UNIVERSIDADE FEDERAL DO CEARÁ

DEPARTAMENTO DE COMPUTAÇÃO

Programação

(CK0226 – 2022.2)

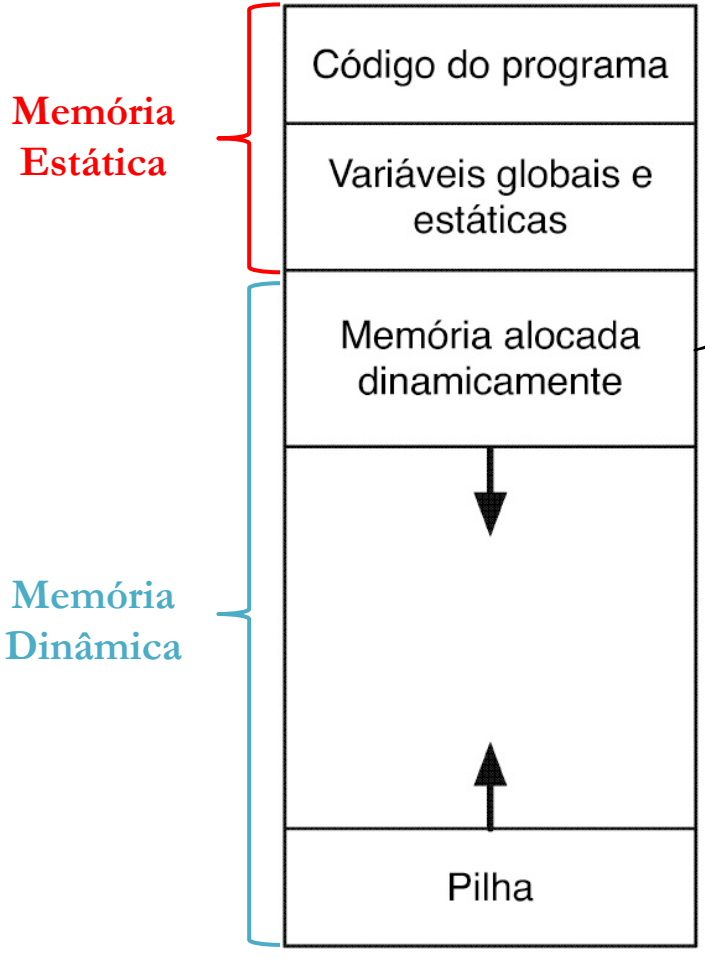
Curso: Ciência da Computação
Professor: Lincoln Souza Rocha
E-mail: lincoln@dc.ufc.br

Introdução à Linguagem C – Parte V

Sumário

- Representação da Memória
- Alocação/Liberação de Memória
- Tipo estrutura
- Definição de novos tipos
- Aninhamento de estruturas
- Vetores de estruturas
- Tipo união
- Tipo enumeração

Representação da Memória



É dessa memória que estamos falando!

Nota!

- "vazamento de memória" (**memory leak**)
- "estourou de pilha" (**stack overflow**)

Alocação Dinâmica

Em C, a manipulação dinâmica de memória é feita por meio de funções da biblioteca padrão "**`stdlib.h`**".

`int sizeof(<tipo>)` - retorna o número de bytes ocupado por um dado tipo informado via argumento da função.

`void* malloc(<#bytes>)` - recebe como parâmetro o número de bytes que se deseja alocar e retorna um ponteiro genérico para o endereço inicial da área de memória alocada, se houver espaço livre. Em caso contrário, retorna **`NULL`**.

Alocação Dinâmica

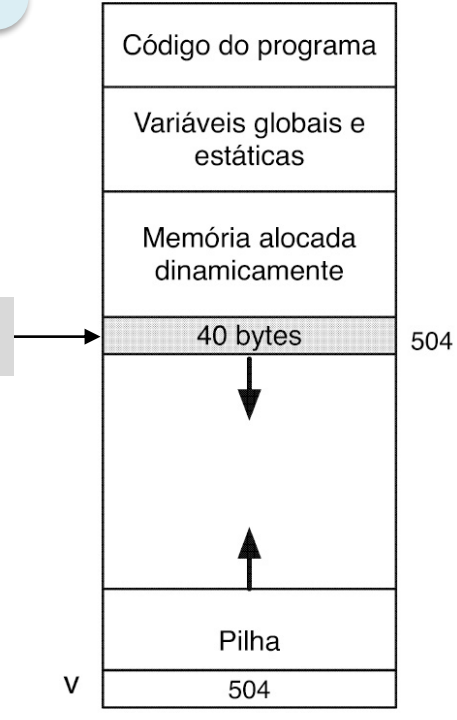
`void* realloc(void*, <#bytes>)` - permite realocar um vetor preservando o conteúdo dos elementos que permanecem válidos após a realocação.

`void free(*void)` - recebe como parâmetro um ponteiro para a área de memória que se quer liberar.

Alocação Dinâmica

Como alocar dinamicamente um vetor de inteiros com 10 posições de memória?

```
int *v = (int*) malloc(10 * sizeof(int));
```



Alocação Dinâmica

Tratando casos em que não existe memória suficiente!

```
(...)
int *v = (int*) malloc(10*sizeof(int));
(...)
if (v==NULL) {
    printf("Memoria insuficiente.\n");
    exit(1);
}
(...)
```

Alocação Dinâmica

Liberando memória ao final da operação!

```
(...)
int *v = (int*) malloc(10*sizeof(int));
(...)
if (v==NULL) {
    printf("Memoria insuficiente.\n");
    exit(1);
}
(...)
free(v);
(...)
```

Vetores: Média e Variância

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

float captura(int n, float* x);
float media (int n, float* x);
float variancia (int n, float* x, float m);

int main (void ) {
   int n; /* número de valores */
   float *x; /* vetor dos valores */

   printf("Entre com o numero de valores: ");
   scanf("%d", &n);
   x = (float *) malloc(n*sizeof(float));

  if (x == NULL) {
      printf("Memoria insuficiente.\n");
      exit(1);
   }
   captura(n, x);
   float m = media(n, x);
   float v = variancia(n, x, m);
   printf("Media: % f\ nVariancia: % f\ n", m, v);
   free(x);
   return 0;
}
```

alocação de vetor com n posições

liberação da memória

Vetores: Média e Variância

```
void captura (int n, float* x) {
   printf("Entre com os valores:\ n");
   for (int i=0; i<n; ++i)
      scanf("% f", & x[i]);
}

float media (int n, float* x) {
   float m = 0.0 f;
   for (int i=0; i<n; ++i)
      m += x[i];
   return m / n;
}

float variancia (int n, float* x, float m) {
   float v = 0.0 f;
   for (int i=0; i<n; ++i)
      v += (x[i] - m) * (x[i] - m);
   return v / n;
}
```

Tipo Estrutura

É um tipo de dado com campos compostos de tipos mais simples onde elementos são acessados através do operador de acesso "ponto" (.).

```
struct ponto {
float x;
float y;
};
```

declara o ponto do tipo struct

```
(...)
```

```
struct ponto p;
```

declara a variável p com tipo struct ponto

```
(...)
```

```
p.x = 10.0;
p.y = 5.0;
```

acessa os campos da variável p

Tipo Estrutura

```
#include <stdio.h>

struct ponto {
    float x;
    float y;
};

int main (void ) {
    struct ponto p;
    printf("Digite as coordenadas do ponto(x y): ");
    scanf("%f %f", &p.x, &p.y);
    printf("O ponto fornecido foi: (%.2f,%.2 f)\ n", p.x, p.y);
    return 0;
}
```

Basta escrever &p.x em lugar de &(p.x). O operador de acesso ao campo da estrutura tem precedência sobre o operador "endereço de".

Tipo Estrutura

Ponteiros para estruturas:

- Acesso ao valor de um campo x de uma variável estrutura p: **`p.x`**
- Acesso ao valor de um campo x de uma variável ponteiro pp: **`pp->x`**
- Acesso ao endereço do campo x de uma variável estrutura pp: `&`**`pp->x`**

```
struct ponto *pp;

/* formas equivalentes de acessar o valor de um campo x */

(*pp).x = 12.0;

//OU

pp->x = 12.0;
```

Tipo Estrutura

Passagem de estruturas para funções (Por Valor)

- Análoga à passagem de variáveis simples
- Função recebe toda a estrutura como parâmetro:
  - Função acessa a cópia da estrutura na pilha
  - Função não altera os valores dos campos da estrutura original
  - Operação pode ser custosa se a estrutura for muito grande

```
void imprime (struct ponto p) {
    printf("O ponto fornecido foi: (%.2f,%.2f)\ n", p.x, p.y);
}
```

Tipo Estrutura

Passagem de estruturas para funções (Por Referência)

- Apenas o ponteiro da estrutura é passado, mesmo que não seja necessário alterar os valores dos campos dentro da função.

```
void imprime (struct ponto* p) {
    printf("O ponto fornecido foi: (%.2f,%.2f)\ n", p->x, p->y);
}
```

Tipo Estrutura

Alocação dinâmica de estruturas

- O tamanho do espaço de memória alocado dinamicamente é dado pela função **sizeof** aplicado sobre o tipo estrutura
- A função **malloc** retorna o endereço do espaço alocado, que é então convertido para o tipo ponteiro da estrutura

```
struct ponto* p;
p = (struct ponto*) malloc (sizeof(struct ponto));
(...)
p->x = 12.0;
(...)
```

Definição de Novos Tipos

A palavra reservada **typedef** permite criar nomes de tipos. Útil para abreviar nomes de tipos e para tratar tipos complexos.

```
typedef unsigned char UChar; /* o tipo char sem sinal*/
typedef int* PInt; /* um tipo ponteiro para int */
typedef float Vetor[4]; /* um tipo vetor de 4 posições */

(…)

UChar u;
PInt p;
Vetor v;
v[0] = 3;
```

Definição de Novos Tipos

Exemplo com tipo estrutura:

- **ponto** representa uma estrutura com 2 campos do tipo float
- **Ponto** representa o tipo da estrutura ponto
- **PPonto** representa o tipo ponteiro para a estrutura Ponto

```
struct ponto {
    float x;
    float y;
};

typedef struct ponto Ponto;
typedef struct ponto *PPonto;
```

Definição de Novos Tipos

Exemplo com tipo estrutura:

- **ponto** representa uma estrutura com 2 campos do tipo float
- **Ponto** representa o tipo da estrutura ponto
- **PPonto** representa o tipo ponteiro para a estrutura Ponto

```
struct ponto {
    float x;
    float y;
};

typedef struct ponto Ponto, *PPonto;
```

Definição de Novos Tipos

Exemplo com tipo estrutura:

- **ponto** representa uma estrutura com 2 campos do tipo float
- **Ponto** representa o tipo da estrutura ponto
- **PPonto** representa o tipo ponteiro para a estrutura Ponto

```
struct ponto {
    float x;
    float y;
} Ponto;

typedef struct ponto *PPonto;
```

Aninhamento de Estruturas

Os campos de uma estrutura podem ser outras estruturas.

```
struct circulo {
    Ponto p;
    float r;
};

typedef struct circulo Circulo;
```

Aninhamento de Estruturas

```
typedef struct ponto {
      float x;
      float y;
} Ponto;

typedef struct circulo {
      Ponto p;
      float r;
} Circulo;

float distancia(Ponto* p, Ponto* q) {
     float d = sqrt((q->x-p->x)*(q->x-p->x) + (q->y-p->y)*(q->y-p->y));
     return d;
}

int interior(Circulo* c, Ponto* p) {
     float d = distancia(&c->p, p);
     return (d < c->r);
}
```

Aninhamento de Estruturas

```
int main (void) {
    Circulo c; Ponto p;
    printf("Digite as coordenadas do centro e o raio do circulo:\n");
    scanf("%f %f %f", &c.p.x, &c.p.y, &c.r);
    printf("Digite as coordenadas do ponto:\n");
    scanf("%f %f", &p.x, &p.y);
    printf("Pertence ao interior = %d\n", interior(&c,&p));
    return 0;
}
```

Vetores e Estruturas

```
Ponto centro_geometrico (int n, Ponto* vetor) {
    Ponto centro = {0.0f, 0.0 f};
    for (int i=0; i<n; ++i) {
        centro.x += vetor[i].x;
        centro.y += vetor[i].y;
    }
    centro.x /= n;
    centro.y /= n;
    return centro;
}

int main (void) {
    Ponto ponto[3] = {{1.0 ,1.0},{5.0 ,1.0},{4.0 ,3.0}};
    Ponto centro = centro_geometrico(3, ponto);
    printf(”Centro Geométrico = (%.2f,%.2 f)\n", centro.x, centro.y);
    return 0;
}
```

Tipo União (Union)

Localização de memória compartilhada por diferentes variáveis, que podem ser de tipos diferentes. As uniões usadas para armazenar valores heterogêneos em um mesmo espaço de memória.

```
union exemplo {
   int i;
   char c;
};

union exemplo v;
v.i = 1;
v.c = 'D';
```

Tipo Enumeração (Enum)

Declara uma enumeração, ou seja, um conjunto de constantes inteiras com nomes que especifica os valores que uma variável daquele tipo pode ter. Oferece uma forma mais elegante de organizar valores constantes.

```
enum bool {
  TRUE = 1,
  FALSE = 0
};

typedef enum bool Bool;

Bool resultado;
```

UNIVERSIDADE FEDERAL DO CEARÁ

DEPARTAMENTO DE COMPUTAÇÃO

Programação

(CK0226 – 2022.2)

Curso: Ciência da Computação
Professor: Lincoln Souza Rocha
E-mail: lincoln@dc.ufc.br