ESTRUTURA DE DADOS I

Tipos Abstratos de Dados

Profª Amanda Tameirão

Objetivos da aula de hoje

- Compreender o conceito de novos tipos;
- Utilizar typedef;
- Diferenciar struct, union e enum;
- Aplicar soluções com novos tipos.

Conceito

- Imagine que você precisa definir os dados de um cliente:

  Nome

  Idade

  Valor de compra

  Gênero

Conceito

- Você obviamente declararia variáveis para cada item:

  Nome - String com 30 posições

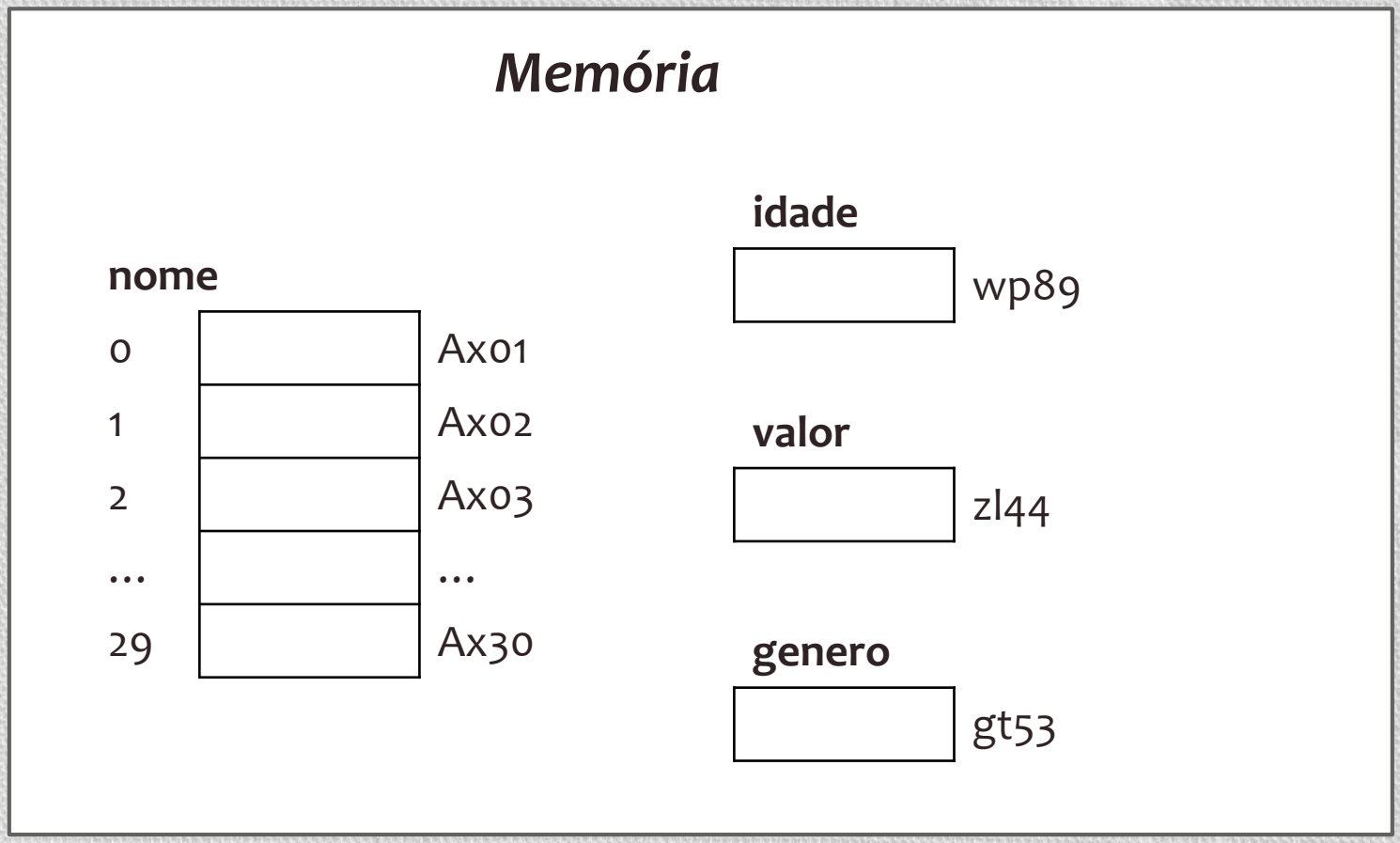
  Idade - Inteiro

  Valor de compra - Real

  Gênero - char

Conceito

- Na memória, o armazenamento seria dessa forma



Conceito

- Agora imagine-se criando cada item deste para 20 clientes:

Nome - String com 30 posições

Idade - Inteiro

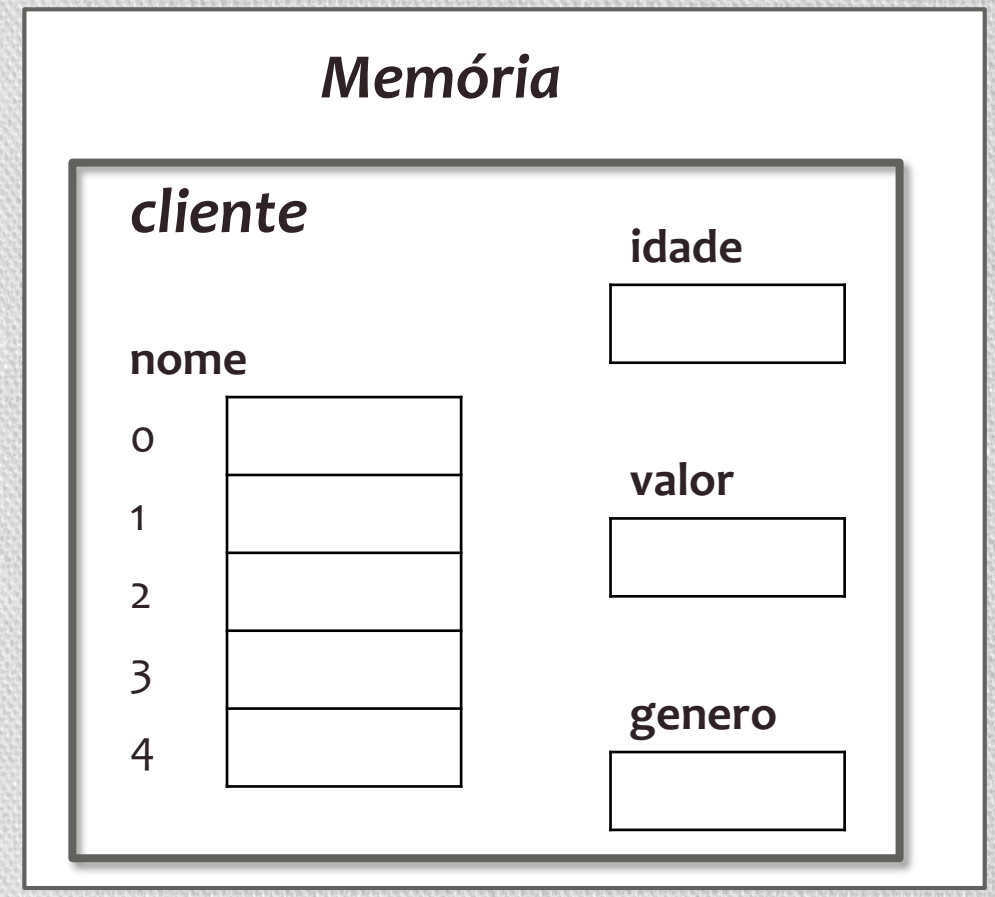
Valor de compra - Real

Gênero - char

struct

Estrutura

- Para isto existem as estruturas, elas permitem encapsular informações.



A memória utilizada considerará o número de bytes necessário para cada membro.

Portanto, imagine que, neste exemplo, o início da memória seria AX01 até a quantidade necessária para amarzenar tudo.

Estruturas

Reúne vários tipos em uma mesma estrutura.

Sintaxe:

```
struct nome_da_estrutura {
        tipo membro_1;
        tipo membro_2;
        ...
        tipo membro_n;
};
```

Estruturas

Para o exemplo do cliente teríamos

Sintaxe:

```
struct dadosCliente {
        char nome[30];
        int idade;
        float valor;
        char genero;
};
```

Atenção – Estruturas são modelos

Não é possível utilizá-las diretamente (é apenas uma abstração), por isso é preciso declarar para criar o item a ser utilizado.

```
#include <stdio.h>
int main() {
        struct dadosCliente cli; //Declarar uma estrutura

        cli.idade = 23; //Acessar membro número

        printf("Digite o valor liberado para o cliente: ");
        scanf("%f", &cli.valor);
        return 0;
 }
```

Observe que...

A estrutura completa possui os membros declarados, mas, uma vez acessado um dos membros, você deverá trabalhar com aquele tipo específico.

Por isso, quando aplicamos o comando scanf para o campo valor, utilizamos %f, afinal, este membro é do tipo float.

Sempre

Observe que sempre que declarar variáveis ou funções com este tipo deverá utilizar…

struct dadosCliente

Será que podemos melhorar isto? Facilitar?

typedef

Definição

Permite a declaração de um novo tipo, a partir de dados já declarados.

Sintaxe:

typedef <características> <novo_nome>;

Exemplo 01

typedef int inteiro;

Dessa forma o tipo **inteiro** poderá ser utilizado sem problemas, com todas as características do tipo int.

Aplicação – Exemplo 01

```
#include <stdio.h>
typedef int inteiro; //Assim estamos simplesmente renomeando

int main() {
        inteiro i;
        i = 85;
        printf("\nNúmero atribuído %i", i);
        printf("Digite um número inteiro: ");
        scanf("%i", &i);
        printf("\nNúmero digitado %i", i);
        return 0;
 }
```

Exemplo 02

typedef int inteiro[5];

Dessa forma o tipo **inteiro** poderá ser utilizado sendo um vetor de int com cinco posições, e terá todas as características referentes a ele.

Aplicação – Exemplo 02

```
#include <stdio.h>
typedef int inteiro[5];

int main() {
        inteiro i;
        int x;
        for (x = 0; i < 5; i++) {
                i[x]  = I * 10;
                printf("\nValor atribuído %i", i[x]);
        }
        return 0;
 }
```

Exemplo 03

typedef int* inteiro;

Dessa forma o tipo **inteiro** poderá ser utilizado sendo um ponteiro do tipo int.

Aplicação – Exemplo 03

```
#include <stdio.h>
typedef int* inteiro;

int main() {
        inteiro i;
        int x = 77;
        i = &x;
        *i *= 100;
        printf("\nValor atribuído %i", *i);

        return 0;
 }
```

Voltando ao struct

Estruturas

Sabendo que o typedef cria um novo tipo, portanto, podemos utilizá-lo com a estrutura de cliente

```
typedef struct  dadosCliente dadosCliente;


struct dadosCliente {
        char nome[30];
        int idade;
        float valor;
        char genero;
    };
```

Agora este tipo estará definido

E todo o código o reconhecerá como válido.

```
int main() {
        dadosCliente cli; //Declarar uma estrutura

        cli.idade = 23; //Acessar membro número

        printf("Digite o valor liberado para o cliente: ");
        scanf("%f", &cli.valor);
        return 0;
 }
```

Outra forma de declarar este tipo

```
typedef struct {
        char nome[30];
        int idade;
        float valor;
        char genero;
 } dadosCliente ;
```

Neste caso, você declara a estrutura junto com o typedef, e ele possui a declaração final.

Também posso ter uma struct em outra

Você poderá definir uma struct dentro de outra, assim, será possível acessar seus membros.

Exemplo:

```
typedef struct dadosEndereco endereco ;
    struct dadosEndereco{
        char rua[20];
        int numero;
        char bairro[20];
        int cep;
    }
```

A struct endereço estará na struct cliente

```
struct dadosCliente{
        char nome[30];
        int idade;
        float valor;
        char genero;
        endereco end;
 };
```

Assim, cada cliente terá seu endereço, e portanto terá os dados referentes a ele.

Importante

Declarar um novo tipo te permite fazer o que você quiser com ele:

- Definir variáveis
- Declarar como retorno de função
- Declarar como ponteiro,
- Etc...

Além disto, a estrutura poderá conter qualquer tipo (primitivo ou não), inclusive ponteiros destes tipos.

union

União

Serve para agrupar tipos diferentes em um mesmo item, porém, em uma união, os membros nunca poderão ser utilizados ao mesmo tempo.

Sintaxe:

```
union nome_da_união{
        tipo membro_1;
        tipo membro_2;
        ...;
        tipo membro_N;
    };
```

União

Também pode ser utilizado com o typedef

```
typedef union valores valores;

union valores {
        int valorInt;
        double valorDouble;
    };
```

União

```
typedef union valores valores;

union valores {
        int valorInt;
        double valorDouble;
    };
```

Neste exemplo, o tipo int armazena 4 bytes e o tipo double armazena 8 bytes, porém, você **NÃO** terá 12 bytes disponíveis para utilização, terá apenas 8 bytes, assim, quando precisar utilizar o double terá 8 bytes e quando precisar utilizar o int terá 4 bytes (destes 8).

Somente 1 espaço disponível!!!

enum

Enumeração

Coloca uma sequencia numérica em itens restritos.

Sintaxe:

```
enum nome_da_enum{
        tipo membro_1;
        tipo membro_2;
        ...;
        tipo membro_N;
   };
```

Enumeração

Também pode ser utilizado com o typedef.

typedef enum diasSemana semana;

enum semana {*segunda, terca, quarta, quinta, sexta, sabado, domingo*};

Aqui ele determinará que:

segunda = 0

terca = 1

quarta = 2

quinta = 3

sexta = 4

sabado = 5

domingo = 6

Enumeração

typedef enum {*segunda = 1, terca, quarta, quinta, sexta, sabado, domingo*} semana;

O default é começar com zero, mas, se for necessário começar com outro valor indique (conforme acima), e será gerado uma sequencia de um em um.

Aqui ele determinará que:

segunda = 1

terca = 2

quarta = 3

quinta = 4

sexta = 5

sabado = 6

domingo =7

Agenda

- Desenvolver o código da página 103 e 104 da apostila, referente a declaração de novos tipos. Entrega até domingo, 19/04 (2 pontos).

- A correção será realizada na próxima aula.

EDI

Aula de hoje

Falamos sobre novos tipos.

EDI

Dúvidas

Dúvidas deverão ser postadas no ambiente do Classroom para conversarmos sobre o assunto.