Programação de Computadores

Aula #02

Revisão de Conceitos

Qual é a relação da memória com os tipos de dados? O que representa uma alocação dinâmica? O que são *structs*? Quais são as estruturas de dados para revisar no contexto da disciplina?

Ciência da Computação – BCC e IBM – 2024/01
Prof. Vinícius Fülber Garcia

Memória

O QUE É MEMÓRIA?

Memória e Tipos de Dados

O QUE É MEMÓRIA?

Essa resposta pode variar dependendo do contexto em que ela é feita!

Em grau de *hardware*, você pode ouvir:

"Componente físico utilizado para armazenar *bits*"

Em grau de programação de baixo nível, você pode ouvir:

"Um conjunto de *bytes* armazenados, representando dados"

Em grau de programação (um pouco mais) alto nível:

"Um conjunto de dados tipados armazenados"

Memória e Tipos de Dados

Até o momento, nossa **programação foi bastante focada nos tipos de dados** (sejam eles nativos ou abstratos) e cada tipo, normalmente, serve apenas para a sua finalidade teórica.

Porém, nesta disciplina, teremos vários momentos em que trabalharemos com arquivos binários, gravação e leitura de registros, operações com bits, ponteiros *void* e para funções...

REMOÇÃO DE CAMADAS DE ABSTRAÇÃO DE MEMÓRIA

Memória e Tipos de Dados

Vamos começar com os tipos nativos!

QUAL É O MENOR (EM BYTES) TIPO DISPONÍVEL EM C?

QUAL É O MAIOR (EM BYTES) TIPO DISPONÍVEL EM C?

Memória e Tipos de Dados

Vamos começar com os tipos nativos!

QUAL É O MENOR (EM BYTES) TIPO DISPONÍVEL EM C?

char (1 byte)

QUAL É O MAIOR (EM BYTES) TIPO DISPONÍVEL EM C?

long double (de 12 a 16 bytes)

Memória e Tipos de Dados

Vamos começar com os tipos nativos! Em geral (AMD64)...

char:	1 byte
short:	2 bytes
int:	4 bytes
float:	4 bytes
long long:	8 bytes
double:	8 bytes
long double:	16 bytes

Memória e Tipos de Dados

MAS O QUE VOCÊ QUER DIZER COM TUDO ISSO?

O que se busca demonstrar é que tipos de dados são apenas facilidades providas pelo nosso compilador/interpretador.

No fundo, tudo o que temos são sequências de bytes.

Memória e Tipos de Dados

Então... podemos compor um tipo utilizando outro?

SIM!

char:	1 byte	1 char	--	--	
short:	2 bytes	2 char	1 short	--	
int:	4 bytes	4 char	2 short	1 int	
float:	4 bytes	4 char	2 short	1 int	...
long long:	8 bytes	8 char	4 short	2 int	
double:	8 bytes	8 char	4 short	2 int	
long double:	16 bytes	16 char	8 short	4 int	

Memória e Tipos de Dados

MAS O QUE DIFERENCIA, POR EXEMPLO, UM *INT* DE UM *FLOAT*?

No geral, ambos são formados por quatro bytes. Porém, o que diferencia os dois é a forma de interpretar esses quatro bytes!

- Inteiro (*int*): 31 bits indicam o valor; 1 bit que indica o sinal
- Real (*float*): 23 bits indicam o valor; 8 bits indicam o expoente; 1 bit indica o sinal (IEEE 754)

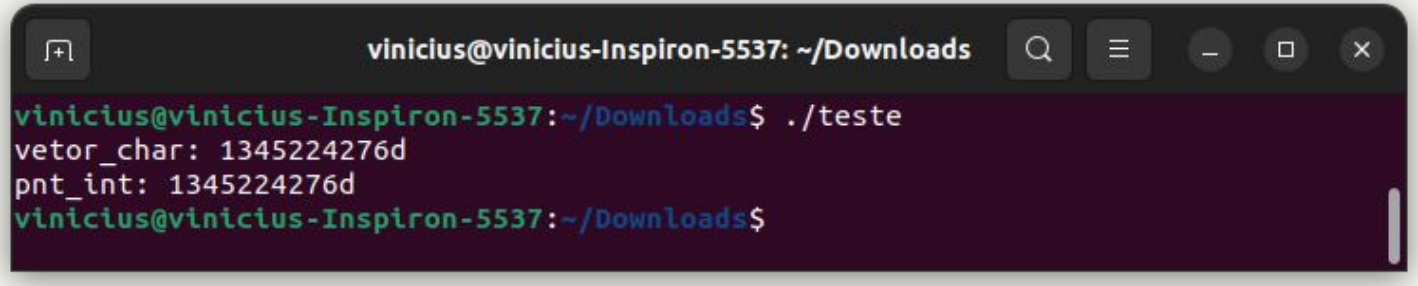
Memória e Tipos de Dados

```
1)  #include <stdlib.h>
2)  #include <stdio.h>
3)  int main (){
4)   unsigned char vetor_char[4] =
    {0,0,0,0};
5)   unsigned int *pnt_int;
6)   pnt_int = (unsigned int*) vetor_char;
7)   printf("vetor_char: %ud\n",
    vetor_char);
8)   printf("pnt_int: %ud\n", pnt_int);
9)   return 0;
10)  }
```

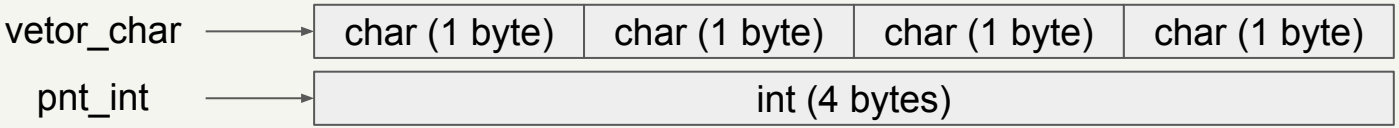
Observe o programa ao lado e responda às seguintes perguntas:

- Esse programa compila?
- Qual é a relação entre o ponteiro *vetor_char* e o ponteiro *pnt_int*?
- O que será exibido na tela?

Memória e Tipos de Dados



Note que os ponteiros **apontam para a mesma região de memória**. Porém, a forma de interpretar os dados armazenados nos bytes dessa região de memória mudam conforme o tipo do ponteiro.



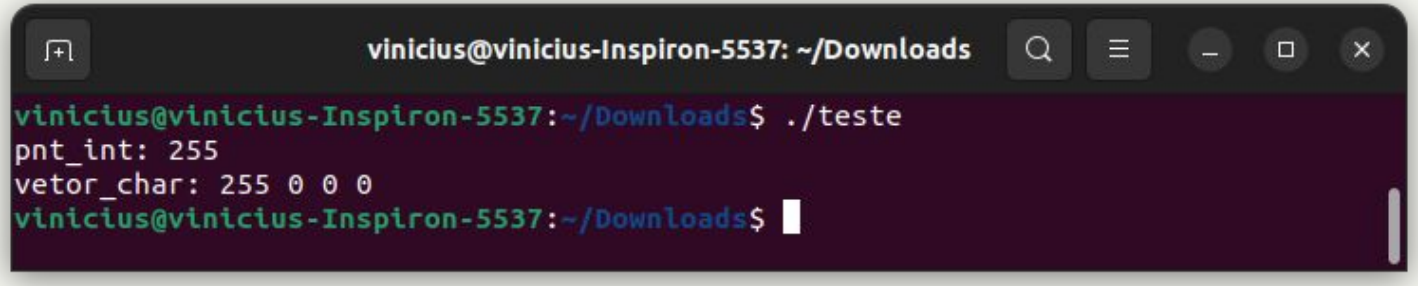
Memória e Tipos de Dados

```
1)  #include <stdlib.h>
2)  #include <stdio.h>
3)  int main (){
4)   unsigned char vetor_char[4] = {0,0,0,0};
5)   unsigned int *pnt_int;
6)   pnt_int = (unsigned int*) vetor_char;
7)   *pnt_int = 255;
8)   printf("pnt_int: %d", *pnt_int);
9)   printf("\nvetor_char: %d %d %d %d \n",
    vetor_char[0],//
10)         vetor_char[1], vetor_char[2], vetor_char[3]);
11)  return 0;
12)  }
```

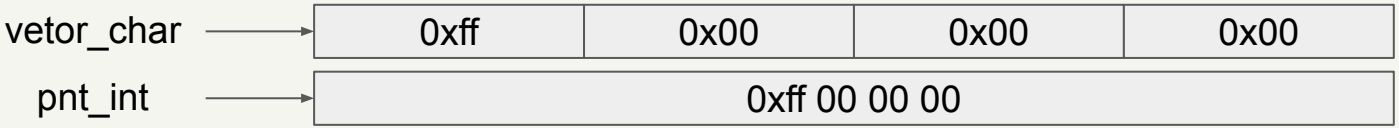
Observe o programa ao lado e responda às seguintes perguntas:

- O que exibe **pnt_int*?
- O que exibe cada posição do vetor *vetor_char*?

Memória e Tipos de Dados



O primeiro *char* do vetor (índice zero) coincide com o valor atribuído ao inteiro. Isso acontece já que **o índice zero (0) do vetor representa os oito (8) bits menos significativos do dado do tipo inteiro** (*little endian*).



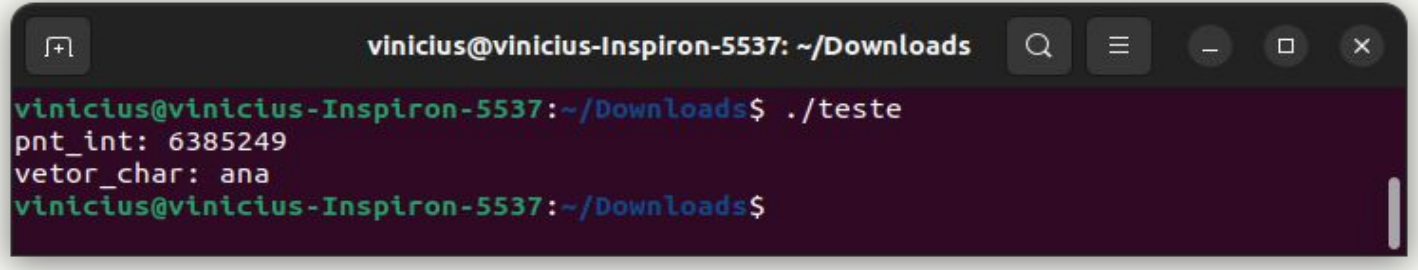
Memória e Tipos de Dados

```
1)   #include <stdlib.h>
2)   #include <stdio.h>
3)   int main (){
4)    unsigned char vetor_char[4] = {0,0,0,0};
5)    unsigned int *pnt_int;
6)    pnt_int = (unsigned int*) vetor_char;
7)    vetor_char[0] = 'a';
8)    vetor_char[1] = 'n';
9)    vetor_char[2] = 'a';
10)   vetor_char[3] = '\0';
11)   printf("\npnt_int: %d", *pnt_int);
12)   printf("\nvetor_char: %s", vetor_char);
13)   return 0;
14)  }
```

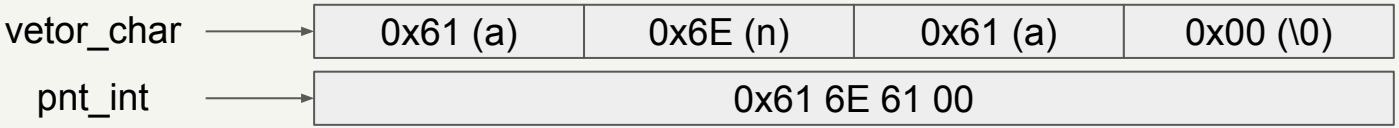
Observe o programa ao lado e responda às seguintes perguntas:

- O que exibe **pnt_int*?
- O que exibe *vetor_char*?

Memória e Tipos de Dados



Cada caractere é representado por um número de oito (8) bits na tabela ASCII. Colocando tais números em sequência e fazendo o tratamento *little endian*, obtemos o valor inteiro "correspondente" à *string* "ana".



Alocação Dinâmica de Memória

Essa ideia de trabalharmos apenas com *bytes* se estende para as alocações dinâmicas de memória!

Vamos analisar o protótipo da função *malloc*:

```
void *malloc(size_t size);
```

O tipo *size_t* é um **número inteiro sem sinal com a quantidade de bytes definida pela arquitetura adjacente**.

VOCÊ JÁ SE PERGUNTOU O QUE O PARÂMETRO *size* REPRESENTA?

Alocação Dinâmica de Memória

```
void *malloc(size_t size);
```

O parâmetro *size* indica a **quantidade de bytes**, em um espaço contínuo, que você deseja alocar! Ou seja:

malloc(10*sizeof(int)) ≡ malloc(20*sizeof(short)) ≡ malloc(40*sizeof(char))

≡

malloc(40)

Alocação Dinâmica de Memória

```
void *malloc(size_t size);
```

O que indica ao programa como interpretar esse espaço contínuo de bytes é, na verdade, o casting realizado para converter o ponteiro *void** para um ponteiro tipado!

```
int *ptr = (int*) malloc(40);
short *ptr = (short*) malloc(40);
char *ptr = (char*) malloc(40);
```

As *structs* em C

As *structs*, em C, são um **tipo de dados que permite criar unidades lógicas que agrupam múltiplas variáveis**, sejam elas de tipo homogêneo ou heterogêneo.

Em baixo nível, uma estrutura é um espaço de memória contínuo de tamanho igual ao somatório dos tamanhos, em bytes, das variáveis que a compõe.

Mais uma vez, estamos falando de agrupamentos de bytes!

As *structs* em C

```
1)   #include <stdlib.h>
2)   #include <stdio.h>
3)   typedef struct {
4)      char char_qualquer;
5)      int int_qualquer;
6)      double double_qualquer;
7)   } meus_dados;
```

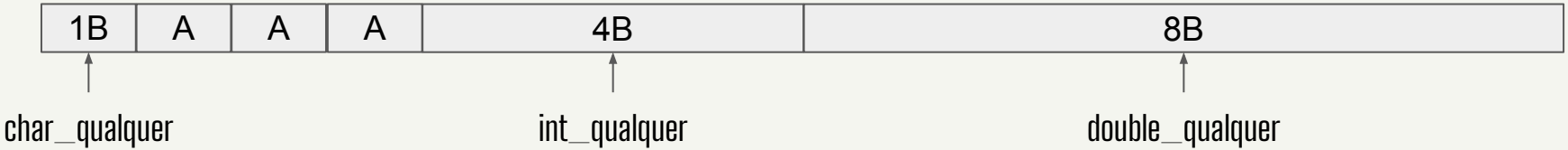
Observe a *struct* ao lado e responda às seguintes perguntas:

- Qual é o nome do tipo definido?
- Quais são as variáveis que compõem o tipo?
- Quantos bytes são necessários para instanciar a *struct*?

As *structs* em C

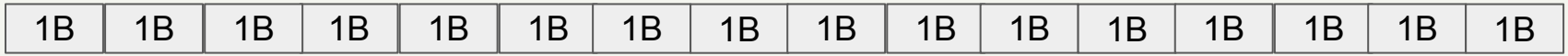
Legal! Então, se instanciamos uma variável local do tipo *meus_dados* teremos um **espaço de memória para essa variável organizado** como segue:

```
meus_dados var_meus_dados;
```



Um vetor de *char* de tamanho dezesseis (16) também tem dezesseis (16) bytes!

```
char vetor_char[16]
```



As *structs* em C

Mas, pera aí, como funciona esse tal de alinhamento?

Alinhar os dados é alocar os mesmos sempre em um endereço múltiplo de seu tamanho!

- Acesso eficiente de memória (otimizações de processador)
- Garantia de operações atômicas para acesso à memória
- ...

As *structs* em C

```
1)  int main (){
2)     unsigned char vetor_char[16];
3)     meus_dados *pnt_struct;
4)     pnt_struct = (meus_dados*) vetor_char;
5)     pnt_struct->char_qualquer = 'a';
6)     pnt_struct->int_qualquer = 1024;
7)     pnt_struct->double_qualquer = 255.255;
8)     printf("char_qualquer: %c \n",
   pnt_struct->char_qualquer);
9)     printf("int_qualquer: %d",
   pnt_struct->int_qualquer);
10)    printf("double_qualquer:
   %lf",pnt_struct->double_qualquer);
11)    return 0;
12)  }
```

Observe o programa ao lado e responda às seguintes perguntas:

- Como eu posso ter certeza do tamanho da minha *struct*?
- O que aconteceria se o vetor tivesse tamanho 13?

Moral da História

Se acostume a pensar os dados e a memória como agrupados de bytes!

Diminuir a abstração sobre os tipos de dados nos permite uma compreensão mais ampla de como os processos ocorrem em baixo nível.

É isso que diferencia um programador de um cientista da computação!

Estruturas de Dados

Até este momento, vocês já estudaram diversas estruturas de dados.

Algumas dessas estruturas serão amplamente utilizadas no decorrer desta disciplina.

Vamos revisar, rapidamente, as principais!

Estruturas de Dados

VETOR

Espaço contínuo de memória de um determinado tamanho (a depender do tipo de dados utilizado e da quantidade de elementos necessários).

Vetor global Vs. Vetor local Vs. Vetor alocado dinamicamente

Estruturas de Dados

LISTA

Estrutura de dados que organiza elementos individuais de forma linear via ponteiros. Qualquer elemento compatível pode ser adicionado, removido ou movido na lista.

Lista simplesmente encadeada Vs. Lista duplamente encadeada
Lista sem sentinela Vs. Lista com sentinela
Lista linear Vs. Lista circular

Estruturas de Dados

FILA

Estrutura de lista com regras específicas de inserção e remoção!

Em uma fila, sempre que um elemento é **inserido**, este é alocado como no **final** da estrutura.

Por outro lado, a operação de **remoção** em uma fila resulta na retirada do **primeiro (início)** elemento da mesma.

Estruturas de Dados

PILHA

Estrutura de lista com regras específicas de inserção e remoção!

Em uma fila, sempre que um elemento é **inserido**, este é alocado como no **início** da estrutura.

Também, a operação de **remoção** em uma fila resulta na retirada do **primeiro (início)** elemento da mesma.

Exercício #2

Você já ouviu falar na estrutura de deque? Deque é uma estrutura de dados que permite inserções e remoções em ambos extremos (é como se fosse uma pilha com uma fila).

Considere essa estrutura para desenvolver um programa que verifica se um conjunto de caracteres providos em um vetor formam um **palíndromo**.

Obrigado!

Vinícius Fülber Garcia
inf.ufpr.br/vinicius
vinicius@inf.ufpr.br