Linguagem de Programação II

Prof. Mario Bessa

Aula 22

http://mariobessa.info

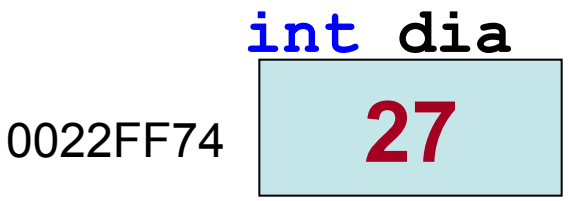
Variáveis

- **Introdução:**
  - Uma variável é um espaço da memória principal reservado para armazenar dados.
  - Variáveis possuem:
    - **Nome:**
      - Identificador usado para acessar o conteúdo.
    - **Tipo:**
      - Determina a capacidade de armazenamento.

        Ex: int, char, float, ...
    - **Endereço:**
      - Posição na memória principal.

Ponteiros

- **Exemplo:**
  - **Nome:** `dia`
  - **Tipo:** `int`
  - **Endereço:** `0022FF74` `(hexadecimal)` **ou**
    `2293620` `(decimal)` **ou**
    `&dia` `(representação simbólica)`
  - **Conteúdo:** `27`



Ponteiros

- **Definição:**
  - Ponteiros são tipos especiais em C que servem para armazenar endereços de memória.
  - Os ponteiros também possuem um tipo, pois ao armazenar o endereço, é preciso identificar qual é o tipo de valor que será armazenado.

- **Declaração (sintaxe):**

```
tipo *nome_do_ponteiro;
```

  - O asterisco indica ao compilador que a variável não irá guardar um valor, mas apenas um endereço especificado.

Ponteiros

– É possível também descobrir o valor do endereço de uma variável. Para isso, basta usar o operador & antes da variável para pegar o endereço.

- **Exemplos:**

```
int *p;       //declara ponteiro para um int.
char *tmp;    //declara ponteiro para um char.
float *pont; //declara ponteiro para um float.
```

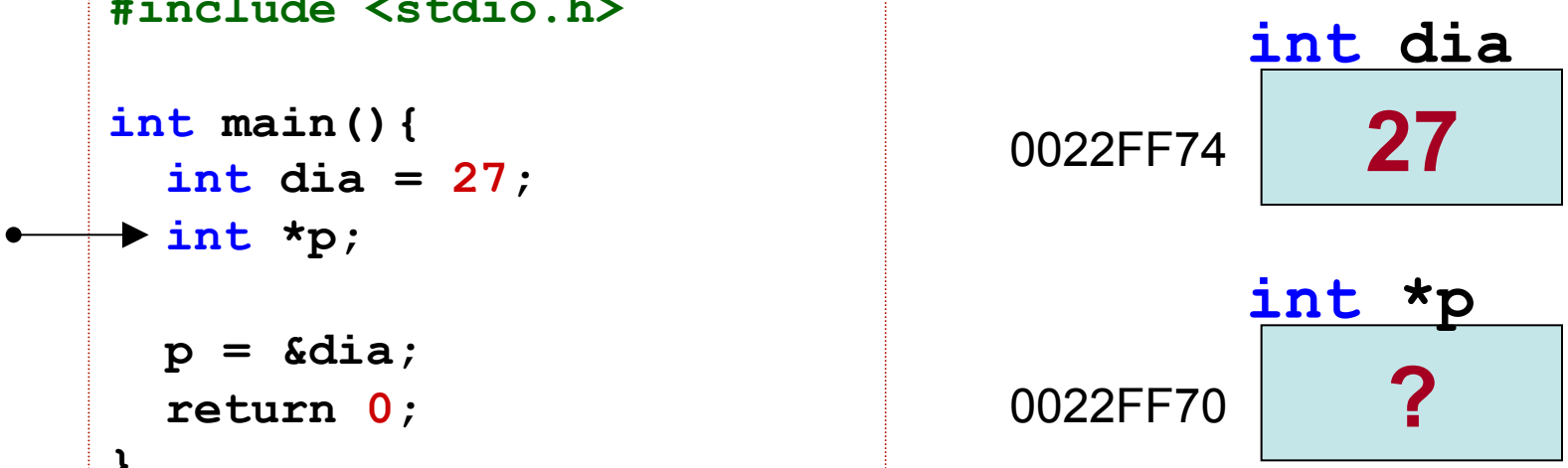
Ponteiros

- **Exemplo:**

```
#include <stdio.h>

int main(){
   int dia = 27;
   int *p;

   p = &dia;
   return 0;
}
```



- **As variáveis são declaradas.**
- **O ponteiro, como toda variável, também possui um endereço de memória (&p = 0022FF70).**

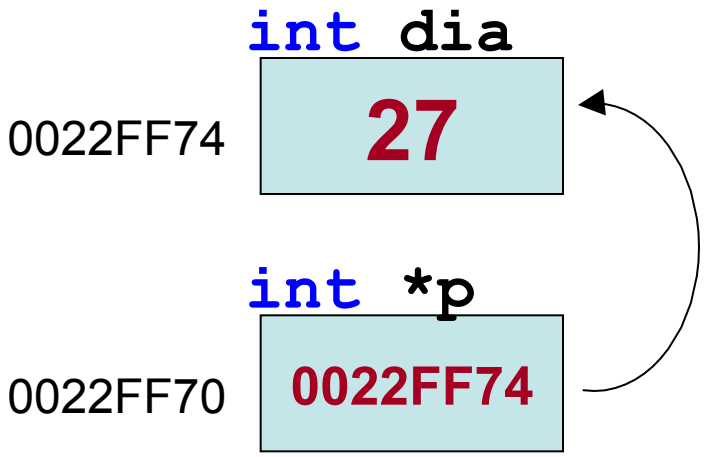
Ponteiros

- **Exemplo:**

```
#include <stdio.h>

int main(){
   int dia = 27;
   int *p;

   p = &dia;
   return 0;
}
```



- **O endereço de `dia` é atribuído para o ponteiro `p`.**
- **Dizemos que `p` aponta para a variável `dia` (graficamente representado por uma seta).**

Exemplos Ponteiros

```
#include <stdio.h>

int main (){

        int num,valor;
        int *p;
        num=55;
        p=&num;
        valor=*p;
        printf("%d - %d - %d\n", num, *p, valor);
        printf("%p - %p - %p\n", &num, p, &valor);
        return(0);
}
```

```
55 - 55 - 55
0x7fff5fbff848 - 0x7fff5fbff848 - 0x7fff5fbff844
```

Ponteiros

- Cuidado no uso:
  - O principal cuidado do programador no uso de ponteiros é saber para onde ele está apontando, ou seja, nunca utilize um ponteiro que não foi inicializado.
  - Os ponteiros sempre devem ser inicializados com um endereço de memória, com 0 (zero) ou com a palavra-reservada NULL.

```
#include <stdio.h>

int main () /* Errado - Nao Execute */
{
        int x,*p;
        x=13;
        *p=x;
        return(0);
}
```

Ponteiros

- **Por que usar ponteiros?**
  - Nos exemplos até agora, o acesso ao conteúdo das variáveis se dava através do nome delas.
  - Ponteiros nos fornecem um novo modo de acesso que explora o endereço das variáveis.
  - Para isso usamos o operador indireto (*), que nos permite ler e alterar o conteúdo das variáveis apontadas por um ponteiro.

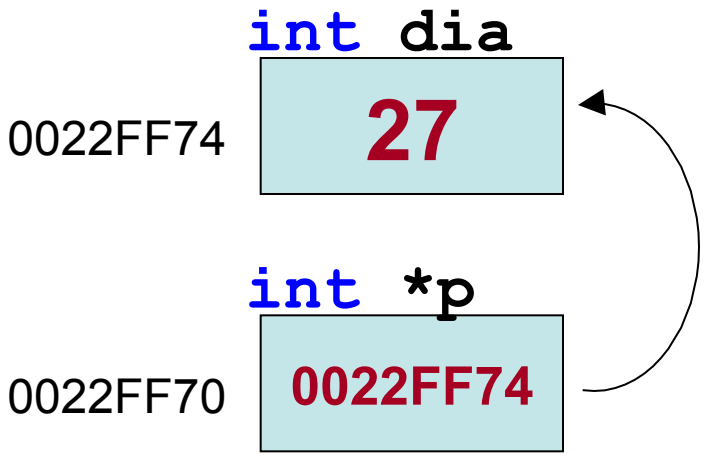
Ponteiros

- **Exemplo:**

```
#include <stdio.h>

int main(){
   int dia = 27;
   int *p;

   p = &dia;
   *p = 10;
   return 0;
}
```



- O endereço de `dia` é atribuído para o ponteiro `p`.
- Dizemos que `p` aponta para a variável `dia` (graficamente representado por uma seta).

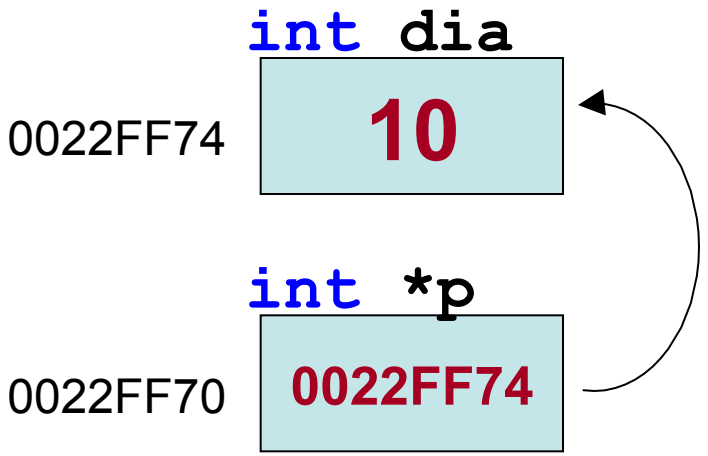
Ponteiros

- **Exemplo:**

```
#include <stdio.h>

int main(){
   int dia = 27;
   int *p;

   p = &dia;
   *p = 10;
   return 0;
}
```



- O código `*p` é o conteúdo da variável apontada por `p`, ou seja o conteúdo de `dia`, que recebe o valor `10`.

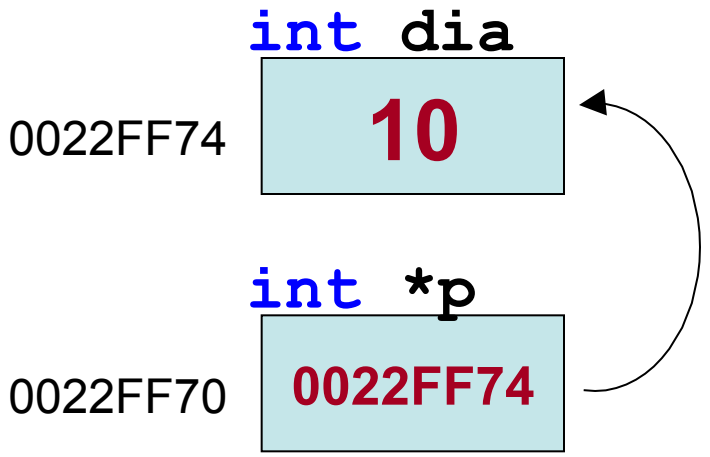
Ponteiros

- **Exemplo:**

```
#include <stdio.h>

int main(){
   int dia = 27;
   int *p;

   p = &dia;
   *p = 10;
   return 0;
}
```



- A declaração `int *p;` indica que a variável `p` é um ponteiro para um inteiro e que `*p` é do tipo `int`.

Ponteiros

- **Por que usar ponteiros?**
  - Variáveis simples e estruturas são passadas por valor para funções. Ou seja, é gerada uma cópia da variável e alterações na função não produzem qualquer efeito externo.
  - Com o uso de ponteiros é possível realizar a passagem dos valores por referência.

Ponteiros

- **Exemplo: Função que lê um valor inteiro.**

```
#include <stdio.h>

void LeInteiro(int a){
  printf("Entre com a: ");
  scanf("%d",&a);
}

int main(){
  int a=0;
  LeInteiro(a);
  printf("a: %d\n",a);
  return 0;
}
```

- O código acima irá imprimir 0 na saída padrão sempre. O problema é que o `scanf` altera apenas uma variável local da função que deixa de existir após a sua execução.
- A variável `a` da função principal permanece intacta.

Ponteiros

- **Exemplo: Função que lê um valor inteiro.**

```c
#include <stdio.h>

void LeInteiro(int a){
  printf("Entre com a: ");
  scanf("%d",&a);
}

int main(){
  int a=0;
  LeInteiro(a);
  printf("a: %d\n",a);
  return 0;
}
```

```c
#include <stdio.h>
int LeInteiro(){
  int a;
  printf("Entre com a: ");
  scanf("%d",&a);
  return a;
}
int main(){
  int a=0;
  a = LeInteiro();
  printf("a: %d\n",a);
  return 0;
}
```

- Uma possível solução é apresentada a direita. Porém funções só podem retornar um único valor e em casos onde é necessário alterar mais de uma variável o código não se aplica.

Ponteiros

- **Exemplo: Função que lê um valor inteiro.**

```c
#include <stdio.h>

void LeInteiro(int a){
  printf("Entre com a: ");
  scanf("%d",&a);
}

int main(){
  int a=0;
  LeInteiro(a);
  printf("a: %d\n",a);
  return 0;
}
```

```c
#include <stdio.h>
void LeInteiro(int *p){
  int a;
  printf("Entre com a: ");
  scanf("%d",&a);
  *p = a;
}
int main(){
  int a=0;
  LeInteiro(&a);
  printf("a: %d\n",a);
  return 0;
}
```

- Uma solução usando ponteiros é mostrada. O ponteiro `p` é inicializado com o endereço da variável `a` da função principal. Logo, `*p` é o próprio conteúdo de `a` da função principal.
- Dizemos que `a` foi passada por referência.

Ponteiros

- **Exemplo Real: Função que troca os valores de duas variáveis.**

```c
#include <stdio.h>
void troca(int *a, int *b){
  int tmp;
  tmp = *a;
  *a = *b;
  *b = tmp;
}
int main(){
  int a=25,b=12;
  troca(&a, &b);
  printf("a: %d, b: %d\n",a,b);
  return 0;
}
```

- Vimos que operações de troca são importantes em algoritmos como o Bubble sort. A função acima recebe o endereço de duas variáveis e usando uma variável temporária procede com a troca dos valores. A saída do programa será "a: 12, b: 25".

Ponteiros

- É possível também realizar operações de incremento e decremento com os ponteiros.
- Quando incrementamos um ponteiro ele passa a apontar para o próximo valor do mesmo tipo para o qual o ponteiro aponta. Ou seja, se temos um ponteiro para um inteiro e o incrementamos ele passa a apontar para o próximo inteiro. Esta é a razão pela qual o compilador precisa saber o tipo de um ponteiro.
- A operação de decremento funciona de forma semelhante, mas com a diminuição dos endereços.
- Exemplo de incremento de ponteiro: p++
- Vale dizer que p++ é diferente de (*p)++. O primeiro representa um incremento no endereço do ponteiro p e o segundo representa um aumento do conteúdo do ponteiro p.
- Explique a diferença entre: p++; (*p)++; *(p++);

Exemplos Ponteiros

```
#include <stdio.h>
int main (){
        int num,*p;
        num=100;
        p=&num;
        printf("O valor de p é: %d.\n", *p);
        printf("O endereço de p é: %p.\n", p);
        (*p)++;
        printf("O valor do conteúdo de p incrementado é: %d.\n", *p);
        printf("O endereço de p com o conteúdo incrementado é: %p.\n", p);
        p++;
        printf("O valor do conteúdo de p incrementado é: %d.\n", *p);
        printf("O endereço de p com o endereço incrementado é: %p.\n", p);
        return(0);
}
```

O valor de p é: 100.
O endereço de p é: 0x7fff5fbff848.
O valor do conteúdo de p incrementado é: 101.
O endereço de p com o conteúdo incrementado é: 0x7fff5fbff848.
O valor do conteúdo de p incrementado é: 0.
O endereço de p com o endereço incrementado é: 0x7fff5fbff84c.

Ponteiros

- É possível também realizar operações relacionais entre ponteiros. Por exemplo, é possível:
  - verificar se dois ponteiros têm o endereços iguais (==) ou diferentes (!=).
  - Ainda é possível verificar se o endereço de um ponteiro é maior (>) ou menor (<) que o endereço de outro ponteiro. Um ponteiro maior que o outro significa que aquele está à direita, enquanto que um ponteiro menor do que o outro significa que aquele está à esquerda deste.

Exemplos Ponteiros

```
#include <stdio.h>
int main (){
    int num,*p,*aux;
    num=100;
    p=&num;
    aux=p;
    p++;
    if (aux < p) {
        printf("%p é menor que %p",aux,p);
    }
    return(0);
}
```

```
0x7fff5fbff848 é menor que 0x7fff5fbff84c
```

Exemplos Ponteiros

```
#include <stdio.h>
int main (){

    int num,*p,*aux;
    num=100;
    p=&num;
    aux=p;
    (*p)++;
    if (aux == p) {
        printf("%p é igual a %p",aux,p);
    }
    return(0);
}
```

```
0x7fff5fbff848 é igual a 0x7fff5fbff848
```

Ponteiros para ponteiros

- **Definição:**
  - Um ponteiro para ponteiro guarda o endereço de um ponteiro, sendo que este guarda o endereço de uma variável. Ou seja, um ponteio para ponteiro guarda um endereço que aponta para outro endereço.
  - Não existe limitação desse recurso.
- **Declaração (sintaxe):**

```
tipo **nome_do_ponteiro;
```

- **Exemplos:**

```
int **p;       //declara ponteiro para um ponteiro de um int.
char **tmp;    //declara ponteiro para um ponteiro de um char.
float **pont; //declara ponteiro para um ponteiro de um float.
```

Exemplos Ponteiros

```
#include <stdio.h>
int main(){

        int x = 12, *px, **ppx;
        px = &x;
        ppx = &px;
        printf("x= %i\n",x);
        printf("&x= %p\n\n",&x);
        printf("*px= %i\n", *px);
        printf("px= %p\n", px);
        printf("**ppx= %i\n",**ppx);
        printf("ppx= %p\n\n", ppx);
        return(0);
}
```

```
x= 12
&x= 0x7fff5fbff848

*px= 12
px= 0x7fff5fbff848

**ppx= 12
ppx= 0x7fff5fbff840
```

Exemplos Ponteiros

```
#include <stdio.h>

int main () {
            int x,*p,a=17;
            x=13;
            p=&a;
           *p=x;
            printf("a= %i",a);

            return(0);
}
```

```
a= 13
```

Ponteiros e vetores

- Foi mostrado que na passagem de vetores para funções especifica-se apenas o nome do vetor e que modificações nos elementos do vetor dentro da função chamada alteram os valores do vetor no programa chamador.
- Isto se deve ao fato de que, na linguagem C, **vetores** são intimamente relacionados a **ponteiros**
- Em C, o **nome** de um vetor é tratado como o **endereço** de seu primeiro elemento. Assim ao se passar o nome de um vetor para uma função está se passando o endereço do primeiro elemento de um conjunto de endereços de memória.

Ponteiros e vetores

- Podemos acessar o endereço de qualquer elemento do vetor do seguinte modo: &vet[i] é equivalente a (vet + i).
- (vet + i) não representa uma **adição** aritmética normal mas o **endereço** do *i-ésimo* elemento do vetor vet (endereço contado a partir do endereço inicial vet[0]).
- Do mesmo modo que se pode acessar o **endereço** de cada elemento do vetor por ponteiros, também se pode acessar o **valor** de cada elemento usando ponteiros.
- Assim vet[i] é equivalente a *(vet + i). Aqui se usa o operador conteúdo (*) aplicado ao endereço do *i-ésimo* elemento do vetor vet.

Ponteiros e vetores

```
#include <stdio.h>
int main(){
        float vet[5] = {1.1,2.2,3.3,4.4,5.5}; // declarando uma vetor real
        int i;   // declarando um contador (DEVE ser inteiro!)
        printf("cont.   valor        valor          endereco              endereco");
        for(i = 0 ; i <= 4 ; i++){
                printf("\ni = %d"           ,i);          // contador
                printf("   vet[%d] = %.1f"    ,i, vet[i]   ); // valor (com vetor)
                printf("   *(vet + %d) = %.1f",i, *(vet+i) ); // valor (com ponteiro)
                printf("   &vet[%d] = %p"     ,i, &vet[i]  ); // endereco (com vetor)
                printf("   (vet + %d) = %p"   ,i, vet+i    ); // endereco (com ponteiro)
        }
        puts("\n\nObserve que os ENDERECOS sao ESPACADOS de 4 em 4 bytes...");
   return 0;
}
```

Ponteiros e strings

- Uma *string* é um *conjunto ordenado de caracteres*.
- Em C, uma *string* é um **vetor unidimensional** de caracteres ASCII, sendo o ultimo destes elementos o caracter especial ’\0’.
- **Sintaxe:** As duas maneiras mais comuns de declararmos uma *string* são:
  - char nome[tam];
  - char *nome;
  - onde:
    nome é o nome do vetor de caracteres e *tam* seu tamanho.

Ponteiros e strings

```
#include <stdio.h>
#include <string.h>
int main(){
    char nome[80];              // vetor
    char *frase = "Ola, ";      // ponteiro
    int i;
    puts("Manipulacao de strings");
    puts("Digite seu nome:");
    i = 0;
    do{
        nome[i] = getchar();        // leitura da tecla
        if(nome[i] == '\n'){        // se pressionou [enter]...
            nome[i] = '\0';         // troca por \0
        }
     }while(nome[i++] != '\0');
     strcat(frase,nome);            // concatena duas strings
     printf("%s",frase);
     return 0;
}
```

Ponteiros

- Se i e j são variáveis inteiras e p e q ponteiros para int, quais das seguintes expressões de atribuição são inválidas?

  a) p = &i;

  b) *q = &j;

  c) p = &*&i;

  d) i = (*&)j;

  e) i = *&j;

  f) i = *&*&j;

  g) q = *p;

  h) i = (*p)++ + *q

Ponteiros

- Verifique o programa abaixo. Encontre o seu erro e corrija-o para que escreva o numero 10 na tela.

```
#include <stdio.h>
int main(){
    int x, *p, **q;
    p = &x;
    q = &p;
    x = 10;
    printf("\n%d\n", &q);
    return(0);
}
```

Ponteiros

- Quais serão os valores de x, y e p ao final do trecho de código abaixo?

```
int x, y, *p;
y = 0;
p = &y;
x = *p;
x = 4;
(*p)++;
--x;
(*p) += x;
```

Ponteiros

- Quais serão os valores de x, y e p ao final do trecho de código abaixo?

```
int x, y, *p;
y = 0;
p = &y; //*p = 0
x = *p; //x = 0
x = 4; //x = 4
(*p)++; //*p = 1, y = 1
--x; //x = 3
(*p) += x; //*p = 4, y = 4
```

Ao final, temos:

x = 3, y = 4, p apontando para y (*p = 4)

Ponteiros

- Os trechos de código abaixo possuem erros. Qual(is)? Como deveriam ser corrigido

```
void main() {
int x, *p;
x = 100;
p = x;
printf("Valor de p: %d.\n", *p);
}
```

Ponteiros

- Os trechos de código abaixo possuem erros. Qual(is)? Como deveriam ser corrigido

```
void main() {
int x, *p;
x = 100;
p = x;
/* p deveria receber o endereço de x, já que p é um ponteiro (e x não).
Ponteiros “armazenam” o endereço para o qual eles apontam! O código correto seria: */
p = &x;
printf(“Valor de p: %d.\n”, *p);
}
```

Ponteiros

- Programa em C para copiar um String sem usar ***strcpy.*** Criando sua própria função usando ponteiros.

```
#include<stdio.h>
void copy_string(char *target, char
*source){
   while(*source){
      *target = *source;
      source++;
      target++;
   }
   *target = '\0';
}
```

```
Int main(){
    char source[100], target[100];
    printf("Enter source string\n");
    gets(source);
    copy_string(target, source);
    printf("Target string is \%s\n",
target);
    return 0;
}
```

Ponteiros

- Programa em C para concatenar String sem usar ***strcat.*** Criando sua própria função usando ponteiros.

```
void concatenate_string(char *original,
char *add){
   while(*original)
      original++;
   while(*add){
      *original = *add;
      add++;
      original++;
   }
   *original = '\0';
}
```

```
int main(){
    char original[100], add[100];
    printf("Enter source string\n");
    gets(original);
    printf("Enter string to concatenate
\n");
    gets(add);
    concatenate_string(original, add);
    printf("String after concatenation
is %s\n", original);
    return 0;
}
```

Ponteiros

- Programa em C para comparar dois Strings usando ponteiros.

```
#include<stdio.h>
int compare_string(char *first, char
*second)
{
   while(*first==*second)
   {
      if ( *first == '\0' || *second == '\0' )
         break;

      first++;
      second++;
   }
   if( *first == '\0' && *second == '\0' )
      return 0;
   else
      return -1;
}
```

```
int main(){
    char first[100], second[100],
result;
    printf("Enter first string\n");
    gets(first);
    printf("Enter second string\n");
    gets(second);
    result = compare_string(first,
second);
    if ( result == 0 )
       printf("Both strings are same.
\n");
    else
       printf("Entered strings are not
equal.\n");
     return 0;
}
```