Alocação dinâmica de memória, Struct

1. Alocação dinâmica de memória

A alocação dinâmica de memória permite o gerenciamento dos blocos de memória durante a execução do programa. Enquanto na alocação estática o tamanho do bloco é definido durante o processo de compilação, na alocação dinâmica o tamanho do bloco é especificado em tempo de execução. Isso permite que o programador faça a alocação de memória necessária para seus dados durante o processo de execução.

Dessa forma, a alocação de memória garante a manipulação de dados cujo tamanho não é conhecido previamente, a criação de estruturas de dados flexíveis (como listas encadeadas, árvores ou grafos) e a otimização do uso da memória, com alocação e liberação sob demanda e conforme as instruções programadas para a execução de uma determinada tarefa.

2. Funções para alocação dinâmica da memória

As funções estão disponíveis na biblioteca <stdlib.h>.

- malloc (memory allocation): a função é utilizada para alocar um bloco de memória de tamanho especificado em bytes e retorna um ponteiro para o início do bloco alocado.

Importante:

1) A função malloc não inicializa os valores dentro do bloco, portanto eles irão conter um valor indeterminado.
2) Caso não haja espaço suficiente para a alocação na memória, a função retornará um ponteiro nulo.

Exemplo 1:

```
// Alocação de um único inteiro
int *p1 = malloc(sizeof(int));
```

Exemplo 2:

```
// Alocação de um array de inteiros
int *p2 = malloc(5 * sizeof(int));
for (int i = 0; i < 5; i++) {
  p2[i] = i + 6;
}
```

- calloc: a função também é utilizada para alocar um bloco de memória, mas para um número específico de elementos. O tamanho em bytes para cada um deve ser especificado, e os valores são inicializados com 0 (zero).
  Apesar de realizar a inicialização dos valores, este processo torna a função mais lenta do que a malloc.

Exemplo 3:

```
// Alocação de um array de inteiros
int *array = calloc(10, sizeof(int));
```

- realloc: o método é utilizado para redimensionar um bloco de memória previamente alocado. Para isso, devem ser especificados o ponteiro para o bloco alocado e o novo tamanho. Caso o tamanho informado seja maior do que o tamanho original, um novo bloco de memória será alocado e os dados do bloco antigo serão copiados para o novo. O método retorna o ponteiro para o novo bloco.

Exemplo 4:

```
int *array;
// Alocação de um array com tamanho 10
array = malloc(10 * sizeof(int));
// Realocação do array para tamanho 20
array = realloc(array, 20 * sizeof(int));
```

3. Liberação de memória

Para realizar a liberação da memória alocada dinamicamente, utilize a função free(ponteiro). Ela é responsável por desalocar a porção de memória alocada e informar o sistema operacional que o bloco de bytes apontado pelo "ponteiro" está liberado para reutilização.

4. Boas práticas de programação

1) Inicialize os ponteiros alocados de forma dinâmica para evitar valores indeterminados.
2) Verifique se o processo de alocação de memória foi realizado corretamente.

5. Struct em C

A struct é um agrupamento de diferentes tipos de variáveis em uma única entidade, ou seja, sob um único nome. Ela possibilita a representação de objetos complexos por meio de um tipo de dado composto que pode conter campos de tipos diferentes. Isso simplifica a manipulação de informações relacionadas.

Exemplo de struct:

```
struct Estudante {
  int id;
  char nome[50];
  float media;
};

struct Estudante e1;
e1.id = 1010;
strcpy(e1.nome, "Edilson");
e1.media = 9.8;
```

Exemplo 5 (struct com ponteiro):

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

```c
// ** Exemplo de struct com ponteiro **
// Definindo a struct Pessoa
struct Pessoa {
  char nome[50];
  int idade;
};

int main() {
  // Alocação dinâmica de uma struct Pessoa
  struct Pessoa* pessoaPtr = malloc(sizeof(struct Pessoa));

  // Preenchimento dos campos da struct usando ponteiros
  strcpy(pessoaPtr->nome, "José");
  pessoaPtr->idade = 23;

  // Exibição dos dados
  printf("Nome: %s\n", pessoaPtr->nome);
  printf("Idade: %d\n", pessoaPtr->idade);

  // Liberação da memória alocada
  free(pessoaPtr);

  return 0;
}
```

Exemplo 6 (array de structs com ponteiro):

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

// ** Exemplo de array de structs com ponteiro **
// Definindo a struct Pessoa
struct Pessoa {
  char nome[50];
  int idade;
};

int main() {
  // Criando um array de structs "Pessoa" usando ponteiros
  struct Pessoa *pessoas = malloc(3 * sizeof(struct Pessoa));

  // Preenchimento dos campos das structs
```

```
    strcpy(pessoas[0].nome, "Alice");
    pessoas[0].idade = 19;

    strcpy(pessoas[1].nome, "José");
    pessoas[1].idade = 38;

    strcpy(pessoas[2].nome, "Maria");
    pessoas[2].idade = 27;

    // Exibição os dados usando ponteiros
    for (int i = 0; i < 3; i++) {
      printf("Pessoa %d: Nome: %s, Idade: %d\n", i + 1, (pessoas + i)->nome, (pessoas + i)->idade);
    }

    // Liberação da memória alocada
    free(pessoas);

    return 0;
}
```

6. Exercícios

6.1. Crie uma struct para representar a data de hoje de maneira que o valor 01/09/2023 seja impresso.

6.2. Analise as formas 1 e 2 para alocação dinâmica de um inteiro. Na primeira, é utilizada a conversão (int *). A partir de pesquisas na Internet e leitura do website (https://c-faq.com/malloc/cast.html), responda: As duas formas são equivalentes? Uma ou outra é melhor? Existe alguma boa prática de programação para o caso?

1: `int *nro = (int *) malloc(sizeof(int));`

2: `int *nro = malloc(sizeof(int));`

6.3. Escreva um programa para cadastrar pessoas (considerando nome, profissão e idade) que realizam trabalho voluntário em 5 (cinco) países (Austrália, Canadá, Irlanda, Nova Zelândia e Suíça). Após isso, determine o número médio de pessoas que realizam trabalho voluntário nesses países e também o país no qual existem mais pessoas colaborando com o trabalho voluntário.

Utilize struct, alocação dinâmica de memória e ponteiro.

6.4. Escreva um programa que recebe como entrada a quantidade de aulas dadas para uma determinada disciplina, a quantidade de aulas frequentadas pelo estudante e as 4 (quatro) notas obtidas por ele nesta disciplina. Com base nos dados informados, calcule e mostre a média do estudante e seu percentual de frequência. O programa também deve exibir a situação do estudante ("Aprovado", "Reprovado" ou "IFA") de acordo com os seguintes critérios:

Aprovação: Se média >= 6.0 e frequência >= 75%
Reprovação: Se média < 4.0 ou frequência < 75%
IFA: Se média >= 4.0 e < 6.0 e frequência >= 75%

Utilize struct, alocação dinâmica de memória e ponteiro.

6.5. Escreva um programa que simule o cadastro e atendimento de pessoas em um consultório médico. Devem ser oferecidas duas opções para o atendente: a primeira realiza o cadastro de uma pessoa, considerando-se o nome e a idade. A segunda opção possibilita que uma pessoa seja chamada para a consulta. Pessoas com mais de 60 anos possuem prioridade, ou seja, devem ser atendidas antes. Uma vez que todos os pacientes com prioridade forem chamados, os demais pacientes podem ser chamados, mantendo-se a ordem de chegada.

Utilize struct, alocação dinâmica de memória e ponteiro.

6.6. Escreva um programa para controlar a venda de ingressos para um jogo amistoso da seleção brasileira. Existem quatro tipos de ingressos: setor amarelo, setor azul, setor branco e setor verde. Os preços assim como a carga de ingressos disponíveis para cada setor estão especificados na tabela abaixo. É importante ressaltar que estudantes e aposentados têm direito à meia entrada. O torcedor deve informar seu nome e solicitar a quantidade de ingressos que deseja comprar. Após isso, o sistema irá verificar se existem ingressos disponíveis e se o torcedor pode pagar meia entrada. Em caso afirmativo, o tipo dos ingressos comprados, quantidade, valor unitário e valor total devem ser exibidos. Além disso, o programa deve exibir todas as vendas realizadas até o momento (considerando o nome do comprador, quantidade de ingressos comprados e valor total da compra).

Tipo	**Valor**	**Qtde Ingressos**
Setor Amarelo	R$ 180	20
Setor Azul	R$ 100	30
Setor Branco	R$ 60	40
Setor Verde	R$ 350	10

Utilize struct, alocação dinâmica de memória e ponteiro.

6.7. Crie um programa que gerencie um estacionamento rotativo de carros. O programa deve armazenar a descrição do carro, a placa, o horário de entrada e o horário de saída (despreze os minutos). O estacionamento cobra X reais pela primeira hora de permanência com o automóvel e X/3 pelas demais horas. Além disso, é fornecido um desconto para o pagamento de acordo com a tabela abaixo:

Valor	**Desconto (%)**
Até R$ 20 (inclusive)	5
Entre R$ 20 e R$ 50 (inclusive)	10
Acima de R$ 50	20

O programa deve exibir um relatório contendo as seguintes informações:
a) Tipo do carro.
b) Placa.
c) Hora da Entrada.
d) Hora da Saída.
e) Valor Pago.

Utilize struct, alocação dinâmica de memória e ponteiro.

Desafio:

- Realize um estudo sobre o conceito de lista linear. Após isso, implemente um programa que leia e imprima 5 (cinco) números. Na primeira versão, utilize array. Na segunda versão, utilize lista linear.