Fazer um programa completo que leia dois valores, n e p, e calcule a combinação de n elementos, p a p.
Fórmula:

n!/(p! * (n-p)!)

- *Função void* para calcular a fatorial
- *Função com retorno* para calcular a combinação

```
// implementa combinações:
#include <stdio.h>
#include <stdlib.h>

// função que devolve a fatorial de N  na variável fat:
void fatorial (int n , float *fat)
{
}

// utiliza a função fatorial
float combinacoes (int n, int p) // retorna o resultado
{
}

int main( )
.....
```



```
// Implementa combinações:
#include <stdio.h>
#include <stdlib.h>

// função void que calcula o fatorial de N
void fatorial (int n , float *fat)
{
  int i=2;
  *fat = 1;
  for (i = 2; i<= n; i++)
       *fat = *fat * i;

}

// função que devolve as combinações de N elementos, P a P
float combinacoes (int n, int p)
//fórmula: n! / ( p! * ( n-p)! )
// utiliza o procedimento fatorial:
{
   float fn, fp, fn_p; //para armazenar fatoriais calculados
   fatorial (n,&fn);
   fatorial (p,&fp);
   fatorial (n - p,&fn_p);
   return fn / ( fp * fn_p ); // retorna
}
```



```
// programa principal:
int main()
{
 int n, p;
 char resp;
 do
  {
    do // consiste se é possível:
       {
          printf("Digite n e p, com n >= p: ");
          scanf("%d%d",&n,&p);
       }
    while (n < p || p <= 0 || n<=0); //ou (!(n >= p && p>0)) - implica n > 0
    printf("combinacoes de %d elementos %d a %d = %.0f",
                                  n, p, p, combinacoes (n, p));
    printf("\nOutra? (s/n)");
    fflush(stdin);
    resp= getchar( );  // não consistiu....
  } while (resp!='n');

 system("pause");
 return 0;
}
```

Alterar o programa anterior, de forma que a função fatorial seja implementada como função com retorno:

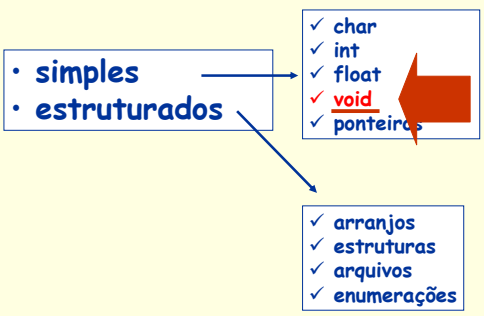
```
// Implementa combinações:
#include <stdio.h>
#include <stdlib.h>

// função que devolve o fatorial de N
float fatorial (int n)
{
  int fat = 1, i=2;
  for (i = 2;i<= n;i++)
       fat = fat * i;
  return fat;
}

// função que devolve as combinações de N elementos, P a P
float combinacoes (int n, int p)
//fórmula: n! / ( p! * ( n-p)! )
// utiliza o procedimento fatorial:
{
   return fatorial(n) / (fatorial (p) * fatorial (n – p)); // retorna result. expressão
}

int main( ) // permanece inalterada
.....
```


Tipos de dados - C



Tipo void

- Associado a funções.
- Indica tipo vazio, isto é, indicam funções que não retornam nada (portanto, sem o comando return) e/ou ausência de parâmetros na função.

Tipos de dados - C

- **simples**
- **estruturados**

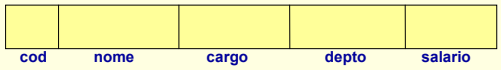
- ✓ char
- ✓ int
- ✓ float
- ✓ void
- ✓ ponteiros

- ✓ arranjos
- ✓ estruturas
- ✓ arquivos
- ✓ enumerações

Estruturas

- Tipo de dado estruturado que permite agrupar informações com características diferentes, isto é, elementos com tipos de dados diferentes.
- Cada unidade de informação é um campo do registro

funcionário:

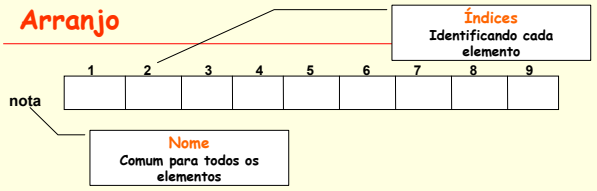


Diferença entre arranjo e estrutura

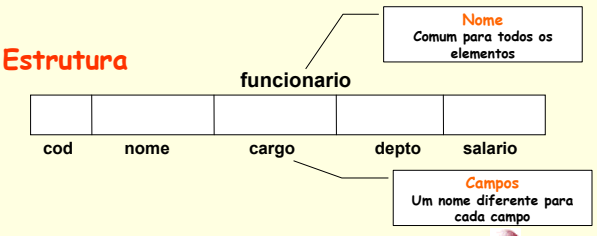
Arranjo (homogêneo)
- Todos os elementos do mesmo tipo.
- Um só nome, com índice para identificar cada elemento.

Estrutura (heterogêneos)
- Elementos podem ser de tipos diferentes
- Cada elemento é identificado através de um nome único - campo.

Arranjo



Estrutura



Declaração de tipo estrutura

nome

```
struct  identificador
    {    tipo1  campo1, campo2;
         tipo2  campo3;
          ...
         tipo_n  campo_n;
    };
```

O que está sendo declarado em cada exemplo é um tipo estrutura!

Ainda não existe qualquer variável declarada que utilize este tipo.

Exemplos:

Tipos estrutura

```
struct funcionario
    {
       int    cod;
       char   nome[30];
       char   cargo[10];
       int    depto;
       float  salario;
    };
```

```
struct tres_num
    {
      int c1;
      float c2;
      float c3;
    };
```


Declaração de variáveis tipo estrutura: em paralelo com a declaração do tipo

sem nome

```
struct { tipo1   campo1, campo2;
         tipo2   campo3;
         ...
         tipo_n  campo_n;
       } variavel1, variavel2, ... ;
```

Exemplos:

```
struct {
         int    cod;
         char   nome[30];
         char   cargo[10];
         int    depto;
         float  salario;
       } func1, func2;
```

Variáveis tipo estrutura.

Os respectivos tipos não têm nome.

Neste caso, não podem ser parâmetro de função.

```
struct
    {
      int c1;
      float c2;
      float c3;
    } var_3_num;
```

Declaração de variáveis tipo estrutura: de forma independente da declaração do tipo

A palavra struct pode ser omitida

```
struct  identificador
     {    tipo1  campo1, campo2;
          tipo2  campo3;
          ...
          tipoN  campoN;
     };
struct identificador  variavel1, variavel2, … ;
```

Exemplos:

```
struct  funcionario
        {
          int     cod;
          char    nome[30];
          char    cargo[10];
          int     depto;
          float   salario;
        } ;
struct funcionario func1, func2;
```

```
struct numeros
        {
          int c1;
          float c2;
          float c3;
        }
numeros tres_num;
```


Inicialização de estrutura na declaração

Junto com a declaração da estrutura

```
struct nome_da_estrutura
  {
     tipo1   campo1, campo2;
     . . .
     tipon   campo;
  } v1 = {valor1, valor2, ..., valorn};
```

Os valores entre chaves correspondem aos valores da estrutura, pela ordem em que foram descritos na definição.

Após a declaração da estrutura

```
struct nome_da_estrutura v1 = {valor1, valor2, ..., valorn};
```


Exemplo

```
struct tipo_funcionario
            {
                int cod;
                char nome[30];
                char cargo[10];
                int depto;
                float salario;
             };

struct  tipo_funcionario  funcionario  =
   { 1432, “Jose Costa”, “Gerente”, 2,  4656.00 };
```


Acesso aos dados da estrutura

Campos são acessados individualmente com a notação:

```
< nome da variável >.< nome do campo >
```

Ex:

```
struct
   {
     int c1;
     float c2;
     float c3;
   } tres_num;
```

```
...
tres_num.c1 = 10;
tres_num.c2 = 35.7;
...
tres_num.c2 = 7.5 * tres_num.c1;
...
scanf (“%d”, &tres_num.c1);
printf (“%d”, tres_num.c1);
...
tres_num.c3 = tres_num.c2;
...
```


Preenchimento de campos por leitura

```
int main ( )
{
...
 struct tipo_funcionario // estrutura e variável locais
            {
                int cod;
                char nome[30];
                char cargo[10];
                int depto;
                float salario;
            };
 struct tipo_funcionario funcionario;
 printf("Forneca os dados do(a) funcionario(a):\n");
 printf(" \nCodigo: ");
 scanf  ("%d", &funcionario.cod);
 fflush(stdin);
 printf("\nNome: ");
 gets(funcionario.nome); // scanf pára no 1º branco encontrado!!!
 fflush(stdin);
 printf("\nCargo: ");
 gets(funcionario.cargo);
 printf("\nDepto: ");
 scanf ("%d", &funcionario.depto);
 printf("\nSalario: ");
 scanf ("%f", &funcionario.salario);
 ...
 return 0;
}
```


Escrita de conteúdo de campos

```
...
int main ( )
{
 ...
 struct tipo_funcionario
            {
                int cod;
                char nome[30];
                char cargo[10];
                int depto;
                float salario;
            };
 struct tipo_funcionario funcionario;
 printf("\nInformacoes do(a) funcionario(a) de codigo %d:\n",
        funcionario.cod);
 printf ("\nNome: %s", funcionario.nome);
 printf ("\nCargo: %s",funcionario.cargo);
 printf ("\nDepto: %d", funcionario.depto);
 printf  ("\nSalario: %6.2f\n", funcionario.salario);
 ...
 return 0;
}
```

Variável tipo *struct* com campo *struct*

```
struct tipo_endereco
      {
        char rua [50];
        int numero;
        char bairro [20];
        char cidade [30];
        char sigla_estado [3];
        char cep[6];
      };
struct ficha_pessoal
      {
        char nome [50];
        char telefone[16];
        struct tipo_endereco endereco;
      };
struct ficha_pessoal ficha;
```

Referência aos campos:

nome da variável . nome do campo . nome do campo

Variável tipo *struct* com campo *struct:* leitura de dados

```
struct tipo_endereco
      {
        char rua [50];
        int numero;
        char bairro [20];
        char cidade [30];
        char sigla_estado [3];
        char cep[6];
      };
struct ficha_pessoal
      {
        char nome [50];
        char telefone[16];
        struct tipo_endereco endereco;
      };
struct ficha_pessoal ficha;
```

```
printf("Forneca os dados de uma pessoa:\n");
fflush(stdin);
printf("\nNome: ");
gets(ficha.nome);
printf("\nTelefone: ");
gets(ficha.telefone);
fflush(stdin);
printf("\nRua: ");
gets(ficha.endereco.rua);
printf("\nNumero: ");
scanf ("%d", &ficha.endereco.numero);
fflush(stdin);
printf("\nBairro: ");
gets(ficha.endereco.bairro);
fflush(stdin);
printf("\nCidade: ");
gets(ficha.endereco.cidade);
fflush(stdin);
printf("\nSigla do estado: ");
gets(ficha.endereco.sigla_estado);
printf("\nCEP: ");
gets (ficha.endereco.cep);
```

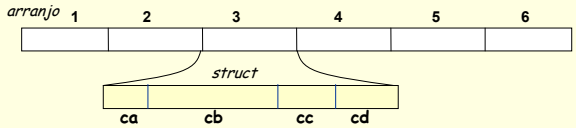

Variável tipo *struct* com campo *struct:* escrita de dados

```
struct tipo_endereco
      {
        char rua [50];
        int numero;
        char bairro [20];
        char cidade [30];
        char sigla_estado [3];
        char cep[6];
      };
```

```
struct ficha_pessoal
      {
        char nome [50];
        char telefone[16];
        struct tipo_endereco endereco;
      };
struct ficha_pessoal ficha;
```

```
printf("\nInformacoes de uma pessoa\n");
printf ("\nNome: %s", ficha.nome);
printf ("\nTelefone: %s",ficha.telefone);
printf ("\nRua: %s", ficha.endereco.rua);
printf ("\nNumero: %d", ficha.endereco.numero);
printf ("\nBairro: %s", ficha.endereco.bairro);
printf ("\nCidade: %s", ficha.endereco.cidade);
printf ("\nSigla do estado: %s", ficha.endereco.sigla_estado);
printf ("\nCEP: %s\n", ficha.endereco.cep);
```


Arranjo de estruturas



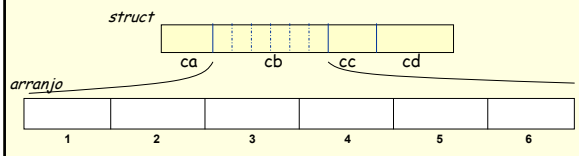

```
struct x {
            int ca;
            float cb;
            char cc;
            int cd;
          };
struct x arranjo[6];
int ind;
  ...
```

```
scanf ("%f",&arranjo[ind].cb );

for (ind=0;ind==5;ind++)
      scanf("%d",&arranjo[ind].ca);

scanf ("%d", &arranjo[2].ca);
scanf ("%c", &arranjo[2].cc);
scanf ("%f", &arranjo[2].cb);
...
```


Estruturas com campos arranjo




```
struct x
      {
        int ca;
        float cb[6];
        char cc;
        int cd;
      };
struct x var_reg;
int i;
```

```
scanf ("%d", &var_reg.ca );

for (ind= 0; ind==5;ind++)
      scanf ("%f", &var_reg.cb [ind]);
```


Atribuição

Nome da **variável *struct*** pode ser utilizado sem os campos **somente** em comando de atribuição **entre duas variáveis do mesmo tipo.**

```
struct  ficha_pessoal  ficha1, ficha2;
  .......
ficha1  =  ficha2 ;  // atribui todos os campos
  .......
```

Ex:

```
...
struct data     {
                  int dia;
                  char mes[3];
                  int ano;
                };
struct pessoa {
                  char nome[10];
                  struct data  dia_admissao;
                  float salario:
                };
{
 struct data admissao; // apenas a data
 struct pessoa funcionario;

 scanf ("%s",funcionario.nome);
 admissao.dia = 1;
 admissao.mes = 'Jul';
 admissao.ano = 2001;
 funcionario.dia_admissao=admissao;

 printf("%d %s %d",funcionario.dia_admissao.dia,
         funcionario.dia_admissao.mes,
         funcionario.dia_admissao.ano);
   ...
```


Exemplo: arranjo de estrutura[1] com campo arranjo[2]

```
# include <stdio.h>
# include <stdlib.h>
# define MAXALUNOS 10
int main( )
{
 struct al
      {
              char nome[20];
              float nota[5]; //2
              float media;
              char conceito;
      };
 struct al um_aluno;   // dados de 1 aluno
 struct al alunos[MAXALUNOS]; //1 vetor com dados de MAXALUNOS
 int  ind;
 float media = 0;
 system("color 70");
 printf("\nNome do ultimo aluno:");
 fflush(stdin);
 gets(alunos[MAXALUNOS - 1].nome);
 um_aluno = alunos[MAXALUNOS - 1]; //atribui elemento estrutura à estrutura
 printf("\nNome do ultimo aluno = %s\n", um_aluno.nome);
 ...
```



```
// Leitura das notas da primeira prova de todos os alunos
for (ind=0;ind<MAXALUNOS;ind++)
{
  printf("\nNota da primeira prova do aluno %d: ", ind + 1);
  scanf ( "%f", &alunos [ind].nota[0] );
}
// Media da primeira prova dos alunos
// media declarada e inicializada com 0 no início do programa
for (ind = 0; ind < MAXALUNOS; ind++)
    media =  media + alunos[ind].nota[0];
media =  media / MAXALUNOS;
printf("\nMedia da primeira prova: %6.2f\n", media);
system("pause");
return 0;
}
```

Operador seta: ->

- Utilizado em funções com parâmetro do tipo estrutura, onde ponteiros devem ser utilizados.
- Indica o acesso a um campo de uma estrutura por meio de um ponteiro para esta estrutura.
- Escrever aluno-> media é o mesmo que escrever (*aluno).media.
- O parênteses é necessário porque o • (ponto) tem prioridade maior que o *.
- Atenção: a notação com o operador seta é a de uso mais freqüente.

Ex: arranjo de estruturas com campo arranjo

```
- //Leitura e escrita de arranjo de estrutura, com passagem por endereço
 # include <stdio.h>
 # include <stdlib.h>
 #define MAXTUR 5
 #define MAXNOTAS 5
 struct al
       {
              char nome[20];
              float nota[MAXNOTAS];
              float media;
              char conceito;
       } ;
 void lealunos(al *, int); // recebe ponteiro para tipo struct (palavra opcional)
 void escrevealunos(struct al,  int);  // recebe conteúdo
 int main( )
 {
  int i;
  struct al turma[MAXTUR]; // elemento passado como parâmetro
  system("color 70");
  for (i = 0; i < MAXTUR; i++)
     lealunos(&turma[i], MAXNOTAS); // manda endereço como argumento
  for (i = 0; i < MAXTUR; i++)
    escrevealunos(turma[i],MAXNOTAS); // manda conteúdo como argumento
  printf("\n");
  system("pause");
  return 0;
 }
```


Ex:

```
void lealunos(al *aluno, int numnotas) // recebe ponteiro
 {
  int j;
  fflush(stdin);
  printf("\nNome: ");
  gets(aluno->nome); //aponta para posição: conteúdo
  aluno->media = 0;
  for (j = 0; j < numnotas; j++)
   {
     printf("\nNota %d: ",  j + 1);
     scanf ( "%f", &(aluno->nota[j])); // leitura para endereço apontado
     aluno->media += aluno->nota[j];
   }
 aluno->media = aluno->media / numnotas;
   if (aluno->media > 8.9)
      aluno->conceito = 'A';
   else
       if (aluno->media > 7.5)
          aluno->conceito = 'B';
       else
          if (aluno->media > 5.9)
             aluno->conceito = 'C';
          else
             aluno->conceito = 'D';
 }
```

Ex:

```
void escrevealunos(al aluno, int numnotas) // recebe conteúdo
 {
   int j;
   printf("\n%21s ", aluno.nome);
   for (j = 0; j < numnotas; j++)
      printf("%5.2f ",aluno.nota[j]);
   printf("%5.2f %c", aluno.media, aluno.conceito);
 }
```


Exercício

Defina em C um novo tipo denominado **Passagem**, conforme *layout* do bilhete de passagem de ônibus mostrado abaixo.

Viação VIAJE BEM

Cod. Empresa: _____ Nome do Passageiro: _____
Origem: _____ Destino: _____
Data: ___/___/___ Horário: ____:____
Assento:_____ Valor do bilhete: _____ .


```
/*  Name: viajebem.cpp        Author: Cida Souto.
    Date: 20/05/08 00:07      Description: definir tipo
    bilhete de passagem de onibus.   */
#include <stdio.h>
#include <stdlib.h>
// struct como parâmetro: precisa ser global!!!
struct data
  {
    int dia, mes, ano;
  };
struct horario
  {
    int hora, min;
  };
struct bilhete
  {
    int cod;
    char nome_p[20];
    char origem[20], destino[20];
    struct data data_b;
    struct horario horario_b;
    int assento;
    float valor;
  };
int main ( )
{

} // fim de main
```

Viação VIAJE BEM

Cod. Empresa: _____ Nome do Passageiro:
Origem: _____ Destino: _____
Data: ___/___/___ Horário: ____:____
Assento:_____ Valor do bilhete: ___

Exercício

A partir da estrutura definida no item 1, implemente as funções ler_bilhete e mostrar_bilhete, que permitem, respectivamente, ler e mostrar todos os dados relativos a um determinado bilhete.

Observe que as funções devem receber a estrutura bilhete como parâmetro, o que implica o uso de ponteiros.


```
// função que efetua a leitura dos dados de uma passage
  void ler_bilhete (struct est_bilhete *bilhete ) // pont
 {
   printf ("Cod. da Empresa: ");
   scanf ("%d", &bilhete->cod);
   fflush(stdin);
   printf ("Nome do Passageiro: ");
   gets ((*bilhete).nome_p);
   printf ("Origem: ");
   gets (bilhete->origem);
   printf ("Destino: ");
   gets (bilhete->destino);
   printf ("Data do Bilhete (dd mm aa): ");
   scanf ("%d%d%d", &bilhete->data_b.dia, &bilhete->data_b.mes,
                                          &bilhete->data_b.ano);
   printf ("Horario do Bilhete (hh mm): ");
   scanf ("%d%d", &(*bilhete).horario_b.hora, &bilhete->horario_b.min);
   printf ("Assento: ");
   scanf ("%d", &bilhete->assento);
   printf ("Valor do Bilhete: ");
   scanf ("%f", &bilhete->valor);
 }
```

```
struct data
  {
    int dia, mes, ano;
  };
struct horario
  {
    int hora, min;
  };
struct est_bilhete
  {
    int cod;
    char nome_p[20];
    char origem[20], destin
    struct data data_b;
    struct horario horario_
    int assento;
    float valor;
  };
```



```
// função que efetua a impressão dos dados de uma passagem
  void mostra_bilhete ( struct est_bilhete *bilhete )
  {
    printf ("\nCod. da Empresa: %d", bilhete->cod);
    printf ("\nNome do Passageiro: %s",bilhete->nome_p);
    printf ("\nOrigem: %s ",bilhete->origem);
    printf ("\nDestino: %s ",bilhete->destino);
    printf ("\nData do Bilhete: %d/%d/%d", bilhete->data_b.dia,
                                          bilhete->data_b.mes, bilhete->data_b.ano);
    printf ("\nHorario do Bilhete: %d:%d hs.", bilhete->horario_b.hora,
                                          bilhete->horario_b.min);
    printf ("\nAssento: %d", bilhete->assento);
    printf ("\nValor do Bilhete: R$ %.2f \n\n", bilhete->valor);
  }
```

```
struct data
  {
    int dia, mes, ano;
  };
struct horario
  {
    int hora, min;
  };
struct est_bilhete
  {
    int cod;
    char nome_p[20];
    char origem[20], dest
    struct data data_b;
    struct horario horario
    int assento;
    float valor;
  };
```

```
// programa principal:
 int main( )
{
 struct est_bilhete passagem;
 system("color f1");
 ler_bilhete(&passagem);
 mostra_bilhete(&passagem);
 system("pause");
 return 0;
}
```

```
/*  Name: viajebem.cpp        Author: Cida Souto.
    Date: 20/05/08 00:07      Description: definir tipo
    bilhete de passagem de onibus.   */
#include <stdio.h>
#include <stdlib.h>
// struct como parâmetro: precisa ser global!!!
struct data
   {
     int dia, mes, ano;
   };
struct horario
   {
     int hora, min;
   };
struct est_bilhete
   {
     int cod;
     char nome_p[20];
     char origem[20], destino[20];
     struct data data_b;
     struct horario horario_b;
     int assento;
     float valor;
   };
void ler_bilhete (struct est_bilhete *bilhete )
.....
void mostra_bilhete ( struct est_bilhete *bilhete )
......

int main ( )
{

} // fim de main
```


Impressão: não precisa receber ponteiro!!!!

```
// função que efetua a impressão dos dados de uma passagem, sem ponteiro:
 void mostra_bilhete ( struct est_bilhete bilhete ) // recebe o conteúdo
 {
  printf ("\nCod. da Empresa: %d", bilhete.cod);
  printf ("\nNome do Passageiro: %s",bilhete.nome_p);
  printf ("\nOrigem: %s ",bilhete.origem);
  printf ("\nDestino: %s ",bilhete.destino);
  printf ("\nData do Bilhete: %d/%d/%d", bilhete.data_b.dia,
                                         bilhete.data_b.mes, bilhete.data_b.ano);
  printf ("\nHorario do Bilhete: %d:%d hs.", bilhete.horario_b.hora,
                                             bilhete.horario_b.min);
  printf ("\nAssento: %d", bilhete.assento);
  printf ("\nValor do Bilhete: R$ %.2f \n\n", bilhete.valor);
 }
```

```
 // programa principal:
  int main( )
 {
  struct est_bilhete passagem;
  system("color f1");
  ler_bilhete(&passagem);        // passa endereço
  mostra_bilhete(passagem);   // passa conteúdo
  system("pause");
  return 0;
 }
```