

AULA 08

{introcomp}

TIPOS ABSTRATOS DE DADOS

Estruturas

O que são?

São tipos de dados utilizados para manusear uma quantidade maior de informações de forma mais simples, eficiente e de tipos diferentes.

Struct em C, estrutura, é um bloco que armazenam diversas informações.

Essas estruturas podem ter quantos elementos você quiser e dos tipos que você quiser.



Estruturas

Qual o objetivo de fazer uma struct?

- Agrupamento de dados e informações de forma mais simplificada e eficiente.
- Facilidade na manipulação destes dados agrupados.
- Modularização do código, organização.



Criando uma Struct

Criação:

```
struct <nomestruct> {  
    <atributos>;  
};
```

Exemplo:

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



Criando uma Struct

Utilização:

```
#include <stdio.h>

struct {
    int x;
    int y;
} ponto;

int main(){
    struct ponto a,b;
    a.x=0;
    a.y=0;
    a.x=1;
    a.y=1;
    printf("Ponto A:\nx:%d,y:%d\n",a.x,a.y);
    printf("Ponto B:\nx:%d,y:%d\n",b.x,b.y);
}
```



Comando typedef

A palavra reservada `typedef` nada mais é do que um atalho em C para que possamos nos referir a um determinado tipo existente com nomes sinônimos.

Por exemplo, com o `typedef`, em vez de termos que nos referir como '`struct Aluno`', poderíamos usar somente '`Aluno`' para criar structs daquele tipo..



Criando uma struct

Criação:

```
typedef struct {  
    <atributos>;  
    <nometipo>;  
}<nomestruct>;
```

Criação:

```
typedef struct {  
    int x;  
    int y;  
} ponto;
```

{ }

Criando uma struct

Utilização:

```
#include <stdio.h>
|
typedef struct {
    int x;
    int y;
}ponto;

int main(){
    ponto a,b;
    a.x=0;
    a.y=0;
    a.x=1;
    a.y=1;
    printf("Ponto A:\nx:%d,y:%d\n",a.x,a.y);
    printf("Ponto B:\nx:%d,y:%d\n",b.x,b.y);
}
```

{ }

Acessando dados

```
int main (){
    aluno aluno1;
    scanf ("%s %d %d",aluno1.nome, &aluno1.idade, &aluno1.matricula);

    printf ("Nome do aluno %s \n",aluno.nome);
    printf ("\tIdade: %d\n",aluno.idade);
    printf ("\tMatricula: %d\n",aluno.matricula);

    aluno.idade++; //Aumenta o valor da variavel idade referente ao aluno1
    return 0;
}
```



Passando uma struct como parâmetro para uma função

```
int AumentaIdade (int idade) {  
    idade++;  
    return idade;  
}  
  
int main () {  
  
    aluno alunoi;  
    alunoi.idade = 10;  
    int x;  
    x = AumentaIdade (aluno.idade);  
    printf ("%d",x);  
  
    return 0;  
}
```

```
aluno AumentaIdade (aluno alunoi){  
    alunoi.idade++  
    return alunoi;  
}  
  
int main () {  
  
    aluno alunoi;  
    alunoi.idade = 10;  
    alunoi = AumentaIdade (alunoi);  
    printf ("%d",alunoi.idade);  
  
    return 0;  
}
```



Exemplos

Crie um struct data.



Exemplos

```
typedef struct data{  
    int dia;  
    int mes;  
    int ano;  
}Data;
```



Exemplos

Faça uma struct que tenha um vetor de coordenadas x,y.



Exemplos

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



Exemplos

Faça um programa que leia N alunos (nome, idade e nota) e diga qual é o mais velho, e qual tem a maior nota.



Estruturas e Estruturas

Levando em conta que estruturas podem armazenar qualquer tipo de variável, isso possibilita a inclusão de outros tipos de estruturas genéricas a outras estruturas.

```
typedef struct{
    int x,y;
}Ponto;
typedef struct{
    Ponto a,b,c,d;
}Retangulo;
```



Estruturas e Vetores

Sabendo da capacidade que os structs possuem, isso nos possibilita a utilização de vetores internos a eles, ou de vetores de estrutura.

```
typedef struct <nomestruct>{  
    <tipovariavel> vetor[<tamanhovetor>];  
};
```



Estruturas e Vetores

As principais características de um vetor em uma estrutura são:

- Podemos copiar dois vetores diretamente
- Vetor a, b; b = a;
- Ao ser passado como parâmetro em uma função, caso se altere valores do vetor dentro da função, ele não terá seu valor alterado na main;
- Podemos retornar vetores em uma função;
- Não podemos definir o tamanho máximo do vetor em tempo de execução.



Estruturas e Vetores - Exemplos

```
typedef struct {    int main (){  
    int vetor[MAX];        Vetor vet[10];  
    int tam;                return 0;  
}Vetor;
```



TAD's

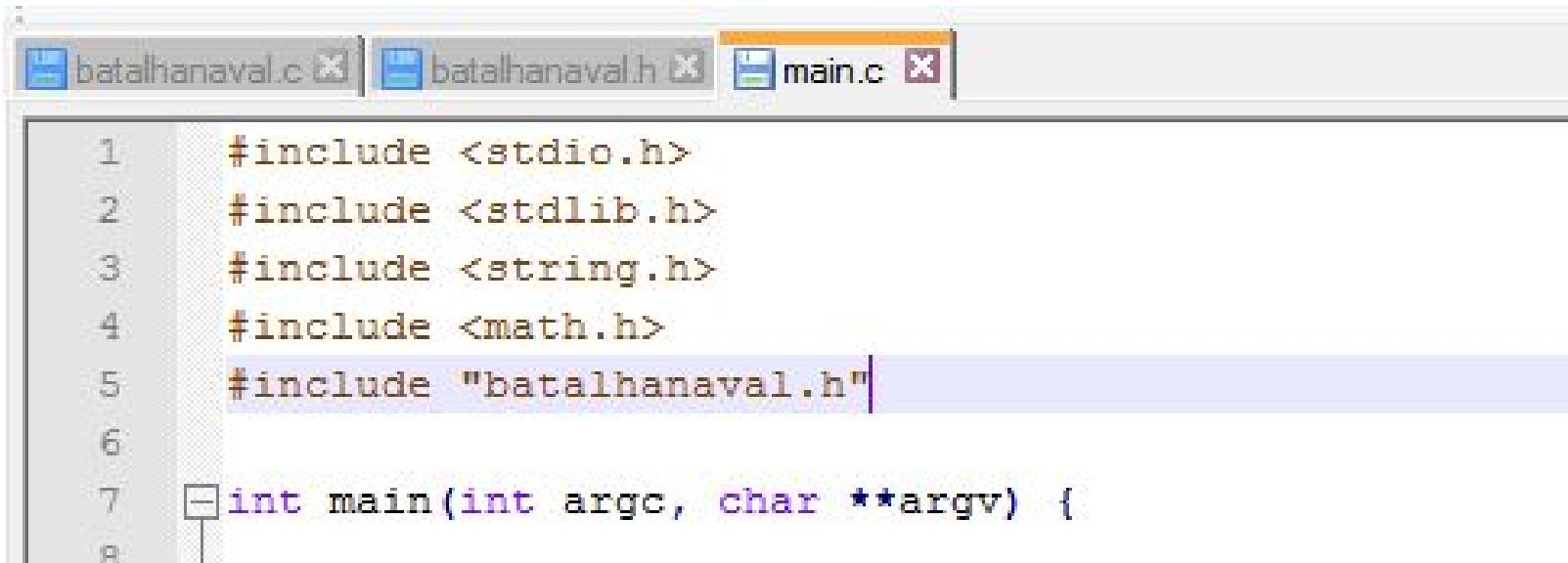
Os TAD's podem ser criados separadamente e incluídos como uma biblioteca.

No .c são implementadas todas as funções com os TAD's.

No .h estão apenas as assinaturas das funções e a estrutura.



TAD's



A screenshot of a code editor showing a C program. The tabs at the top are labeled "batalhanaval.c", "batalhanaval.h", and "main.c". The "main.c" tab is active. The code in the editor is:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <math.h>
5 #include "batalhanaval.h"
6
7 int main(int argc, char **argv) {
```

{ }

TAD's

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <math.h>
5 #include "batalhanaval.h"
6
7 tTabuleiro CriaTabuleiro(int x) {
8
9     tTabuleiro m;
10    int i, j;
11    m.linha = x;
12    m.coluna = x;
13    m.matriz = malloc(x * sizeof (int*));
14    for (i = 0; i < m.linha; i++) {
15        m.matriz[i] = malloc(x * sizeof (int));
16    }
17    return m;
18 } //cria um tabuleiro quadrado.
19
20 tNavio LeNavio(FILE *pCONFIG) {
57
58 void DestroiNavios(int qtdTipos, tNavio *navios) {
72
73 void PreencheTabuleiro(tTabuleiro tabuleiro) {
82
83 void ImprimeTabuleiro(tTabuleiro tabuleiro) {
93
94 void DestroiTabuleiro(tTabuleiro tabuleiro) {
102
103 void PoeNavioNoTabuleiro(FILE *pTABU, tTabuleiro tabu, tNavio *navios, int qtdTipos, int qtdTotal) {
141
142 void DaTiro(tTabuleiro tabu, int x, int y, tNavio *navios, int qtdTipos) {
```

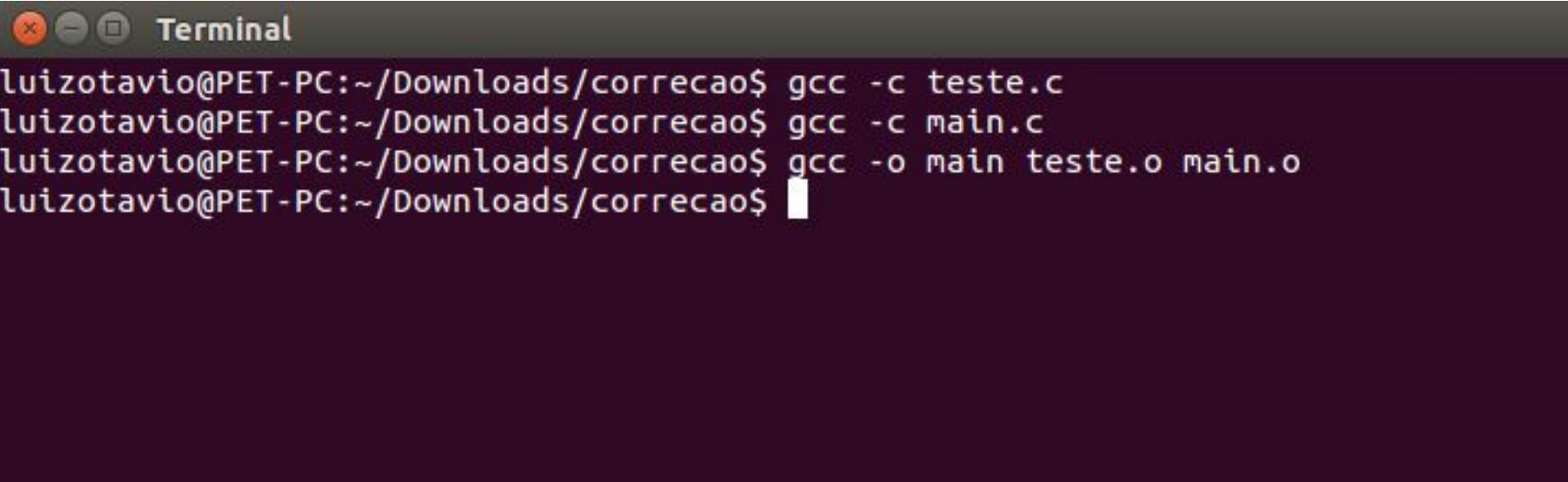
{ }

TAD's

```
1 #ifndef BATALHANAVAL_H
2 #define BATALHANAVAL_H
3
4 typedef struct {
5     int linha;
6     int coluna;
7     int** matriz;
8 } tTabuleiro;
9
10 typedef struct {
11
12     typedef struct {
13
14         tTabuleiro CriaTabuleiro(int x);
15
16         tNavio LeNavio(FILE *pCONFIG);
17
18         void DestroiNavios(int qtdTipos, tNavio *navios);
19
20         void PreencheTabuleiro(tTabuleiro tabuleiro);
21
22         void ImprimeTabuleiro(tTabuleiro tabuleiro);
23
24         void DestroiTabuleiro(tTabuleiro tabuleiro);
25
26         void PoeNavioNoTabuleiro(FILE *pTABU, tTabuleiro tabu, tNavio *navios, int qtdTipos, int qtdTotal);
27
28         int DaTiro(tTabuleiro tabu, int x, int y, tNavio *navios, int qtdTipos);
29
30     #endif
```

{ }

Compilação



A screenshot of a terminal window titled "Terminal". The window has a dark background and light-colored text. It shows the following command-line session:

```
luizotavio@PET-PC:~/Downloads/correcao$ gcc -c teste.c
luizotavio@PET-PC:~/Downloads/correcao$ gcc -c main.c
luizotavio@PET-PC:~/Downloads/correcao$ gcc -o main teste.o main.o
luizotavio@PET-PC:~/Downloads/correcao$
```



Makefile

Quando trabalhamos com muitos arquivos, começa a ficar trabalhoso sempre que fizermos uma pequena modificação compilarmos tudo de novo.

O makefile é um arquivo com regras para compilação, o que facilita o uso de vários arquivos.



Makefile

The screenshot shows a Linux desktop environment. In the top panel, there is a window titled "makefile (~/Downloads/correcao) - gedit". The gedit interface has tabs for "teste.c", "main.c", and "makefile", with "makefile" currently selected. The code in the makefile is:

```
1 all:compila executa
2 compila:
3     gcc -c teste.c
4     gcc -c main.c
5     gcc -o main teste.o main.o
6 executa:
7     ./main
8 limpa:
9     rm *.o
```

Below the gedit window is a terminal window titled "Terminal". The terminal output is:

```
luizotavio@PET-PC:~/Downloads/correcao$ make compila
gcc -c teste.c
gcc -c main.c
gcc -o main teste.o main.o
luizotavio@PET-PC:~/Downloads/correcao$
```

{ }

AULA 08

{introcomp}

TIPOS ABSTRATOS DE DADOS