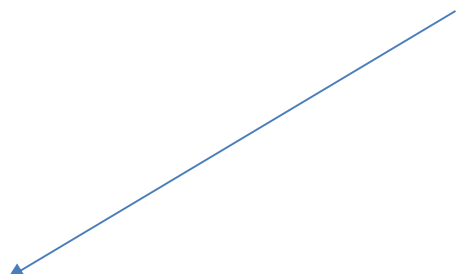
```
int stack_size(Stack *p);
```

```
int stack_empty(Stack *p);
```

# Tipos Abstratos de Dados

```
struct _stack;
```

Não preciso conhecer a  
definição de struct \_stack;



```
typedef struct _stack Stack;
```

```
Stack *stack_new();  
void stack_destroy();
```

```
void stack_push(Stack *p, int i);  
int stack_pop(Stack *p);
```

```
int stack_size(Stack *p);  
int stack_empty(Stack *p);
```

# Tipos Abstratos de Dados

"é uma especificação de um conjunto de dados e operações que podem ser executadas sobre esses dados. Além disso, é uma metodologia de programação que tem como proposta reduzir a informação necessária para a criação/programação de um [algoritmo](#) através da [abstração](#) das variáveis envolvidas em uma única entidade fechada, com operações próprias à sua natureza. " (Wikipedia)

# Tipos Abstratos de Dados

- Conjunto de dados e operações
  - arquivo *.h*
- Criação de algoritmos com essas operações
  - Não depende de detalhes internos

# Tipos Abstratos de Dados

- Vantagens:
  - Código mais expressivo
  - Diminui erros por repetição
  - Evita deixar struct em estado inconsistente
  - Versionamento

# Tipos Abstratos de Dados

- Desvantagens:
  - Esconde todos os detalhes
  - Não permite usos mais avançados ou diferentes do original

# Tipos Abstratos de Dados

Atividade de hoje:

1. Entrega da parte 1 e do exercício 8 da aula passada



# Insper

[www.insper.edu.br](http://www.insper.edu.br)