Dominando Estruturas de Dados 1

Structs

Prof. Samuel Martins (Samuka)
@xavecoding @hisamuka

xavecoding

Structs

- Uma **struct** (ou **registro** ou ***record***) é um *"pacote"* de variáveis, que podem ter tipos diferentes;
- Visa representar grupos de dados que resultem em algo mais "concreto":
    - P. ex.: registro de alunos, automóveis, etc...
- Cada variável é um campo do registro;
- Em C, registros são conhecidos como ***structs*** (que vem de structure, do inglês);

Structs: Definição

```
struct [nome do registro]{
    tipo campo1;
    tipo campo2;
    ...
    tipo campoN;
} [uma ou mais variáveis];
```

- *[nome do registro]* e *[uma ou mais variáveis]* são **opcionais**;
- *[nome do registro]* é simplesmente o nome (não um tipo) da classe de registros do mesmo tipo
- Você já pode **declarar variáveis**, referentes a esta *struct*, colocando-as no lugar de ***[uma ou mais variáveis];***

Structs: Definição

- Vamos declarar uma *struct* para armazenar **alunos;**

Opção 1: Declara a struct Aluno

```
struct Aluno {
    char nome[100];
    int idade;
};

// declaração de uma variável
struct Aluno barney;
```

Structs: Definição

- Vamos declarar uma *struct* para armazenar **alunos**;

Opção 2: Define a struct Aluno e já declara variáveis deste tipo;

```
struct Aluno {
    char nome[100];
    int idade;
} barney, ted;
```

Structs: Definição

- Vamos declarar uma *struct* para armazenar **alunos**;

Opção 3: Define a struct sem nome e declara variáveis deste tipo;

```
struct {
    char nome[100];
    int idade;
} barney, ted;
```

- **Problema:**

Structs: Definição

- Vamos declarar uma *struct* para armazenar **alunos**;

Opção 3: Define a struct sem nome e declara variáveis deste tipo;

```
struct {
    char nome[100];
    int idade;
} barney, ted;
```

- **Problema:** Não será possível declarar novas variáveis deste tipo, apenas as variáveis barney e ted;

Structs: Acessando os Campos

- Para acessar um campo de um registro, utilize o **ponto:**

```c
struct Aluno {
    char nome[100];
    int idade;
};

int main() {
    // declaração de uma variável
    struct Aluno barney;

    // copia a string Barney para o campo nome
    strcpy(barney.nome, "Barney");
    barney.idade = 10;

    return 0;
}
```

Structs: Criando um Tipo de Dados

- Ou ainda, um jeito mais elegante:

```
typedef struct Aluno {
    char nome[100];
    int idade;
} TipoAluno;

struct Aluno barney;
TipoAluno ted;
```

- **Aluno** é opcional;
- Poderíamos omiti-lo, deixando apenas o *TipoAluno* para representar o tipo da struct

Structs: Criando um Tipo de Dados

- Vamos adotar a seguinte convenção no curso:

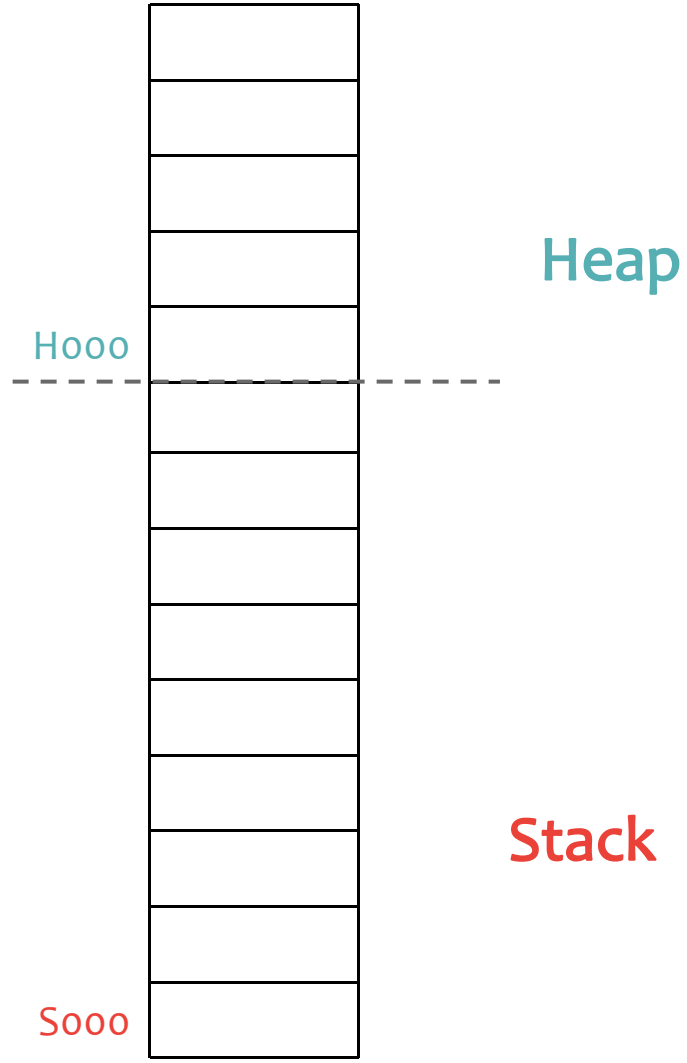
```
typedef struct _aluno {
    char nome[100];
    int idade;
} Aluno;

// vamos utilizar apenas o
// tipo definido
Aluno ted;
```

Structs: Alocação Estática

```c
typedef struct _aluno {
    char nome[100];
    int idade;
} Aluno;
```

```c
Aluno ted;
strcpy(ted.nome, "Ted");
ted.idade = 10;
```

```c
printf("sizeof(Aluno) = %lu bytes\n", sizeof(Aluno));
```

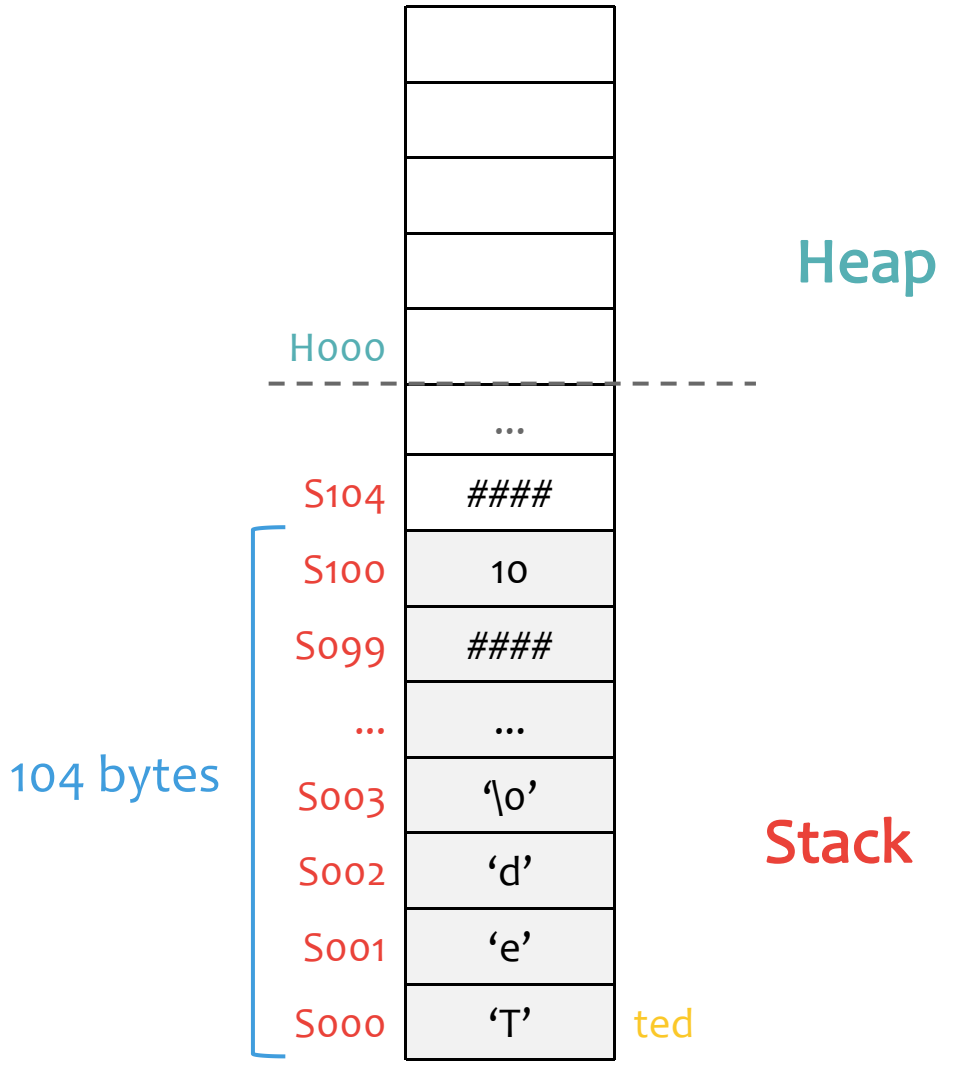


Structs: Alocação Estática

```c
typedef struct _aluno {
    char nome[100];
    int idade;
} Aluno;
```

```c
Aluno ted;
strcpy(ted.nome, "Ted");
ted.idade = 10;
```

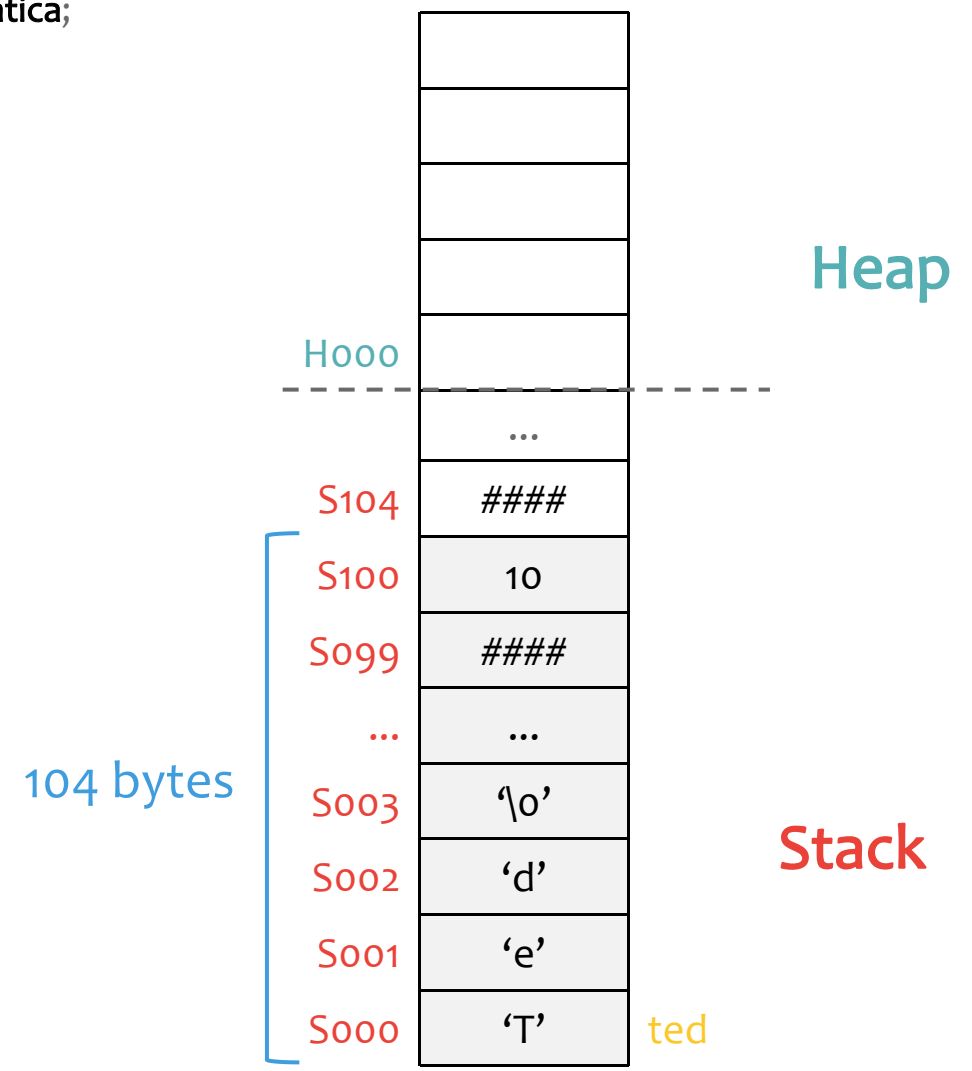
```c
printf("sizeof(Aluno) = %lu bytes\n", sizeof(Aluno));
```



Structs: Alocação Estática

- Podemos ainda **atribuir valores** aos campos de uma struct durante sua **alocação estática**;

```
Aluno ted = {.nome = "Ted", .idade = 10};
```



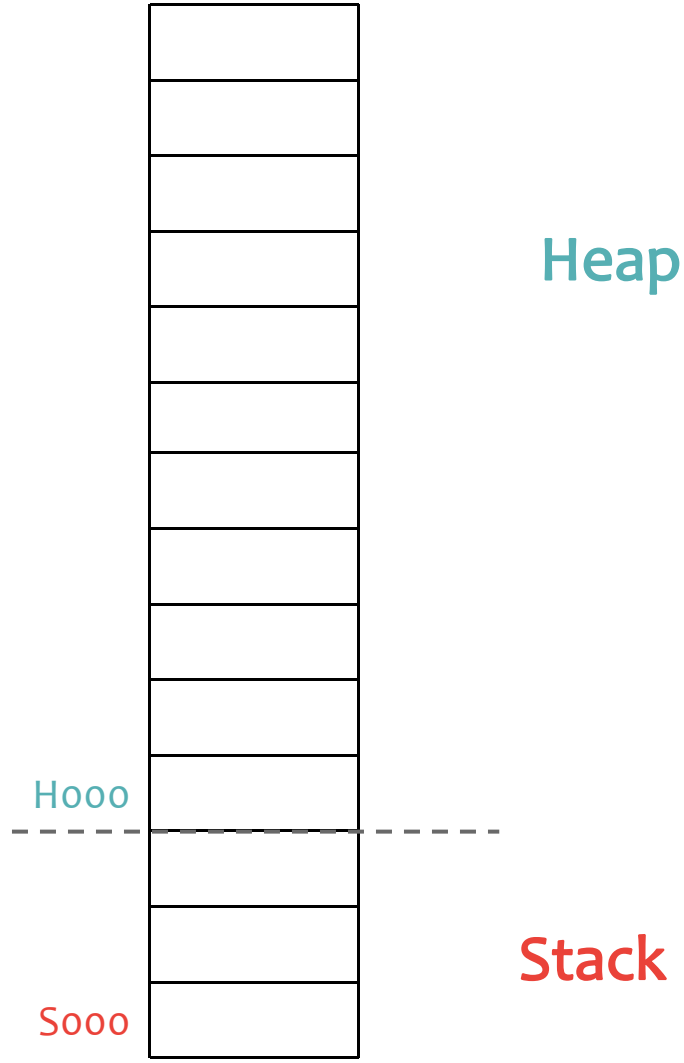
Structs: Alocação Dinâmica

- Podemos alocar **instâncias** de structs dinamicamente

```
Aluno *ted = (Aluno*) calloc(1, sizeof(Aluno));
```

- Pra acessar os campos de uma struct partindo de um **ponteiro**, usamos o **->**

```
strcpy(ted->nome, "Ted");
ted->idade = 10;
```



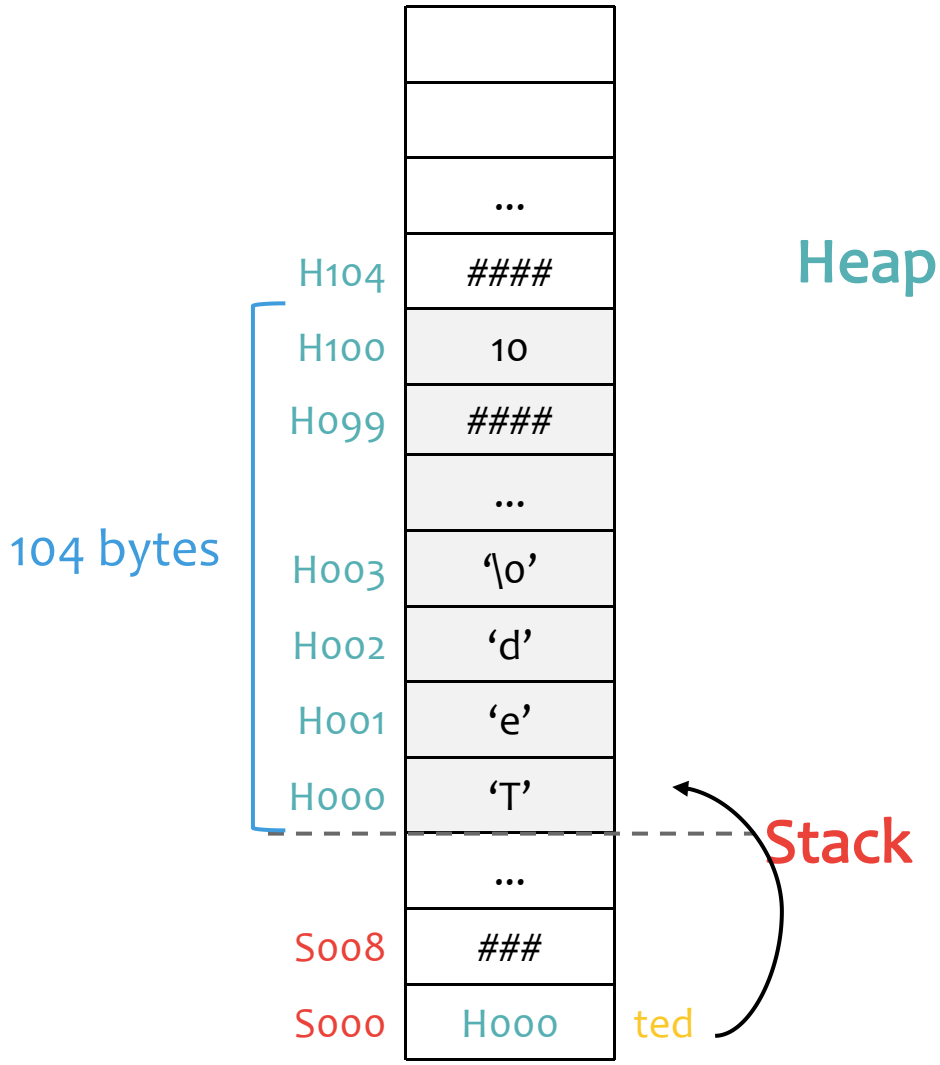
Structs: Alocação Dinâmica

- Podemos alocar **instâncias** de structs dinamicamente

```
Aluno *ted = (Aluno*) calloc(1, sizeof(Aluno));
```

- Pra acessar os campos de uma struct partindo de um **ponteiro**, usamos o **->**

```
strcpy(ted->nome, "Ted");
ted->idade = 10;
```



CRUD de Structs

- Considere que o Aluno possui **um livro favorito**, que, por simplificação, possui um título, número de páginas e preço.
  - Codifique a struct de Livro e adapte a struct de Aluno;
- Crie as funções de Criação (C), Delete (D) e de Impressão para a Struct Aluno e Livro
- Na função de Delete, garanta que o ponteiro é atribuído como NULL depois da desalocação

Vetores de Structs

Passando Structs como de Argumentos de Funções

- Passagem por valor
- Passagem por referência
- const

Exercíacios

Para os exercícios abaixo, simule a alocação e o comportamento das variáveis na memória RAM. Divida-a em memória Stack ou Heap se necessário.
Considere que o primeiro endereço de memória Stack disponível é ***S000****, e de memória Heap* ***H000****.*
Considere os seguintes tamanhos para cada tipo primário:
char *= 1 byte*
int *= 4 bytes*
long *= 8 bytes*
float *= 4 bytes*
double *= 8 bytes*
ponteiros *= 8 bytes*

Considere a estrutura abaixo para a resolução dos exercícios de 1 a 4.

```
typedef struct _livro {
    char titulo[100];
    float preco;
    int n_paginas;
} Livro;
```

```
Livro *CriaLivro(char titulo[], float preco, int n_paginas) { ... }
```

1. Qual a saída do programa abaixo que imprime o tamanho das variáveis? Explique sua resposta.

```
int main() {
    Livro livro1;
    Livro *livro2 = (Livro*) calloc(1, sizeof(Livro));

    printf("Tamanho livro: %ld bytes\n", sizeof(livro1));
    printf("Tamanho livro: %ld bytes\n", sizeof(livro2));

    return 0;
}
```

Exercíacios

Para os exercícios abaixo, simule a alocação e o comportamento das variáveis na memória RAM. Divida-a em memória Stack ou Heap se necessário.
Considere que o primeiro endereço de memória Stack disponível é ***S000****, e de memória Heap* ***H000****.*
Considere os seguintes tamanhos para cada tipo primário:
char *= 1 byte*
int *= 4 bytes*
long *= 8 bytes*
float *= 4 bytes*
double *= 8 bytes*
ponteiros *= 8 bytes*

Considere a estrutura abaixo para a resolução dos exercícios de 1 a 4.

```
typedef struct _livro {
    char titulo[100];
    float preco;
    int n_paginas;
} Livro;
```

```
Livro *CriaLivro(char titulo[], float preco, int n_paginas) { ... }
```

5. Analise o trecho de código abaixo e explique se os itens propostos funcionam ou não. Utilize a representação de memória se necessário.

```
Livro livro1  = {.titulo = "Harry Potter 1", 30.0, 250};
Livro *livro2 = CriaLivro("O Segredis de Cacilds", 10.0, 100);
```

a) printf("titulo1 = %s\n", livro1.titulo);
b) printf("titulo1 = %s\n", livro1->titulo);
c) printf("titulo1 = %s\n", &livro1->titulo);
d) printf("titulo1 = %s\n", (&livro1)->titulo);
e) printf("titulo2 = %s\n", livro2.titulo);
f) printf("titulo2 = %s\n", livro2->titulo);
g) printf("titulo2 = %s\n", *livro2.titulo);
h) printf("titulo2 = %s\n", (*livro2).titulo);
i) printf("titulo2 = %s\n", livro2[0].titulo);

Dominando Estruturas de Dados 1

Structs

Prof. Samuel Martins (Samuka)
@xavecoding @hisamuka

xavecoding