FACAMP
FACULDADES DE CAMPINAS

Estruturas de Dados

Aula 2 – Ponteiros; passagem por referência; cadeias de caracteres e suas funções; entrada e saída formatadas

1º semestre de 2020
Prof. José Martins Jr

Ponteiro

- Tipo de variável especial que **armazena o endereço** de um campo de dados, cujo tipo é declarado
- É declarado com a prefixação do caractere * antes do nome da variável (que pode ser um tipo básico ou estruturado)
  - Em C, toda **variável** do tipo arranjo (**array**) **é** também um **ponteiro**
- Exemplo
  - Declaração de uma variável do tipo ponteiro para um campo inteiro

```
int *p;
```

  - O endereço é armazenado na variável e pode ser recuperado, obtendo-se o valor de p

```
printf("Endereço para onde p aponta: %d\n", p);
```

Ponteiro

- O operador **&** antes do nome de uma variável fornece o seu endereço na memória
  - Sendo assim:

```
printf("Endereço de p: %d\n", &p);
```

- O operador * antes de um ponteiro indica o valor armazenado no endereço que aponta
  - Então

```
int i = 10;
p = &i;
printf("Valor do campo apontado por p: %d\n", *p);
//exibirá o valor 10
```

Exemplos de operações com ponteiros

```
int x = 1, y = 2, z[10] = {1,2,3,4,5,6,7,8,9,10};
int *ip;                        // ip é um ponteiro para int
ip = &x;                        // ip agora aponta para x
y = *ip;                        // y agora recebe o valor 1
*ip = 0;                        // x agora recebe o valor 0
ip = z;                         // ip agora aponta para z[0]
ip++;                           // ip agora aponta para z[1]
printf("%d\n", *(++ip));        //exibe valor do elemento z[2]
printf("%u\n", &y);             //exibe endereço da variável y
printf("%d\n", *(&y));          //exibe conteúdo do endereço de y
printf("%u\n", ip);             //exibe conteúdo do ponteiro ip
printf("%u\n", &z[2]);          //exibe endereço de z[2], atual ip
printf("%u\n", *ip);            //exibe conteúdo do endereço em ip
printf("%u\n", z[2]);           //exibe valor guardado em z[2]
```

Obs.: analise os outros exemplos fornecidos

Passagem por referência

- Ponteiros são especialmente úteis na passagem de argumentos por referência entre funções
  - Permite passar a referência de uma variável local de uma função, cujo endereço pode ser acessado pela função chamada
  - Evita-se o uso de variáveis globais

- Exemplo (implementar com o professor)
  - Função que troca os valores de duas variáveis na memória

```
void troca(int *px, int *py);
```

  - Pode ser chamada no procedimento de origem na forma

```
troca(&x, &y);
// x e y são as variáveis inteiras em questão
```

Cadeias de caracteres

- Uma string em C é representada como uma sequência de caracteres na memória, **terminada pelo caractere '\0'**
- Pode ser declarada
  - Como um ponteiro para caractere (referência para o primeiro caractere da string armazenada na memória)

```
char *c = "Hello World";
//armazena a constante “Hello World” na memória
//atribui o endereço do 1° caractere a c
```

  - Como um array de caracteres (local para armazenar uma string)

```
char s[100];
strcpy(s, c); //veremos adiante
printf("%s\n", c);
printf("%s\n", s);
//imprimirá todos os caracteres até '\0'
```

Manipulação de strings

`char *strcat(char *s, const char *ct)`

- Concatena a string ct no final da string s e retorna s

`char *strncat(char *s, const char *ct, size_t n)`

- Concatena n caracteres (no máximo) da string ct no final da string s, terminando-a com '\0', e retorna s

`char *strcpy(char *s, const char *ct)`

- Copia a string ct para a string s, incluindo '\0' e retorna s

`char *strncpy(char *s, const char *ct, size_t n)`

- Copia n caracteres (no máximo) da string ct para a string s, e retorna s
- Completa com '\0's no final se ct tiver menos que n caracteres

Obs.: o tipo size_t é um unsigned int (retornado por sizeof)

Manipulação de strings

`int strcmp(const char *cs, const char *ct)`

- Compara a string cs à string ct, e retorna <0 se cs<ct, 0 se cs==ct, ou >0 se cs>ct

`int strncmp(const char *cs, const char *ct, size_t n)`

- Compara n caracteres (no máximo) da string cs à string ct; e retorna <0 se cs<ct, 0 se cs==ct, ou >0 se cs>ct

`size_t strlen(const char *cs)`

- Retorna o comprimento da string cs

`char *strstr(const char *cs, const char *ct)`

- Retorna um ponteiro para a primeira ocorrência da string ct na string cs, ou NULL, se não estiver presente

`char *strtok(char *s, const char *ct)`

- Procura em s por tokens delimitados pelos caracteres em ct

Obs.: analise os outros exemplos fornecidos

Saída formatada

```
int printf(const char *format, ...)
int fprintf(FILE *stream, const char *format, ...)
int sprintf(char *str, const char *format, ...)
```

- O argumento format refere-se a uma cadeia de caracteres que será impressa no dispositivo de saída (terminal, arquivo ou string)
- Nessa cadeia encontram-se caracteres normais, sentenças de formatação (conversão) e caracteres especiais (códigos de escape)
- Os códigos de conversão devem ser dispostos na cadeia em igual número e ordem aos respectivos argumentos subsequentes da função
- Ex.: `printf("Resultado: %d\n", res);`
  - A cadeia format é composta pela sentença “Resultado: %d\n”
  - %d é uma sentença de formatação (do argumento res) decimal
  - \n é um código de escape que significa nova linha

Formatos para saída

d,i	int; decimal number
o	int; unsigned octal number (without a leading zero)
x,X	int; unsigned hexadecimal number (without a leading 0x or 0X), using abcdef or ABCDEF for 10, ...,15.
u	int; unsigned decimal number
c	int; single character
s	char *; print characters from the string until a '\0' or the number of characters given by the precision.
f	double; [-]m.dddddd, where the number of d's is given by the precision (default 6).
e,E	double; [-]m.ddddddе+/-xx or [-]m.ddddddE+/-xx, where the number of d's is given by the precision (default 6).
g,G	double; use %e or %E if the exponent is less than -4 or greater than or equal to the precision; otherwise use %f. Trailing zeros and a trailing decimal point are not printed.
p	void *; pointer (implementation-dependent representation).
%	no argument is converted; print a %

Sequência da conversão

- Outras informações sobre o formato podem ser colocadas entre o % e o código do formato
  - Um sinal de subtração indica alinhamento do conteúdo do campo à esquerda
  - Um número, que especifica o tamanho mínimo do campo
  - Um ponto, que separa o tamanho do campo de sua precisão
  - Um número, que descreve a precisão
    - Número máximo de caracteres a serem impressos de uma string
    - Ou o número de casas decimais de um valor com ponto flutuante
    - Ou o número mínimo de dígitos de um inteiro
  - Um h para imprimir um inteiro como short, ou um l para o formato long

Obs.: analise os outros exemplos fornecidos

Códigos de escape

newline	NL (LF)	`\n`
horizontal tab	HT	`\t`
vertical tab	VT	`\v`
backspace	BS	`\b`
carriage return	CR	`\r`
formfeed	FF	`\f`
audible alert	BEL	`\a`

backslash	\	`\\`
question mark	?	`\?`
single quote	'	`\'`
double quote	"	`\"`
octal number	ooo	`\ooo`
hex number	hh	`\xhh`

Entrada formatada

```
int scanf(const char *format, ...)
int fscanf(FILE *stream, const char *format, ...)
int sscanf(const char *str, const char *format, ...)
```

- A cadeia format deve conter sentenças de formatação (com códigos de conversão), em número igual ao de argumentos a serem lidos
- Ao contrário do printf, cada argumento subsequente deve indicar o endereço da variável que receberá o valor
- Aplicam-se também muitas das regras de sequência de conversão, usadas para saída
- Exs.:

```
int x;
char frase[100];
scanf("%d", &x);
scanf("%s", frase);
```

Formatos para entrada

d	decimal integer; int *
i	integer; int *. The integer may be in octal (leading 0) or hexadecimal (leading 0x or 0X).
o	octal integer (with or without leading zero); int *
u	unsigned decimal integer; unsigned int *
x	hexadecimal integer (with or without leading 0x or 0X); int *
c	characters; char *. The next input characters (default 1) are placed at the indicated spot. The normal skip-over white space is suppressed; to read the next non-white space character, use %1s
s	character string (not quoted); char *, pointing to an array of characters long enough for the string and a terminating '\0' that will be added.
e,f,g	floating-point number with optional sign, optional decimal point and optional exponent; float *
%	literal %; no assignment is made.

Lendo linhas

- Problema com o scanf
  - Apresenta problemas para leitura de linhas com espaços
  - Lê a primeira string (primeiro argumento), antes do primeiro espaço
  - O restante é atribuído em sequência às próximas leituras!

- Uma possível solução

```
char *gets(char *frase)
```

  - Lê da entrada padrão (teclado), todos os caracteres fornecidos, até encontrar um enter (ou \n, que é descartado)
  - Use com cuidado, pois pode apresentar problema de segurança

Bibliografia

- KERNIGHAN, B. W., RITCHIE, D. M. C, a linguagem de programação: padrão ANSI. Rio de Janeiro: Campus, 1989. 289 p.
- LOOSEMORE, S.; STALLMAN, R. M. et al. The GNU C Library Reference Manual. Disponível no endereço: http://www.gnu.org/software/libc/manual/