Linguagem C ANSI:
Aula 3 – Strings e Ponteiros

Software Básico, turma A

(Baseado no Curso de Linguagem C da UFMG)

Prof. Marcelo Ladeira – CIC/UnB

Sumário da Aula

- Strings
  - Manipulação de strings
  - Biblioteca string.h
- Ponteiros
  - Ponteiros e Vetores
  - Ponteiros para Ponteiros
  - Cuidados a Serem Tomados ao se Usar Ponteiro

Strings

- **Vetores de caracteres com o último elemento com o conteúdo ‘\0’.**
- **Sintaxe geral**

  char nome_da_string [tamanho];
- **Comentários**
  - **O armazenamento alocado para a string deve incluir espaço para o delimitador ‘\0’.**
  - **A biblioteca *string.h* fornece diversas funções para manipular strings.**

Strings
Função gets – Biblioteca Padrão stdio.h

- **Protótipo**

  char *gets (**char** s[])

- **Comentários**
  - **Lê linha da entrada padrão e a armazena em s.**
  - **Substitui o '\n' por um '\0' no final da linha em s.**
  - **Se encontrar EOF ou erro então retorna NULL, senão retorna s.**
  - **Se até encontrar '\n' houver mais caracteres que a área em s, invade outras áreas, gerando erros imprevistos.**

- **Exemplo (UFMG, pág. 48)**

```
#include <stdio.h>
int main ()
{
char string[100];
printf ("Digite o seu nome: ");
gets (string);
printf ("\n\n Ola %s",string);
return (0);
}
```

Note o uso do nome do vetor como argumento de gets() e para a função printf(). Qual o problema com o uso de gets() ?

Strings
getline: Alternativa Para a Função gets

- **Protótipo**

  int getline(char s[], int lim)
- **Comentários**
  - **Não é padrão ANSI.**
  - **Não ocorre invasão de área.**
  - **Retorna o tamanho da linha lida (<= lim) ou zero se EOF.**
    - **O tamanho da linha inclui o delimitador '\0'.**
    - **Se a linha for vazia, isto é, for apenas um Enter, então tamanho terá valor 1.**
    - **Crtl Z representa o EOF de uma entrada via teclado.**

- **Implementação KR, pág. 60**

```
int getline (char s[], int lim)
{
int c, i;
i = 0;
while (--lim > 0 &&
    (c=getchar()) != EOF && c !=
    '\n') s[i++] = c;
if (c == '\n') s[i++] = c;
s[i] = '\0';
return i;
}
```

Cuidado! No GNU C++, há uma getline com três argumentos!

Strings
fgets: Alternativa Padrão Para getline

- **Protótipo**

  char *fgets (**char** s[], **int** max, FILE *fp)

- **Comentários**
  - **Padrão ANSI – stdio.h**
  - **Lê uma linha de fp (inclui '\n') e a armazena em s.**
    - **Máximo max-1 caracteres.**
    - **Se EOF ou erro, retorna NULL; senão retorna s.**
    - **Não há invasão de área.**
    - **Insere '\0' em s[max-1].**

- **Implementação de getline a partir de fgets**

```
int getline (char *s, int max)
{
if (fgets(s, max, stdin) ==
    NULL)
return 0;
else
return strlen(s);
}
```

Gravando Strings

int fputs (const char *s, FILE *stream)

- Grava a string s em stream.
  - O null caracter de s não é escrito.
  - Não adiciona o caracter newline (não pula de linha), exceto se stream for a saída padrão.
- Retorna um valor não negativo (caracteres gravados)
  - Em caso de erro, retorna EOF.

int puts (const char *s)

- Grava a string s em stdout, seguindo de um newline.
  - O caracter nulo de s não é escrito.
  - Adiciona o caracter newline ao final de s.
- É a mais conveniente para imprimir mensagens simples.
- Exemplo: puts (“isto é uma msg”)

RECOMENDADA !

Strings
Funções da Biblioteca string.h

- **Função strcpy**
  - **Copia strings.**
  - **Protótipo**

    char *strcpy (**char** s[], **const char** ct[]);
  - **Forma de chamada**

    strcpy (string_destino, string_origem);
- **Comentários**
  - **Copia ct em s, incluíndo '\0'. Retorna s.**
    - **Funciona mesmo se a string ct for vazia.**
    - **Não testa estouro de índice.**
      - **É responsabilidade do usuário prover espaço suficiente em s.**
      - **Pode invadir área se ct for maior do que s.**

Strings
Exemplo de Uso da Função strcpy

```
#include <stdio.h>
#include <string.h>
int main ()
{
char str1[100], str2[100], str3[100];
printf ("Entre com uma string: ");
gets (str1);
strcpy (str2,str1);          /* Copia str1 em str2 */
strcpy (str3,"Voce digitou a string ");
                             /* Copia "Voce digitou a string" em str3 */
printf ("\n\n%s%s",str3,str2);
return (0);
}
```


UFMG, pág. 49. Há possibilidade de estouro de subscrito devido ao uso da gets().

Strings
Funções da Biblioteca string.h

- **Função strncpy**
  - **Copia no máximo n caracteres de uma string.**
  - **Protótipo**

    char *strncpy (**char** s[], **const char** ct[], **unsigned int** n);
  - **Forma de chamada**

    strncpy (string_destino, string_origem, n);
- **Comentários**
  - **Copia máximo de n caracteres de ct em s, incluíndo '\0'. Retorna s. Preenche com '\0' se ct for menor que n.**
    - **Funciona mesmo se a string ct for vazia.**
    - **Não testa estouro de índice.**
      - **É responsabilidade do usuário prover espaço suficiente em s.**
      - **Pode invadir área se n for maior do que o tamanho de s.**

Strings
Funções da Biblioteca string.h

- **Função strcat**
  - **Concatena (une) strings.**
  - **Protótipo**

    char *strcat (**char** s[], **const char** ct[]);
  - **Forma de chamada**

    strcat (string_destino, string_origem);
- **Comentários**
  - **Concatena ct no final de s. Inclui o '\0'. Retorna s.**
    - **Funciona mesmo se a string ct for vazia.**
    - **Não testa estouro de índice.**
      - **É responsabilidade do usuário prover espaço suficiente em s.**
      - **Pode invadir área se ct for maior do que o espaço em s.**

Strings
Exemplo de Uso da Função strcat

```
#include <stdio.h>
#include <string.h>
int main ()
{
char str1[100], str2[100];
printf ("Entre com uma string: ");
gets (str1);
strcpy (str2,"Voce digitou a string ");
strcat (str2,str1);
        /* str2 armazenara' Voce digitou a string + o conteudo de str1 */
printf ("\n\n%s",str2);
return (0);
}
```

UFMG, pág. 49. Há possibilidade de estouro de subscrito devido ao uso da gets() e ao tamanho de str2

Strings
Funções da Biblioteca string.h

- **Função strncat**
  - **Concatena no máximo n caracteres a uma string.**
  - **Protótipo**

    char *strncat (**char** s[], **const char** ct[], **unsigned int** n);
  - **Forma de chamada**

    strncat (string_destino, string_origem, n);
- **Comentários**
  - **Concatena máximo de n caracteres de ct em s, incluíndo '\0'. Retorna s.**
    - **Não testa estouro de índice.**
      - **É responsabilidade do usuário prover espaço suficiente em s.**
      - **Pode invadir área se n for maior do que o espaço em s.**

Strings
Funções da Biblioteca Padrão string.h

- **Função strcmp**
  - **Compara duas strings, lexograficamente.**
  - **Protótipo**

    int strcmp (**const char** cs[], **const char** ct[]);
  - **Forma de chamada**

    strcmp (string1, string2);
- **Comentários**
  - **Compara cada caracter de cs com o correspondente de ct.**
    - **Retorna 0 se ∀i, cs[i] == ct[i]. Senão seja k o índice do primeiro diferente. Retorna <0 se cs[k]<ct[k] ou >0 se cs[k]>ct[k].**

Strings
Exemplo de Uso da Função strcmp

```
#include <stdio.h>
#include <string.h>
int main ()
{
char str1[100],str2[100];
printf ("Entre com uma string: ");
gets (str1);
printf ("\n\nEntre com outra string: ");
gets (str2);
if (strcmp(str1,str2))
printf ("\n\nAs duas strings são diferentes.");
else printf ("\n\nAs duas strings são iguais.");
return (0);
}
```


UFMG, pág. 50. Há possibilidade de estouro de subscrito devido ao uso da gets().

Strings
Funções da Biblioteca Padrão string.h

- **Função strncmp**
  - **Compara, lexograficamente, no máximo n caracteres de duas strings.**
  - **Protótipo**

    int strncmp (**const char** cs[], **const char** ct[], **unsigned int** n);
  - **Forma de chamada**

    strncmp (string1, string2, n);
- **Comentários**
  - **Compara até máximo de n caracteres de cs com o correspondente de ct.**
    - **Retorna 0 se ∀i, cs[i] == ct[i]. i=0,n-1. Senão seja k o índice do primeiro diferente. Retorna <0 se cs[k]<ct[k] ou >0 se cs[k]>ct[k].**

Strings
Funções da Biblioteca Padrão string.h

- **Função strlen**
  - **Calcula o comprimento de uma string.**
  - **Protótipo**

    unsigned int strlen (**const char** cs[]);
  - **Forma de chamada**

    strlen (string);
- **Comentários**
  - **Retorna o comprimento de cs.**
    - **O terminador '\0' não é contado.**
      - **De fato o tamanho de cs deve ser um a mais do valor retornado por essa função.**

Strings
Exemplo de Uso da Função strlen

```
#include <string.h>
                                /* reverse: inverte string s no lugar */
void reverse (char s[])
{
    int c, i, j;
    for (i = 0, j = strlen(s)-1; i < j; i++, j--) {
            c = s[i];
            s[i] = s[j];
            s[j] = c;
    }
}
```

Strings
Outras Função de string.h

char *strchr (**const char** cs[], **char** c)
retorna endereço da 1ª ocorrência de c em cs, senão NULL.

char *strrchr(**const char** cs[], **char** c)
retorna endereço da última ocorrência de c em cs, senão NULL.

size_t strspn(**const char** cs[], const **char** ct[])
retorna o tamanho do prefixo de cs consistindo de caracteres em ct.

size_t strcspn(**const char** cs[], const **char** ct[])
retorna o tamanho do prefixo de cs consistindo de caracteres não em ct.

char *strpbrk(**const char** cs[], const **char** ct[])
retorna o endereço da 1ª ocorrência em cs de caracter de ct, senão NULL.

char *strstr(**const char** cs[], const **char** ct[])
returna endereço da 1ª ocorrência da string ct na cs, senão NULL.

char *strerror (**unsigned int** n)
returna endereço da mensagem de erro de número n do sistema.

KR, pág. 207

Expressões
Tabela de Precedência do C

Operadores	Associatividade	Prioridade
() [] -> .	esquerda para a direita	1
! ~ ++ -- - (tipo) * & sizeof	direita para a esquerda	2
* / %	esquerda para a direita	3
+ -	esquerda para a direita	4
<< >>	esquerda para a direita	5
< <= > >=	esquerda para a direita	6
== !=	esquerda para a direita	7
&	esquerda para a direita	8
^	esquerda para a direita	9
|	esquerda para a direita	10
&&	esquerda para a direita	11
||	esquerda para a direita	12
?:	direita para a esquerda	13
= += -= /= *= %= >>= <<= &= ^= |=	direita para a esquerda	14
,	esquerda para a direita	15

Ponteiros

- **Variáveis que contêm endereços de variáveis.**
- **Sintaxe geral**

  *tipo_do_ponteiro * nome_da_variável;*
- **Comentários**
  - **Em C os ponteiros são tipados.**
  - **A declaração de um ponteiro aloca espaço apenas para o ponteiro.**
    - **O ponteiro não fica iniciado.**
      - **Aponta para um endereço indeterminado.**
    - **Precisa ser iniciado antes de poder ser utilizado.**
      - **A iniciação atribui um valor de endereço ao ponteiro.**

Ponteiros

- **Tipos de Ponteiros**
  - char, int, float, double **e** void.
    - **Modificadores de tipo**
      - **short, long, signed e unsigned.**
- **Ponteiro** void
  - **Ponteiro genérico capaz de manipular ponteiros de qualquer um dos outros tipos.**
    - **Não requer uso do operador de cast.**

Ponteiros

- **Obtenção do endereço de uma variável**
  - **Relocável: pelo compilador, durante a compilação**
  - **Real: após a relocação em tempo de execução.**
  - **Operador (referenciação) &:**
    - **Informa ao compilador que se deseja obter o endereço de um objeto.**
    - **Aplicável apenas a objetos na memória**
      - **Variáveis e elementos de matrizes**
      - **Não se aplica a constantes, expressões ou variáveis da classe de armazenamento** register**.**

Ponteiros

- **Operador Indireção ***
  - **Acessa objeto apontado pelo ponteiro.**
    - **Não se aplica a ponteiro do tipo** void
  - **Também é utilizado na definição de ponteiros**

    *tipo_do_ponteiro * nome_da_variável;*
    - **veja** ***nome_da_variável* como mnemônico para a variável de mesmo tipo.**
      - **Por ortogonalidade é usável como a variável.**

- **Exemplo (KR, pág. 78)**

```
int x = 1, y = 2, z[10];
int *ip;          /* ip ponteiro int */
ip = &x;          /* ip aponta x */
y = *ip;          /* y recebe 1 */
*ip = 0;          /* x recebe 0 */
ip = &z[0];       /* ip aponta z[0] */
```

Ponteiros

```c
#include <stdio.h>
int main ()
{
int num,valor, *p;
num=55;
p=&num;              /* Pega o endereco
    de num */
valor=*p;            /* Valor e' igualado a
    num de uma maneira indireta */
printf ("\n\n%d\n",valor);
printf ("Endereco para onde o ponteiro
    aponta: %p\n",p);
printf ("Valor da variavel apontada:
    %d\n",*p);
return (0);
}
```

```c
#include <stdio.h>
int main ()
{
int num,*p;
num=55;
p=&num;              /* Pega o
    endereco de num */
printf ("\nValor inicial: %d\n",num);
*p=100;              /* Muda o valor de
    num de uma maneira indireta */
printf ("\nValor final: %d\n",num);
return (0);
}
```

Note o uso da frase de edição %p.

Ponteiros

- **Aritmética de Ponteiros**
  - Atribuição de ponteiros do mesmo tipo.
    - Apontam para o mesmo endereço.
  - Adicionar (subtrair) um inteiro n de um ponteiro
    - Soma (subtrai) do endereço no ponteiro a quantidade de bytes de n objetos do mesmo tipo do ponteiro.
      - Aponta para o endereço do n-éssimo objeto após (antes)
  - Atribuição ou comparação com zero ou NULL
    - Ponteiro vazio (não definido)
  - Se p e q aponta para elementos de uma matriz
    - Operações relacionais e de igualdade são válidas.
    - Se q>p então a subtração q-p também é válida.

Ponteiros

- **Operações ilegais**
  - **Todas as outras operações com ponteiros são ilegais.**
    - **Exemplos:**
      - **Somar, multiplicar ou dividir**
      - **Executar operações lógicas ou bit-a-bit**
      - **Somar constantes do tipo float ou double**
      - **Atribuir um ponteiro de um tipo a um ponteiro de outro tipo, sem um cast**
        - **exceto para void ***

Exemplos de Operações com Ponteiros

- KR, pág. 86

```
/* strlen: tamanho de s */
int strlen(char *s) {
char *p = s;
while (*p) p++;
return p - s;
}
```

- UFMG, pág. 57

```
p++;
p--;
(*p)++;
p=p+15; ou p+=15;
*(p+15);
elem = p<q? q-p+1: p-q+1;
```

Ponteiros e Vetores

- **Relacionamento Estreito**
  - **Qualquer operação de referenciamento de vetor pode ser feita com ponteiros.**
    - **Em geral a versão com ponteiros é mais rápida.**
    - **É equivalente a uma operação ponteiro + offset**

nome_da_variável[índice]

é equivalente a

**(nome_da_variável+índice)*

Lembre-se que subscritos de matrizes são endereços na forma "base + deslocamento", onde "base" é o endereço do primeiro elemento da matriz.

Ponteiros e Vetores

```
int main ()
{
float matrx [50][50];
int i,j;
for (i=0;i<50;i++)
    for (j=0;j<50;j++)
            matrx[i][j]=0.0;
return (0);
}
```

```
int main ()
{
float matrx [50][50], *p;
int count;
p=matrx[0];
for (count=0;count<2500;count++)
{
    *p=0.0;
    p++;
}
return (0);
}
```

Fazer 2500 incrementos em um ponteiro é muito mais rápido que calcular 2500 deslocamentos completos.

Ponteiros e Vetores

- **Relacionamento Estreito**
  - **Nome do vetor é sinônimo para endereço do primeiro elemento.**
    - ***nome_da_variável* equivale a *&nome_da_variável[0]***
    - **Ponteiro é uma variável. Nome de um vetor não é uma variável.**
      - **É um ponteiro com modificador** const.
        - **Não se pode alterar o endereço do vetor.**
    - **Passagem de vetor como parâmetro atual de uma função**
      - **Passado endereço do primeiro elemento do vetor.**
      - **O parâmetro formal correspondente é uma variável local do tipo ponteiro.**
        - **Alterações no valor dessa variável são locais à função.**

Ponteiros e Vetores

- **UFMG, pág. 60.**

```
int vetor[10];
int *ponteiro, i;
ponteiro = &i;
/* as operações a
   seguir são invalidas
   porque vetor não é
   uma variável */
vetor = vetor + 2;
vetor++;
vetor = ponteiro;
```

- KR, pág 83.

```
strlen("hello, world");     /* string constant */
strlen(array);              /*char array[100]; */
strlen(ptr);                /* char *ptr; */

/* strlen: tamanho de s */
int strlen(char *s)
{
int n;
for (n = 0; *s; s++) n++;
return n;
}
```


Nome do vetor não pode aparecer do lado esquerdo de uma atribuição. Não é um Lvalue.

Ponteiros e Vetores

- **Relacionamento Muito Estrito**
  - **Por outro lado, qualquer operação com ponteiro pode ser feita com referenciamento de vetor.**
    - **Se pa é um ponteiro, pode-se usá-lo com subscrito:**

      pa[i] é identico a *(pa+i).

```
#include <stdio.h>
int main ()
{
int matrx [10] = { 1, 2, 3, 4, 5, 6,
    7, 8, 9, 10 };
int *p;
p=matrx;
printf ("O terceiro elemento do
    vetor e': %d",p[2]);
return (0);
}
```

Vetores de Ponteiros

- **Por ortogonalidade podemos declarar matrizes de ponteiros**
- **Sintaxe Geral**

  *tipo_do_ponteiro *nome_da_variável [tam1] ...[tamN];*
- **Exemplo**

  int *pmatrx [10];

Iniciando Ponteiros

- **Ponteiros para char podem ser iniciados**

  char *str1= "String constante.";

  - **O compilador C substitui todas as ocorrências de strings do usuário por um ponteiro.**
    - **As strings são consideradas constantes strings.**
    - **Elas são armazenadas em um segmento de constantes**
      - **Se alterarmos *str1 podemos corromper o banco de strings criado pelo compilador.**
      - **Se alterarmos str1 podemos perder a referência para a string.**
        - **Essa declaração só deve ser feita se as strings forem muito duplicadas.**

Iniciando Ponteiros

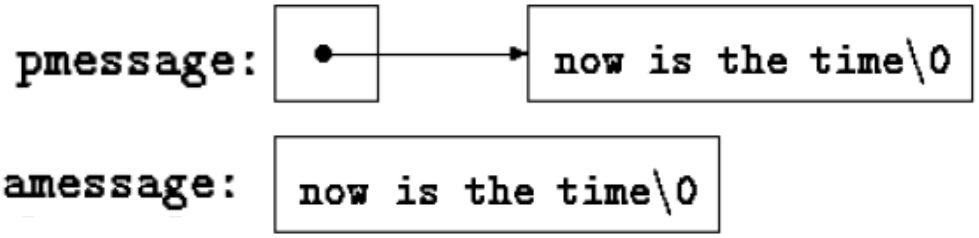
```
/* mes_nome: retorna nome do n-éssimo mês */
char *mes_nome(int n)
{
static char *nome[] = {
“Mês ilegal",
"Janeiro", "Fevereiro", "Março",
"Abril", "Maio", "Junho",
"Julho", "Agosto", "Setembro",
"Outubro", "Novembro", "Dezembro"
};
return (n < 1 || n > 12) ? nome[0] : nome[n];
}
```

Qual a razão de nome ser da classe de armazenamento static ?

Iniciando Ponteiros

- **As declarações abaixo são diferentes**

```
char amessage[] = "now is the time"; /* an array */
char *pmessage = "now is the time"; /* a pointer */
```



- **amessage aponta sempre para o mesmo local.**

```
amessage[0] = 'N';
```

- **pmessage**

```
*pmessage = 'N';      /* resultado pode ser imprevisível */
```

Exemplos de Ponteiros Para Caracter

- KR, pág. 88

```
/* strcpy: copia t para s */
void strcpy (char *s, char
    *t)
{
while (*s++ = *t++) ;
}
```

- **KR, pág 89**

```
/* strcmp: retorna <0 se s<t, 0
    se s==t, >0 se s>t */
int strcmp (char *s, char *t)
{
for ( ; *s == *t; s++, t++)
    if (!*s) return 0;
return *s - *t;
}
```

Ponteiros Para Ponteiros

- **É o endereço de um endereço ...**
- **Sintaxe Geral**

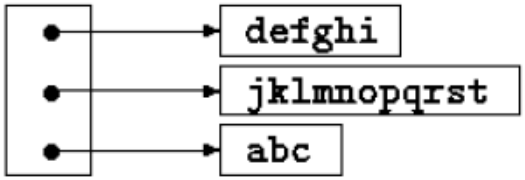
  *tipo_da_variável **nome_da_variável;*
- **Comentários**
  - **Por ortogonalidade podemos declarar ponteiros para ponteiros para ... com mais asteriscos.**

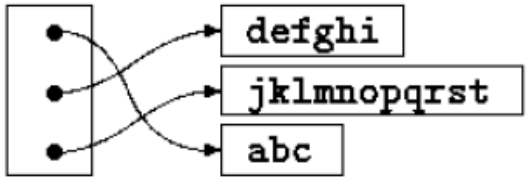
    ****nome_da_variável é o conteúdo do objeto apontado.**

    ***nome_da_variável é o conteúdo do ponteiro intermediário.**
  - **Uma aplicação interessante é um ponteiro para strings.**

Ponteiros Para Ponteiros

```
#include <stdio.h>
int main() {
float fpi = 3.1415, *pf, **ppf;
pf = &fpi;              /* pf armazena o endereco de fpi */
ppf = &pf;              /* ppf armazena o endereco de pf */
printf("%f", **ppf);    /* Imprime o valor de fpi */
printf("%f", *pf);      /* Tambem imprime o valor de fpi */
return (0);
}
```





Ponteiros para strings são úteis para ordenação de strings.

Cuidados ao Serem Tomados ao se Usar Ponteiros

- **Nunca use ponteiro não iniciado**
  - **Pode provoca resultados inesperados.**
- **Saiba para onde o ponteiro aponta**

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

Ponteiros vs Vetores Multidimensionais

int a[10][20];

int *b[10];

- **a[3][4] e b[3][4] são referências legais a um** int**.**
- **a é um vetor de fato: aloca 200 inteiros.**
  - A expressão 20**row* +*col* é usada para localizar a[row][col]
- **A declaração de b aloca apenas 10 ponteiros**
  - **Devem ser iniciados estaticamente ou por código.**
  - **Se os elementos de b apontam para vetores de inteiros, esses vetores não precisam ser do mesmo tamanho.**
    - **Se de 20 elementos teremos 10*20 + 10 objetos alocados.**

Argumentos de Linha de Comandos

- **Passados via a função main**

  int main (**int** argc, **char** ** argv)

- **Comentários**
  - **argc é o número de argumentos de linha de comando com que o programa foi invocado.**
    - **Sempre maior ou igual a 1.**
  - **argv é um ponteiro para um vetor de strings que contem os argumentos**
    - **Um argumento por string.**
    - **argv[0] aponta para o nome do programa invocado.**

Argumentos de Linha de Comando

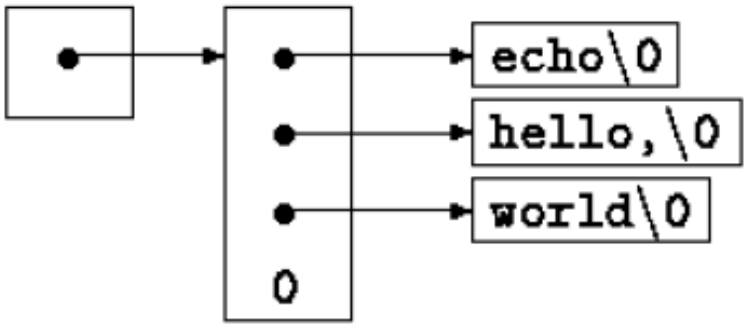
- Seja a chamada para executar o programa echo com argumento “hello, world”

  echo hello, world

- Os valores dos argumentos são:

  argc = 3

  e argv:



Note que argv finaliza com NULL.

Argumentos de Linha de Comando
Versões de echo

- **Ponteiros de strings**

```
#include <stdio.h>
main (int argc, char *argv[])
{
int i;
for (i = 1; i < argc; i++)
printf("%s%s", argv[i], (i < argc-
    1) ? " " : "");
printf("\n");
return 0;
}
```

- **Ponteiro de Ponteiro**

```
#include <stdio.h>
main (int argc, char **argv)
{
while (--argc > 0)
printf("%s%s", *++argv, (argc >
    1) ? " " : "");
printf("\n");
return 0;
}
```