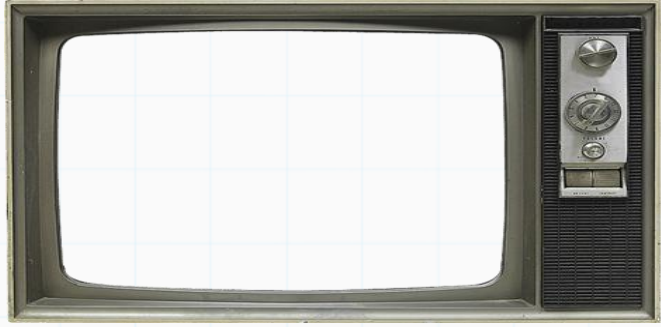
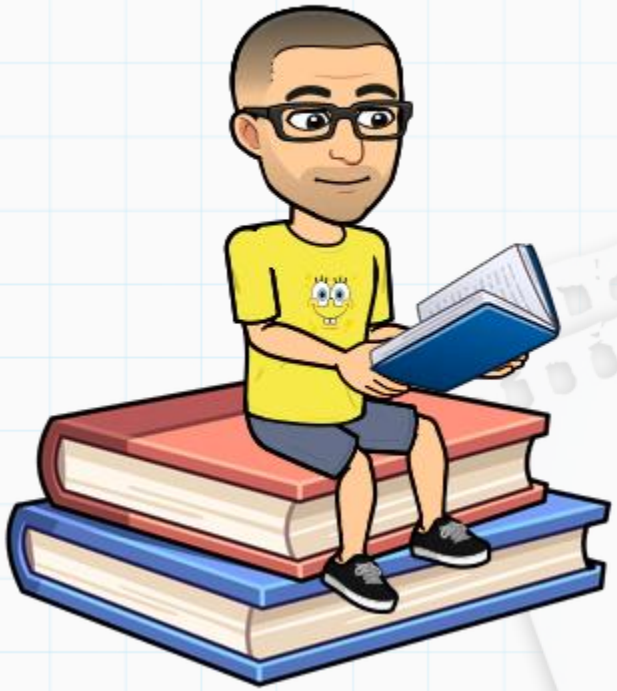
Programação De Computadores

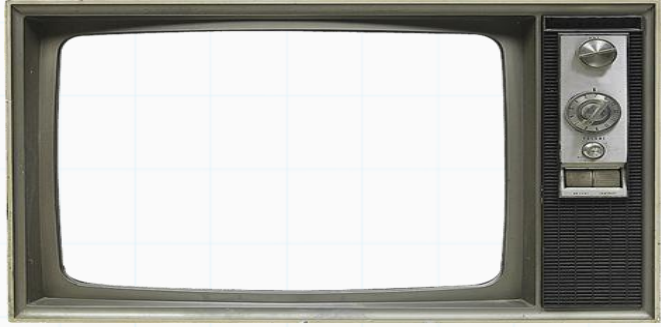


Professor : Yuri Frota

yuri@ic.uff.br



Estruturas



- Uma estrutura é uma coleção de uma ou mais váriáveis colocadas juntas em um único nome

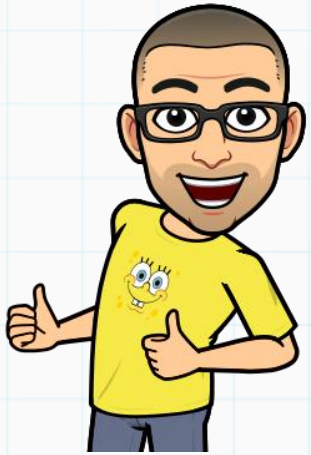
Exemplos:

Aluno

- Número de matrícula
- CR
- Nome

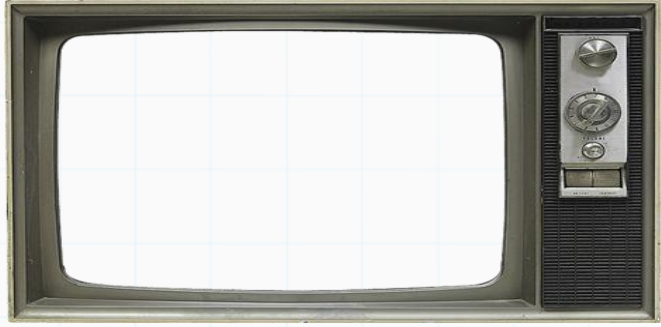
Ponto

- x
- y



- Permitem agrupar dados para uma manipulação mais organizada e conveniente

Estruturas

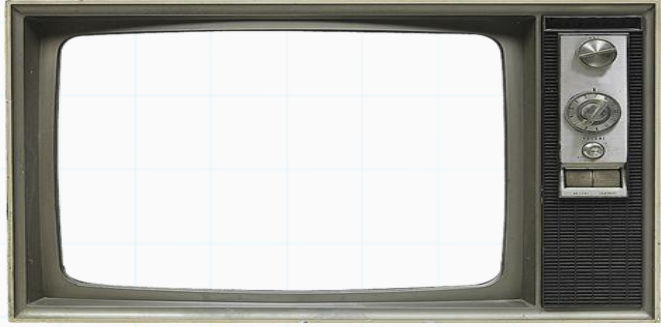


- Declaração:

```
struct nome_da_estrutura {
                    tipo comp1;
                    tipo comp2;
                    ...
                };
```

geralmente declaradas no começo do programa, antes da main(), mas poderia ser qualquer lugar

Estruturas



- Declaração:

```
struct nome_da_estrutura {
                tipo comp1;
                tipo comp2;
                ...
            };
```

geralmente declaradas no começo do programa, antes da main(), mas poderia ser qualquer lugar

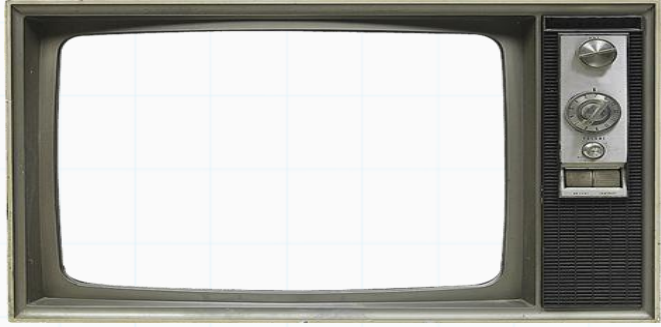
- Exemplo:

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

```
struct tipoAluno {
    int numMat;
    float CR;
    char nome[40];
};
```

esta declaração não aloca memória, apenas define um tipo de estrutura

Estruturas



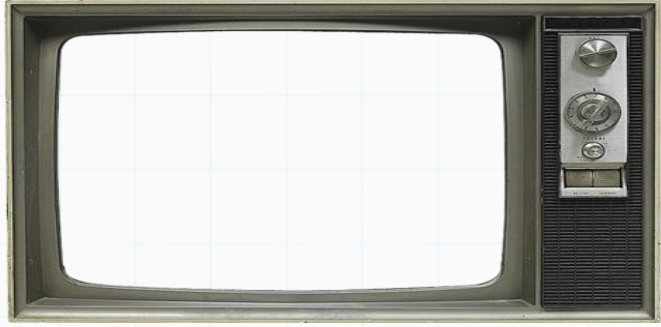
- Uma declaração `struct` pode ser seguida por uma lista de variáveis

```
struct tipoPonto {
        int x;
        int y;
} p1, p2, p3;
```

p1,p2 e p3 são variáveis do tipo dessa estrutura

Se a declaração não é seguida por uma lista de variáveis, não é reservado espaço de memória. Apenas é descrito o formato da estrutura.

Estruturas



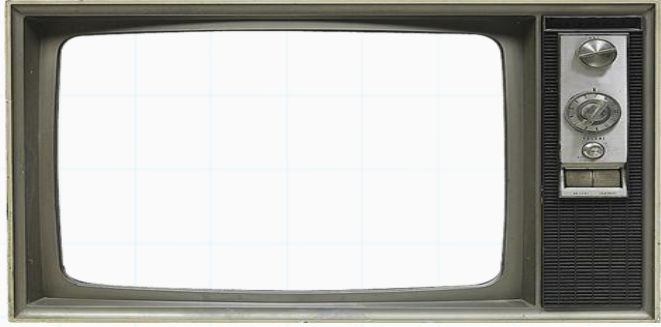
O programador também pode criar um tipo da sua definição de estrutura.

Exemplo:

```
struct tipoAluno {
        int numMat;
        float CR;
        char nome[40];
};
typedef struct tipoAluno tAluno;
tAluno aluno1, aluno2;
```

aluno1 e aluno2 são do tipo tAluno que foi definido como uma estrutura tipoAluno

Estruturas



O programador também pode criar um tipo da sua definição de estrutura.

Exemplo:

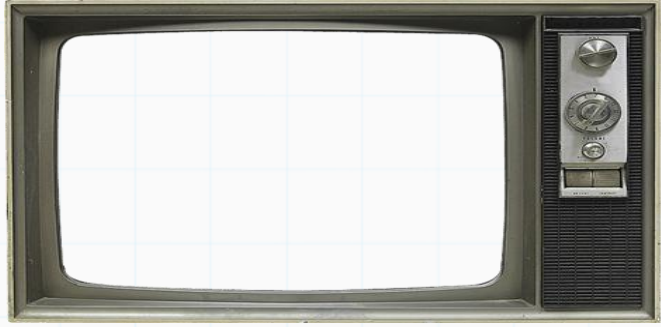
```
struct tipoAluno {
        int numMat;
        float CR;
        char nome[40];
};
typedef struct tipoAluno tAluno;
tAluno aluno1, aluno2;
```

aluno1 e aluno2 são do tipo tAluno que foi definido como uma estrutura tipoAluno

```
struct tipoAluno aluno1, aluno2;
```

mas você poderia ter definido as variáveis diretamente usando a estrutura

Estruturas



O programador também pode criar um tipo da sua definição de estrutura.

Exemplo:

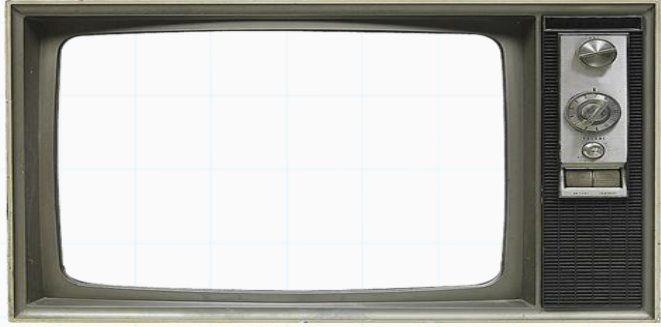
```
struct tipoAluno {
        int numMat;
        float CR;
        char nome[40];
};
typedef struct tipoAluno tAluno;
tAluno aluno1, aluno2;
```

aluno1 e aluno2 são do tipo tAluno que foi definido como uma estrutura tipoAluno

- Acessando as variáveis da estrutura, usamos o nome da variável estrutura e “.”

```
aluno1.numMat = 5;
aluno1.CR     = 7.6;
strcpy(aluno1.nome, "seya");
```

Estruturas



O programador também pode criar um tipo da sua definição de estrutura.

Exemplo:

```
struct tipoAluno {
        int numMat;
        float CR;
        char nome[40];
};
typedef struct tipoAluno tAluno;
tAluno aluno1, aluno2;
```

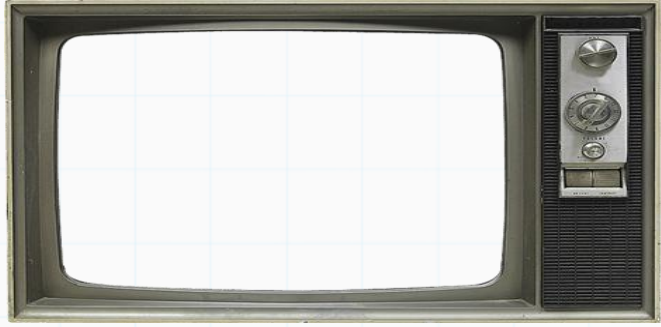
aluno1 e aluno2 são do tipo tAluno que foi definido como uma estrutura tipoAluno

- As únicas operações legais em um estrutura são copiá-las/atribuí-las como uma unidade

```
aluno1.numMat = 5;
aluno1.CR     = 7.6;
strcpy(aluno1.nome, "seya");
aluno2 = aluno1;
```

- Mas não podem ser comparadas como uma unidade (aluno1 == aluno2 ✕)

Estruturas



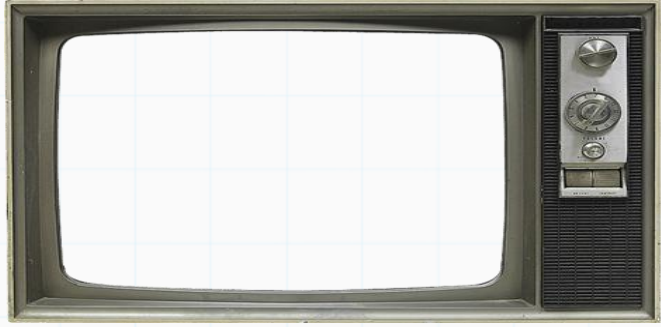
Estruturas podem ser passadas para funções

Exemplo:

```
struct tipoPonto {
        int x;
        int y;
   };
   typedef struct tipoPonto tPonto;

int main (void) {
        tPonto A,B,C;
        A.x = 1;
        A.y = 2;
        B.x = 3;
        B.y = 4;
        C = somaPonto(A,B);
        printf("%d %d", C.x, C.y);
        return 0;
   }
```

```
tPonto somaPonto(tPonto M, tPonto N) {
        tPonto K;
        K.x = M.x + N.x;
        K.y = M.y + N.y;
        return K;
}
```

Estruturas



Podemos criar estruturas de estruturas

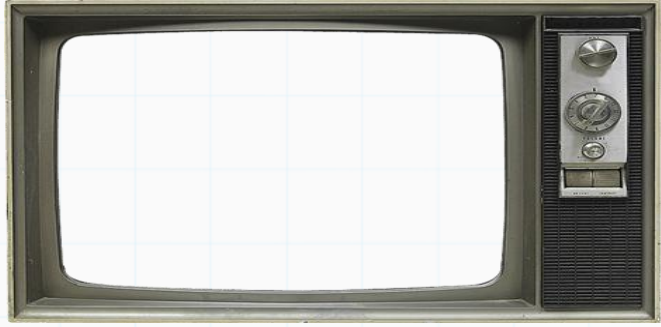
Exemplo:

```
struct tipo_hora {
      int hora;
      int minuto;
      int segundo;
};
typedef struct tipo_hora thora;
```

```
struct tipo_data {
      int dia;
      int mes;
      int ano;
};
typedef struct tipo_data tdata;
```

```
struct tipo_data_hora {
      thora h;
      tdata d;
};
typedef struct tipo_data_hora  tdata_hora;
```

Estruturas



Podemos criar estruturas de estruturas

Exemplo:

```
struct tipo_hora {
      int hora;
      int minuto;
      int segundo;
};
typedef struct tipo_hora thora;
```

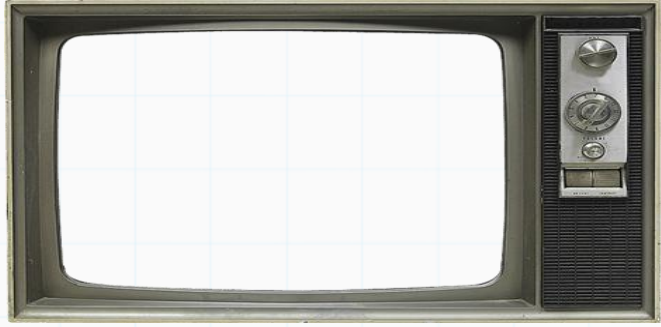
```
struct tipo_data {
      int dia;
      int mes;
      int ano;
};
typedef struct tipo_data tdata;
```

```
struct tipo_data_hora {
      thora h;
      tdata d;
};
typedef struct tipo_data_hora  tdata_hora;
```

Acesso:

```
tdata_hora evento;

evento.d.dia = 3;
evento.h.hora = 15;
```

Estruturas



Podemos criar vetor de estruturas

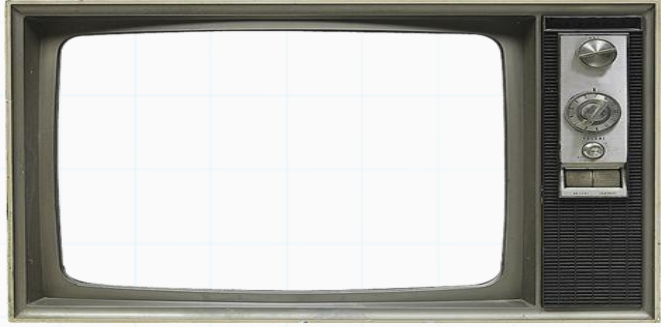
Exemplo:

```
struct tipoAluno {
      int numMat;
      float CR;
      char nome[40];
};
typedef struct tipoAluno tAluno;
```

```
 tAluno vetAluno[40];
```

```
vetAluno[2].numMat
vetAluno[2].CR
vetaluno[2].nome

vetAluno[5].numMat
vetAluno[5].CR
vetaluno[5].nome
```

Estruturas



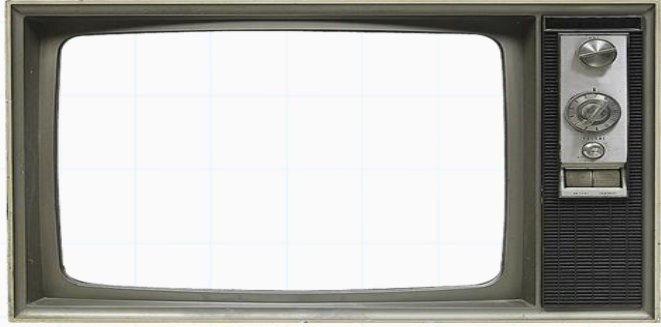
Podemos fazer ponteiros para estruturas

Exemplo:

```
struct tipoAluno {
      int numMat;
      float CR;
      char nome[40];
};
typedef struct tipoAluno tAluno;
tAluno aluno;

tAluno * pt_aluno;
pt_aluno = &aluno;

(*pt_aluno).CR = 7.6;
```

acesso a estrutura através do ponteiro

Estruturas



Podemos fazer ponteiros para estruturas

Exemplo:

```
struct tipoAluno {
      int numMat;
      float CR;
      char nome[40];
};
typedef struct tipoAluno tAluno;
tAluno aluno;

tAluno * pt_aluno;
pt_aluno = &aluno;

(*pt_aluno).CR = 7.6;
```

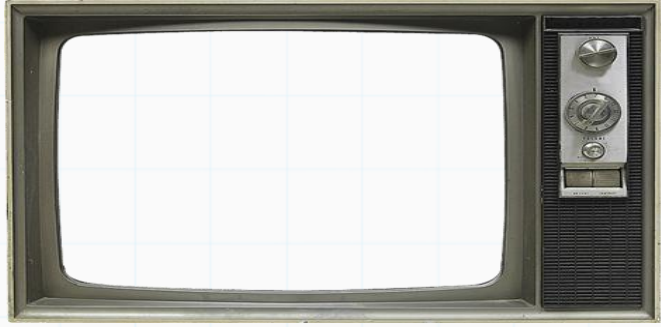
acesso a estrutura através do ponteiro

Apontadores para estruturas são tão usados que existe uma notação especial "->":

```
(*pt_aluno).CR OU pt_aluno->CR
```

MUITO USADO !!!

Estruturas



Ao passarmos estruturas para as funções, podemos passar tanto por valor (copia) quanto por referencia (ponteiro para a própria estrutura).

Exemplo:

```
struct tipoAluno {
      int numMat;
      float CR;
      char nome[40];
};
typedef struct tipoAluno tAluno;
tAluno aluno;
```

```
exemploFunc(aluno);   → por valor

exemploFunc(&aluno); → por referência
```

passando o endereço de memoria (ponteiro)

Estruturas

Ao passarmos estruturas para as funções, podemos passar tanto por valor (copia) quanto por referencia (ponteiro para a própria estrutura).

Exemplo:

```
struct tipoAluno {
      int numMat;
      float CR;
      char nome[40];
};
typedef struct tipoAluno tAluno;
tAluno aluno;
```

```
void exemploFunc(tAluno aluno){
   ...
   printf("Mat: &d", aluno.numMat);
   printf("CR: &f", aluno.CR);
```

```
void exemploFunc(tAluno *aluno){
   ...
   printf("Mat: &d", (*aluno).numMat);
   OU printf("Mat: &d", aluno->numMat);
   printf("CR: &f", (*aluno).CR);
   OU printf("CR: &f", aluno->CR);
```

```
exemploFunc(aluno);   → por valor
exemploFunc(&aluno); → por referência
```

passando o endereço de memoria (ponteiro)

Estruturas

Ao passarmos estruturas para as funções, podemos passar tanto por valor (copia) quanto por referencia (ponteiro para a própria estrutura).

- Passar estruturas por referencia é muito mais rápido do que por valor porque não existe a cópia da estrutura.

- A estrutura poderia ser gigante ☹

- Mas se passar por referencia, você irá alterar a estrutura original

```
void exemploFunc(tAluno aluno){
   ...
   printf("Mat: &d", aluno.numMat);
   printf("CR: &f", aluno.CR);
```

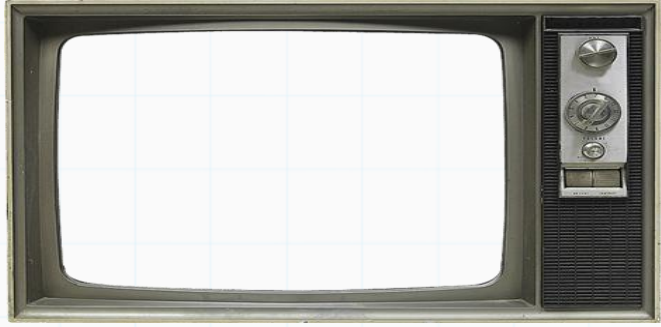
```
void exemploFunc(tAluno *aluno){
   ...
   printf("Mat: &d", (*aluno).numMat);
   OU printf("Mat: &d", aluno->numMat);
   printf("CR: &f", (*aluno).CR);
   OU printf("CR: &f", aluno->CR);
```

Exemplo

Considere a seguinte estrutura pontos:

```
struct tipoPonto {
        int x;
        int y;
    };
    typedef struct tipoPonto tPonto;
```

Vamos fazer na função main() a declaração de 2 variáveis do tipo ponto e ler 2 pontos dado pelo usuário

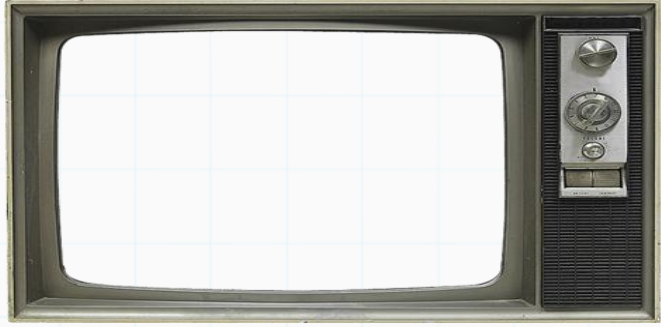


```
int main()
{
    tPonto p1,p2, ps;

    printf("p1.x:");
    scanf("%d",&p1.x);
    printf("p1.y:");
    scanf("%d",&p1.y);
    printf("p2.x:");
    scanf("%d",&p2.x);
    printf("p2.y:");
    scanf("%d",&p2.y);



    printf("x=%d y=%d\n",p1.x, p1.y);
    printf("x=%d y=%d\n",p2.x, p2.y);
    printf("x=%d y=%d\n",ps.x, ps.y);

    return 0;
}
```

Exemplo

Considere a seguinte estrutura pontos:

```
struct tipoPonto {
        int x;
        int y;
    };
    typedef struct tipoPonto tPonto;
```



Vamos fazer na função main() a declaração de 2 variáveis do tipo ponto e ler 2 pontos dado pelo usuário

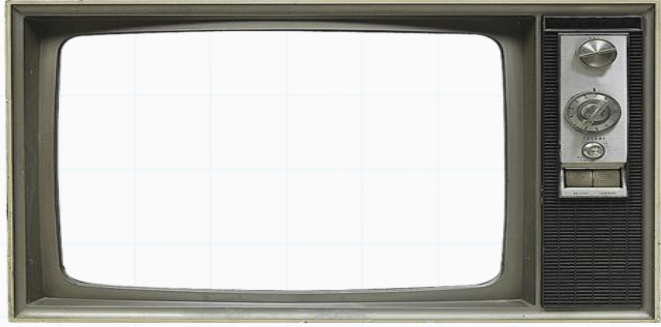
O programa deve ter uma função única soma_multi() que receba os dois pontos (passagem por valor da estrutura) e retorne um novo ponto gerados pela dos 2 pontos originais.

```
int main()
{
    tPonto p1,p2, ps;

    printf("p1.x:");
    scanf("%d",&p1.x);
    printf("p1.y:");
    scanf("%d",&p1.y);
    printf("p2.x:");
    scanf("%d",&p2.x);
    printf("p2.y:");
    scanf("%d",&p2.y);

    ps = soma_multi(p1, p2);

    printf("x=%d y=%d\n",p1.x, p1.y);
    printf("x=%d y=%d\n",p2.x, p2.y);
    printf("x=%d y=%d\n",ps.x, ps.y);

    return 0;
}
```

Exemplo

Considere a seguinte estrutura pontos:

```
struct tipoPonto {
        int x;
        int y;
    };
    typedef struct tipoPonto tPonto;
```



Vamos fazer na função main() a declaração de 2 variáveis do tipo ponto e ler 2 pontos dado pelo usuário

O programa deve ter uma função única soma_multi() que receba os dois pontos (passagem por valor da estrutura) e retorne um novo ponto gerados pela dos 2 pontos originais.

```
int main()
{
    tPonto p1,p2, ps;

    printf("p1.x:");
    scanf("%d",&p1.x);
    printf("p1.y:");
    scanf("%d",&p1.y);
    printf("p2.x:");
    scanf("%d",&p2.x);
    printf("p2.y:");
    scanf("%d",&p2.y);

    ps = soma_multi(p1, p2);

    printf("x=%d y=%d\n",p1.x, p1.y);
    printf("x=%d y=%d\n",p2.x, p2.y);
    printf("x=%d y=%d\n",ps.x, ps.y);

    return 0;
}
```

```
tPonto soma_multi(tPonto p1, tPonto p2)
{
    tPonto soma;

    soma.x = p1.x + p2.x;
    soma.y = p1.y + p2.y;

    return soma;
}
```

Exercicio

1) Itens: Faça um programa que crie uma estrutura item com os seguintes campos:

nome (ponteiro para caractere)
qtd (inteiro)
preço (float)
total (float) /* *irá armazenar qtd*preço* */

Use só o que aprendemos até hoje

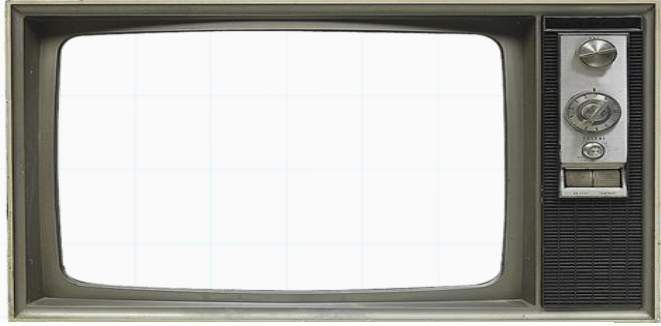
- O programa deve ter uma função ler_item que deve receber um ponteiro para a estrutura item, receber do usuário (**scanf**) e armazenar na estrutura nome, quantidade, preço e total de um produto.

- O programa deve ter uma função imprime_item que deve receber um ponteiro para a estrutura item e imprimir as informações do item.

- na função main() deve-se declarar uma variável do tipo item e uma outra do tipo ponteiro para este item.
  - Você deve alocar memória (malloc) para o nome (tam=50)
  - o ponteiro do item deve ser passado para as duas funções
  - Não esquecer de liberar memoria

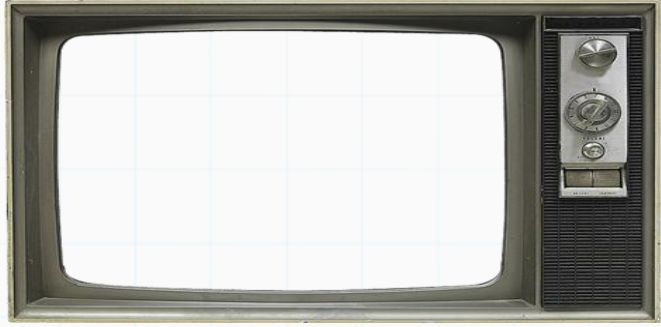
```
struct tipoPonto {
        int x;
        int y;
};
typedef struct tipoPonto tPonto;

tPonto p1,p2;
tPonto *p3;

p1.x = 12;
p1.y = 27;
p3 = &p1;
```

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

struct item
{
    char *nome;
    int qtd;
    float preco;
    float total;
};

void ler_item(struct item *i)
{
    printf("nome do produto: ");
    scanf("%s", i->nome);
    printf("\npreco: ");
    scanf("%f", &i->preco);
    printf("\nquantidade: ");
    scanf("%d", &i->qtd);
    i->total = (float)i->qtd * i->preco;
}

void imprime_item(struct item *i)
{
    /*print item details*/
    printf("\nName: %s",i->nome);
    printf("\npreco: %.2f",i->preco);
    printf("\nQuantity: %d",i->qtd);
    printf("\nTotal total: %.2f",i->total);
}
```



```c
int main()
{
    struct item itm;
    struct item *pItem;
    pItem = &itm;
    pItem->nome = (char *) malloc(50 * sizeof(char));

    ler_item(pItem);
    imprime_item(pItem);

    free(pItem->nome);
    return 0;
}
```

TAD



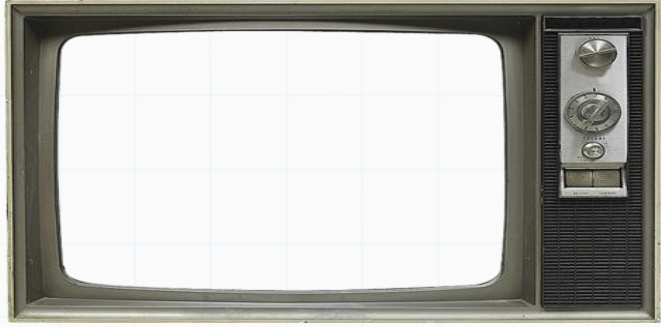
Tipo Abstrato de Dados: É um conceito em que definimos em um programa um tipo de dado específico e um conjunto de operações/funções que podem ser realizadas nesse dado:

Ex: Dado: inteiro x

Operações: + , - , * , / , ++ , ...

O usuário não precisa saber como as operações serão implementadas, ele quer apenas usa-las.

TAD



Tipo Abstrato de Dados: É um conceito em que definimos em um programa um tipo de dado específico e um conjunto de operações/funções que podem ser realizadas nesse dado:

Ex: Dado: inteiro x

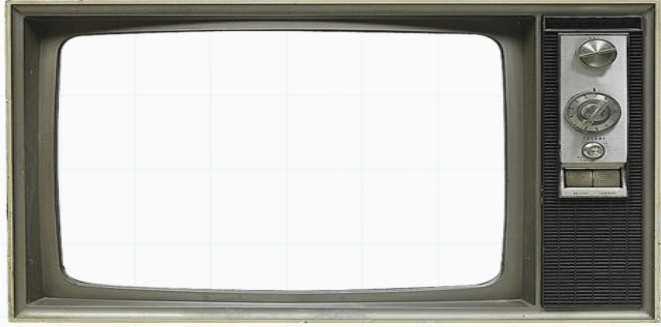
Operações: + , - , * , / , ++ , ...

O usuário não precisa saber como as operações serão implementadas, ele quer apenas usa-las.

Este dado pode ser definido como :

- por um tipo simples existente na linguagem (ex: inteiros, vetores, etc)
- por um tipo composto, estruturado (ex: estruturas, etc)

TAD



Tipo Abstrato de Dados: É um conceito em que definimos em um programa um tipo de dado específico e um conjunto de operações/funções que podem ser realizadas nesse dado:

Ex: Dado: inteiro x

Operações: + , - , * , / , ++ , ...

O usuário não precisa saber como as operações serão implementadas, ele quer apenas usa-las.

Este dado pode ser definido como :

- por um tipo simples existente na linguagem (ex: inteiros, vetores, etc)
- por um tipo composto, estruturado (ex: estruturas, etc)

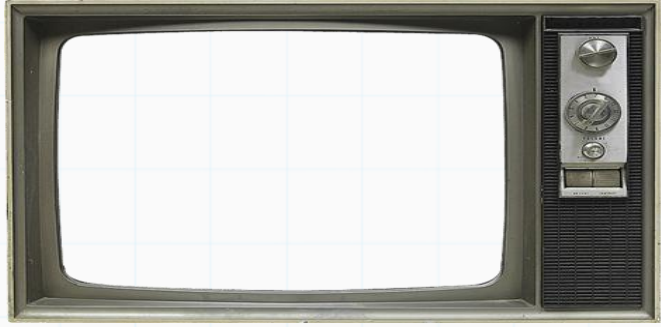
Geralmente é implementado em um arquivo separado para facilitar o reuso

- podemos implementar ele em um arquivo tad.c e incluir como uma biblioteca

```
#include <stdio.h>

#include <stdlib.h>

#include "tad.c"
```

Bibliotecas indicadas com “” são definidas pelo usuário, enquanto as indicadas por <> são definidas pelo sistema

TAD



Ex: Como ficaria o TAD dos pontos para o problema 1)

main.c

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

#include "tad.c"

int main()
{
    tPonto p1,p2, ps;

    printf("p1.x:");
    scanf("%d",&p1.x);
    printf("p1.y:");
    scanf("%d",&p1.y);
    printf("p2.x:");
    scanf("%d",&p2.x);
    printf("p2.y:");
    scanf("%d",&p2.y);

    ps = soma_multi(p1, p2);

    printf("x=%d y=%d\n",p1.x, p1.y);
    printf("x=%d y=%d\n",p2.x, p2.y);
    printf("x=%d y=%d\n",ps.x, ps.y);

    return 0;
}
```

o usuário não está interessado em saber como soma_multi() foi implementada

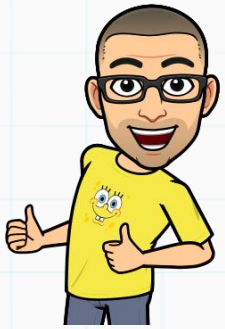
tad.c

```c
// Definição do TAD para pontos

struct tipoPonto {
    int x;
    int y;
};
typedef struct tipoPonto tPonto;

tPonto soma_multi(tPonto p1, tPonto p2)
{
    tPonto soma;

    soma.x = p1.x + p2.x;
    soma.y = p1.y + p2.y;

    return soma;
}
```

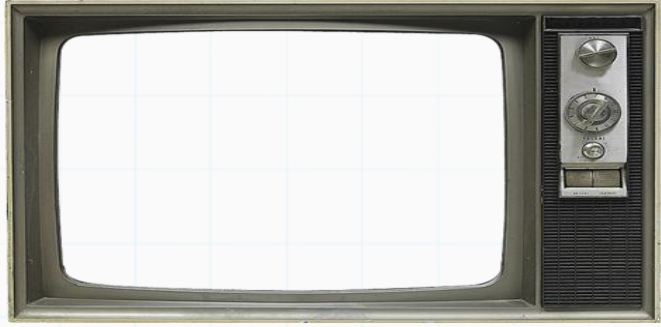
TAD

Podemos expandir esse tad para conter diversas outras funções:

- tPonto * cria_ponto(int x, int y)
- void deleta_ponto(tPonto * p1)
- float distancia(tPonto * p1, tPonto * p2)
- void imprime(tPonto * p1)
- void altera(tPonto * p1, int x, int y)
- ....



Sem afetar o arquivo main.c, podendo ser utilizado em outros projetos também.



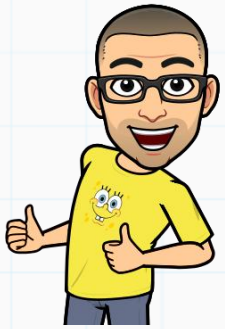
tad.c

```c
// Definição do TAD para pontos

struct tipoPonto {
    int x;
    int y;
};
typedef struct tipoPonto tPonto;

tPonto soma_multi(tPonto p1, tPonto p2)
{
    tPonto soma;

    soma.x = p1.x + p2.x;
    soma.y = p1.y + p2.y;

    return soma;
}
```

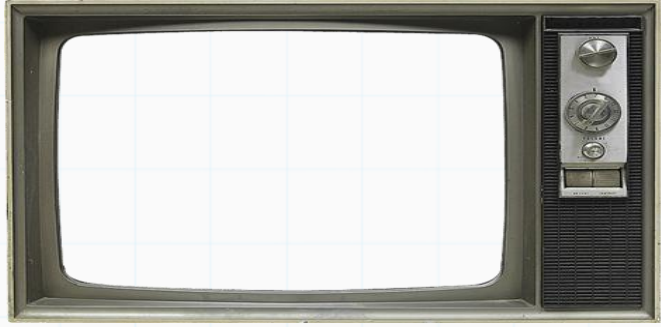
TAD

Podemos expandir esse tad para conter diversas outras funções:

- tPonto * cria_ponto(int x, int y)
- void deleta_ponto(tPonto * p1)
- float distancia(tPonto * p1, tPonto * p2)
- void imprime(tPonto * p1)
- void altera(tPonto * p1, int x, int y)
- ....



Sem afetar o arquivo main.c, podendo ser utilizado em outros projetos também.



tad.c

```c
// Definição do TAD para pontos

struct tipoPonto {
    int x;
    int y;
};
typedef struct tipoPonto tPonto;

tPonto soma_multi(tPonto p1, tPonto p2)
{
    tPonto soma;

    soma.x = p1.x + p2.x;
    soma.y = p1.y + p2.y;

    return soma;
}
```

Exemplo:

```c
#include <stdio.h>
#include <stdlib.h>
#include "tad.c"

int main (){
    float x, y;
    tPonto* p = cria_ponto(2.0,1.0);
    tPonto* q = cria_ponto(3.4,2.1);

    float d = distancia(p,q);
    printf("Distancia entre pontos: %f\n",d);

    deleta_ponto(q); deleta_ponto(p);
    return 0;
}
```

Exercicio

2) Pontos TAD: Dados os programas abaixo, escreva as funções em tad.c que precisam para usarmos na main.c

main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "tad.c"

int main (){
    float x, y;
    tPonto* p = cria_ponto(2.0,1.0);
    tPonto* q = cria_ponto(3.4,2.1);
    imprime_ponto(p1);
    imprime_ponto(p2);

    float d = distancia(p,q);
    printf("Distancia entre pontos: %f\n",d);

    deleta_ponto(q); deleta_ponto(p);
    return 0;
}
```

tad.c

```c
// Definição do TAD para pontos

struct tipoPonto {
    float x;
    float y;
};
typedef struct tipoPonto tPonto;
```

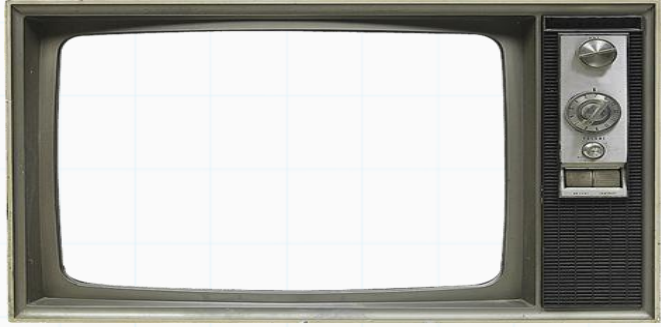
$$(X_1, Y_1); (X_2, Y_2)$$

$$d = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}$$

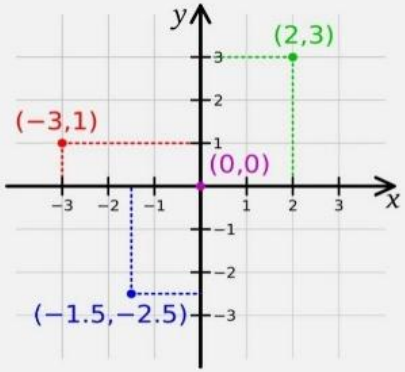
OBS Em cria_ponto, alocar com malloc a estrutura e retornar o ponteiro para estrutura criada

OBS2 Em distancia, usar a função matemática de raiz quadrada em C **sqrt**

```c
// Definição do TAD para pontos

struct tipoPonto {
    float x;
    float y;
};
typedef struct tipoPonto tPonto;

tPonto* cria_ponto(float x, float y)
{
    tPonto* p = (tPonto*) malloc(sizeof(tPonto));
    if (p == NULL) {
        printf("Memória insuficiente!\n");
        exit(1);
    }
    p->x = x;
    p->y = y;
    return p;
}

void imprime_ponto(tPonto* p1)
{
    printf("x:%f\n",p1->x);
    printf("y:%f\n",p1->y);
}
```



```c
float distancia(tPonto* p1, tPonto* p2)
{
    float fx = (p1->x - p2->x)*(p1->x - p2->x);
    float fy = (p1->y - p2->y)*(p1->y - p2->y);
    return sqrt(fx + fy);
}

void deleta_ponto(tPonto* p1)
{
    free(p1);
}
```



Exercicio

3) Vetor TAD: Dados os programas abaixo, escreva as funções em tad.c que precisam para usarmos na main.c

main.c

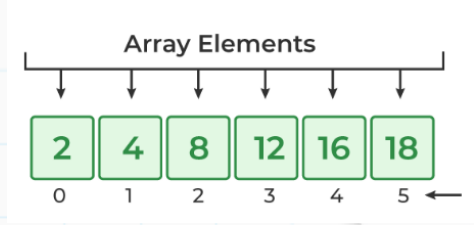
```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "tad.c"

int main (){
    meuVetor* vetor;

    vetor = aloca_vetor(10);      // max=10
    for (int i=0; i<10; i++)
        inserir_vetor(vetor, (i*10));
    imprime_vetor(vetor);

    printf("elemento maximo = %d\n", max_vetor(vetor));
    libera_vetor(vetor);
    return 0;
}
```

tad.c

```c
// Definição do TAD para pontos

struct st_meuVetor{
    int max;   // tamanho maximo
    int tam;   // tamanho usado
    int *v;    // vetor
};
typedef struct st_meuVetor meuVetor;
```

OBS Em aloca_vetor, primeiro alocar estrutura, depois vetor da estrutura

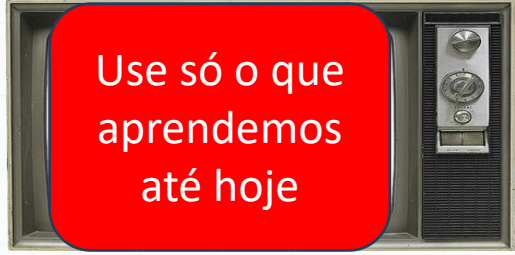
OBS2 Em inserir_vetor, inserir na primeira posição desocupada. Se estiver cheio, imprimir “cheio”



```c
// Definição do TAD para pontos

struct st_meuVetor{
    int max;   // tamanho maximo
    int tam;   // tamanho usado
    int *v;    // vetor
};
typedef struct st_meuVetor meuVetor;

meuVetor* aloca_vetor(int max)
{
   meuVetor* vetor = (meuVetor*) malloc(sizeof(meuVetor));
   vetor->v        = (int*) malloc(max * sizeof(int));

   vetor->max = max;
   vetor->tam = 0;
   return vetor;
}

void inserir_vetor(meuVetor* vetor, int el)
{
    if (vetor->tam < vetor->max)
    {
        vetor->v[vetor->tam] = el;
        vetor->tam++;
    }
    else
        printf("vetor ja esta cheio\n");
}
```

```c
void imprime_vetor(meuVetor* vetor)
{
    printf("vetor = ");
    for (int i=0; i<vetor->max; i++)
        printf("%d, ", vetor->v[i]);
    printf("\n");
}

int max_vetor(meuVetor* vetor)
{
    if (vetor->tam >0)
    {
        int maximo=vetor->v[0];
        for (int i=1; i<vetor->max; i++)
            if (vetor->v[i] > maximo)
                maximo = vetor->v[i];
        return maximo;
    }
    else
    {
        printf("vetor vazio\n");
        return -1;
    }
}

void libera_vetor(meuVetor* vetor)
{
    free(vetor);
}
```

Exercicio



4) Considerando a estrutura isótopo usada para representar o elemento com o símbolo e a massa.

Faça um programa que receba um número (inteiro) n e armazene num vetor:

iso vet[n];

apenas os isótopos que são distintos tanto em massa quanto em símbolo. No fim imprima a quantidade de isótopos distintos e quais são.

O programa deve usar uma função:

int validaiso(char s, int m, int tam,iso v[]);

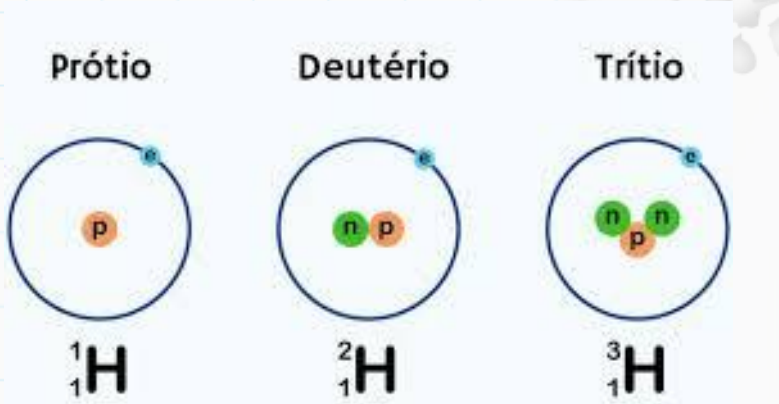
Que checa se o isótopo de símbolo s e massa m está presente no vetor v de tamanho tam (onde tam<=n) . A função retorna 1 se o isótopo não está presente e 0 caso contrário.

```
struct isotopo{
        char sim;
        int mas;
};
typedef struct isotopo iso;
```

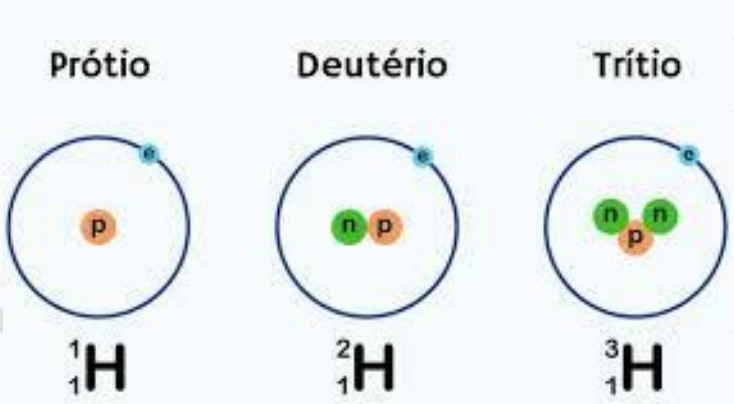
Exemplo:

```
Numero de isotopos:5
   simbolo=:H
   massa=:100
   simbolo=:H
   massa=:250
   simbolo=:O
   massa=:350
   simbolo=:H
   massa=:250
   simbolo=:N
   massa=:100
```

```
diferentes = 4
H) 100
H) 250
O) 350
N) 100
```



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

struct isotopo{
    char sim;
    int mas;
};
typedef struct isotopo iso;


int validaiso(char s, int m, int tam,iso v[]){
    if(tam != 0){
        for(int i=0;i<tam;i++){
            if((s == v[i].sim) && (m == v[i].mas)){
                return 0;
            }
        }
    }
    return 1;
}
```




```c
int main(void){
    int n;

    printf("Numero de isotopos:");
    scanf("%d",&n);
    printf("\n");

    char simbolo;
    int massa,cont=0;
    iso vet[n];

    for (int i=0; i<n; i++)
    {
        printf("   simbolo=:");
        scanf(" %c",&simbolo);
        printf("   massa=:");
        scanf("%d",&massa);

        if(validaiso(simbolo, massa,cont,vet)==1){
            vet[cont].sim=simbolo;
            vet[cont].mas=massa;
            cont++;
        }
    }
    printf("\ndiferentes = %d\n",cont);
    for (int i=0; i<cont; i++)
        printf("%c) %d\n",vet[i].sim, vet[i].mas);
}
```

Exercicio

5) Considerando a estrutura Aluno usada para representar o cadastro de alunos de uma disciplina. Faça um programa que receba um número (inteiro) n e armazene as informações de n alunos em um vetor de estruturas Aluno:

```
Aluno* turma[n];   // vetor de ponteiros para estrutura aluno
```

Para cada aluno, você deve criar (malloc) uma nova estrutura aluno no vetor turma e armazenar suas informações.

O programa deve chamar uma função:

```
void imprime_aprovados(int n, Aluno** turma)
```

que irá exibir na tela o nome a matricula e a média de todos os alunos que foram aprovados na disciplina.(Média ≥ 6)

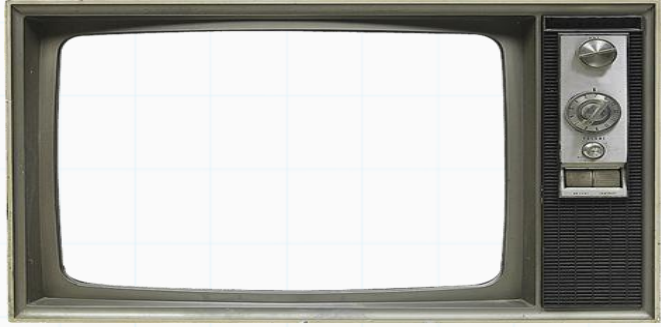
```
struct aluno{
  char nome[81];
  char matricula[8];
  float p1;
  float p2;
  float p3;
};
typedef struct aluno Aluno;
```

Exemplo:

```
Numero de alunos:3
    nome do aluno 0:Chaves
    mat do aluno 0:001
    notas do aluno 0:3 2 1
    nome do aluno 1:Madruga
    mat do aluno 1:002
    notas do aluno 1:3 3 0
    nome do aluno 2:Kiko
    mat do aluno 2:003
    notas do aluno 2:9 8 7
```

```
APROVADOS:
        Kiko mat=003:
        media = 8.00
```

Até a próxima



Slides baseados no curso de Aline Nascimento