Insper

Sistemas Hardware-Software

Tipos abstratos de dados

Engenharia
Fabio Lubacheski
Maciel C. Vidal
Igor Montagner
Fábio Ayres

`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

Exemplo – ex1_slide.c

```c
#include <stdio.h> // printf
#include <stdlib.h> // malloc

int main(){
    char *str = malloc(3);
    str[0] = 'a';
    str[1] = 'b';
    str[2] = 'c';
    str[3] = 'd';
    printf("%s",str);
    return 0;
}
```

O programa executa normalmente ??

Possui problema de memória ??

Exemplo – ex1_slide.c

O programa executa normalmente, sem erros. Quando executamos a valgrind temos alguns erros relatados, o primeiro erro encontrado é:

```
==19464== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==19464== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==19464== Command: ./ex1_slide
==19464==
==19464== Invalid write of size 1
==19464==    at 0x1091A8: main (ex1_slide.c:12)
==19464==  Address 0x4a9a043 is 0 bytes after a block of size 3 alloc'd
==19464==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==19464==    by 0x10917E: main (ex1_slide.c:8)
```

Qual o problema ?

Exemplo – ex1_slide.c

O segundo erro é:

```
==19464== Invalid read of size 1
==19464==    at 0x484ED24: strlen (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==19464==    by 0x48E4D30: __vfprintf_internal (vfprintf-internal.c:1517)
==19464==    by 0x48CE79E: printf (printf.c:33)
==19464==    by 0x1091C5: main (ex1_slide.c:13)
==19464==  Address 0x4a9a043 is 0 bytes after a block of size 3 alloc'd
==19464==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==19464==    by 0x10917E: main (ex1_slide.c:8)
```

Qual o problema ?

Como resolver ?

Exemplo – ex1_slide.c

E por fim a seção HEAP SUMMARY faz um resumo dos dados alocados/desalocados no seu programa.

```
==19464== HEAP SUMMARY:
==19464==     in use at exit: 3 bytes in 1 blocks
==19464==   total heap usage: 2 allocs, 1 frees, 1,027 bytes allocated
==19464==
==19464== 3 bytes in 1 blocks are definitely lost in loss record 1 of 1
==19464==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==19464==    by 0x10917E: main (ex1_slide.c:8)
==19464==
```

Qual o problema ?

Como resolver ?

Tipos-de-dados

Um **tipo-de-dados** (=*data type*) é um conjunto de **valores** munido de um conjunto de **operações**.

Quais tipos de dados estão disponíveis em C?

Tipos-de-dados

Um **tipo-de-dados** (=*data type*) é um conjunto de **valores** munido de um conjunto de **operações**.

Quais tipos de dados estão disponíveis em C?

Exemplo o tipo **`int` representam valores inteiros** e é possível realizar **operações** de comparação (<, <=, ==, >, >=) e as aritméticas (+, -, *, /).

Tipos Abstratos de Dados

Um tipo abstrato de dados (=*abstract data type*) é um **tipo-de-dados** definido em termos do seu **comportamento** e não em termos de sua **representação**.

As operações são especificadas em uma **interface que diz o que cada operação faz, sem dizer como faz o que faz**.

Exemplo o TAD – Pilha (=Stack)

No TAD Pilha temos as seguintes **operações**: **inserção** de um item (**Push**) e **remoção** de um item mais recente (**Pop**).

Os **itens** podem tidos-de-dados (int,char) e outros TADs

Comportamento: o primeiro item entrar é o primeiro a sair.

Exemplos de uso de pilha:

- Eliminação de recursividade em funções;
- Navegação entre páginas Web (botão voltar); e
- cálculo do valor de uma expressão aritmética.

TAD – Pilha (=Stack) - Interface

Os programas que usam o TAD só tem acesso a **interface**. Na linguagem C a interface é representada por um arquivo-interface (**=header file**).

```
typedef struct {
    int capacity;
    int *data;
    int size;
} stack_int;

stack_int *stack_int_new(int capacity);
void stack_int_delete(stack_int **_s);
int stack_int_empty(stack_int *s);
int stack_int_full(stack_int *s);
void stack_int_push(stack_int *s, int value);
int stack_int_pop(stack_int *s);
```

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

Tipos Abstratos de Dados

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

Atividade prática

Implementação de Point2D (30 minutos)

1. Revisão de malloc
2. Compilação de programas com mais de um arquivo .c

Vetor dinâmico

O tipo de dados **vetor dinâmico** é implementado em diversas linguagens de alto nível.

- Python: `list`
- Java: `ArrayList`
- C++: `std::vector`

Vetor dinâmico

Suas principais operações são

- criação/destruição
- **at(i)** – devolve elemento na posição i
- **remove(i)** – remove o elemento na posição i, deslocando todos os outros para a esquerda
- **insert(i)** – insere um elemento na posição i, deslocando todos os elementos para a direita

Vetor dinâmico

As operações abaixo mudam o tamanho do vetor!

- **remove(i)** – remove o elemento na posição i, deslocando todos os outros para a esquerda
- **insert(i)** – insere um elemento na posição i, deslocando todos os elementos para a direita

Não é preciso declarar tamanho para o vetor dinâmico

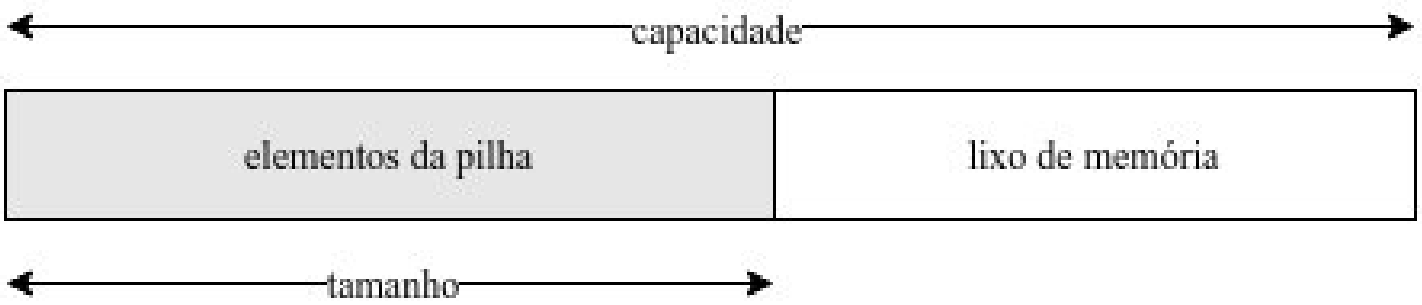
Vetor dinâmico - capacidade

Relembrando *Desafios*

Supondo que soubéssemos o tamanho máximo que o vetor dinâmico assumiria, podemos aplicar esta técnica

Vetor dinâmico - capacidade

E se `tamanho == capacidade`?



Bom, nesse caso precisamos de um espaço de memória maior para nosso vetor!

realloc

#include <stdlib.h>

void *realloc(void *ptr, size_t new_size)

Se bem sucedido: aloca um novo bloco de tamanho new_size, copia o conteúdo apontado por `ptr` para o novo bloco e retorna seu endereço. Antes de retornar chama `free(ptr)`.

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

Vetor dinâmico – redimensionamento

- **Quando encher**: dobrar capacidade
- Quando ficar com **menos de um quarto** da capacidade: **diminuir** a capacidade pela **metade**

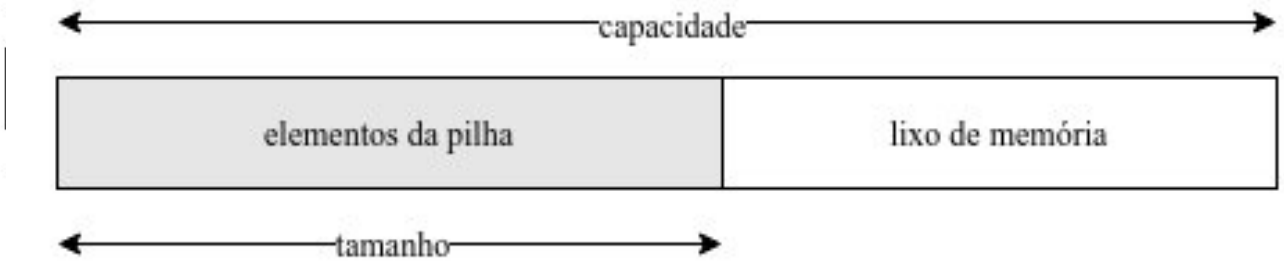
Vetor dinâmico - capacidade

E se `tamanho == capacidade`?

1) Criamos um novo espaço de memória e copiamos o conteúdo para lá com realloc

2) Atualizamos a nova capacidade

3) Atualizamos o ponteiro para os novos dados



Atividade prática

Implementação de Vetor dinâmico (Entrega)

1. Revisão de malloc
2. Compilação de programas com mais de um arquivo .c
3. Entender uso de um TAD a partir de exemplos de uso

Insper

www.insper.edu.br