Estruturas de Dados
Aula 6: Cadeias de Caracteres

28/03/2010

Caracteres

- Caracteres são representados internamente por códigos numéricos
- Tipo char (inteiro “pequeno”)
  - 1 byte (8 bits)
  - 256 caracteres possíveis
- Códigos são associados em uma tabela de códigos
  - Por exemplo, ASCII

Tabela ASCII

	0	1	2	3	4	5	6	7	8	9
30			sp	!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~			

Tabela ASCII de caracteres de controle

0	`nul`	*null*: nulo
7	`bel`	*bell*: campainha
8	`bs`	*backspace*: volta e apaga um caractere
9	`ht`	*tab*: tabulação horizontal
10	`nl`	*newline* ou *line feed*: muda de linha
13	`cr`	*carriage return*: volta ao início da linha
127	`del`	*delete*: apaga um caractere

Caracteres em C

```
char c = 97;
printf ("%d %c\n", c, c);
```

- printf imprime o conteúdo de c em dois formatos diferentes:
  - %d : imprime o valor do código numérico (97)
  - %c : imprime o caracter associado ao código na tabela ('a')

Caracteres em C

- Devemos evitar o uso explícito dos códigos
  - Muda de acordo com a arquitetura usada
  - Em C, usamos "constantes de caractere"
- Constantes de caractere
  - Caractere com aspas simples

```
char c = 'a';
printf ("%d %c\n", c, c);
```

Caracteres em C

- Os dígitos são codificados sequencialmente na tabela ASCII
  - ‘0’ (48), ‘1’ (49), etc...
- O código a seguir verifica se um caracter é um dígito (de 0 a 9)

```
int digito (char c)
{
    if ((c>=‘0’) && (c<=‘9’))
          return 1;
    else
          return 0;
}
```

Caracteres em C

- Para converter de letra minúscula para maiúscula (usando codificação sequencial)

```
char maiuscula (char c)
{
    if (c>='a' && c<='z')
        c = c – 'a' + 'A';
     return c;
 }
```

Cadeias de Caracteres (strings)

- Representação de cadeias de caracteres
  - Vetor do tipo char, terminando com o caractere nulo '\0'
- printf com especificador %s
  - Imprime caracteres de um vetor até encontrar o '\0'

```
int main (void)
{  char str[4];
   str[0] = 'O';
   str[1] = 'l';
   str[2] = 'a';
   str[3] = '\0';
   printf ("%s \n", str);
}
```

Cadeias de Caracteres

```
int main (void)
{
    char str [] = “Ola”;
    printf (“%s \n”, str);
}

int main (void)
{
    char str [] = {‘O’, ‘l’, ‘a’, ‘\0’};
    printf (“%s \n”, str);
}
```

Leitura de caracteres e cadeias de caracteres

- Usando scanf
  - Com especificação do formato

```
char a;
...
scanf (“%c”, &a);
...
```

Leitura de caracteres e cadeias de caracteres

- Para pular caracteres em branco

```
char a;
...
scanf (“ %c”, &a);
...
```

Leitura de caracteres e cadeias de caracteres

- %s pula caracteres em branco
  - Cidades com nomes duplos não seriam capturadas corretamente

```
char str[81];
...
scanf (“%s”, str);
...
```

Leitura de caracteres e cadeias de caracteres

- Para capturar nomes compostos:
  - %[...]
  - Entre colchetes especificamos os caracteres que serão aceitos na leitura
  - Se [^...], tem-se o efeito inverso, i.e., os elementos especificados não são lidos
  - Espaço antes do % garante que espaços em branco antes do nome sejam descartados

```
char str[81];
...
scanf (“ %[^\n]”, str);
...
```

Leitura de caracteres e cadeias de caracteres

- Para garantir que o tamanho é suficiente
  - Especifique o número máximo de caracteres lidos

```
char str[81];
...
scanf (“ %80[^\n]”, str);
...
```

Exemplos de funções

- Função que calcula comprimento de uma cadeia de caracteres

```
int comprimento (char* s)
{
    int i;
    int n=0;
    for (i=0; s[i] != '\0'; i++)
        n++;
    return n;
}
```

- Função análoga da biblioteca padrão (string.h): strlen

Exemplos de funções

- Função que copia os elementos de uma cadeia de caracteres para outra

```
void copia (char* destino, char* origem)
{
    int i;
    for (i=0; origem[i] != '\0'; i++)
        destino[i] = origem[i];
    destino[i] = '\0';
}
```

- Função análoga da biblioteca padrão (string.h): strcpy

Exemplos de funções

- Função que concatena uma cadeia de caracteres a outra

```
void concatena (char* dest, char* orig)
{
   int i = 0; // índice destino
   int j; // índice origem
   while (dest[i] != '\0') // acha o final da cadeia destino
      i++;
   for (j=0; orig[j] != '\0'; j++)
   {
       dest[i] = orig[j];
       i++;
    }
   dest[i] = '\0';
}
```

- Função análoga da biblioteca padrão (string.h): strcat

Exemplos de funções

- Função que compara duas cadeias de caracteres, caractere a caractere
  - Usamos os códigos dos caracteres para determinar precedência
  - Retorno:
    - Se s1 preceder s2 -> -1
    - Se s2 preceder s1 -> 1

- Função análoga da biblioteca padrão (string.h): strcmp

Exemplos de funções

```
int compara (char* s1, char* s2)
{
    int i;
    for (i=0; s1[i] != '\0' && s2[i] != '\0'; i++) {
       if (s1[i] < s2[i])
           return -1;
       else (s1[i] > s2[i])
           return 1;
    }
    if (s1[i] == s2[i]) //strings iguais
         return 0;
    else if (s2[i] != '\0') //s1 é menor, pois tem menos caracteres
         return -1;
    else
         return 1; // s2 é menor, pois tem menos caracteres
}
```

Exemplos de funções

- Exemplo com alocação dinâmica

```
#include <stdlib.h>
#include <string.h>

char* duplica (char* s)
{
   int n = strlen(s);
   char* d = (char*) malloc ((n+1)*sizeof(char));
   strcpy (d, s);
   return d;
}
```

Vetor de cadeias de caracteres

- Conjunto bidimensional do tipo char
- Por exemplo:
  - Número máximo de alunos em uma turma é 50
  - Cada nome terá no máximo 80 caracteres
  - char alunos [50][81]
  - alunos[i] acessa (i+1)-ésimo aluno da turma
  - alunos[i][j] acessa (j+1) –ésima letra do nome do (i+1)-ésimo aluno

```
void imprime (int n, char alunos[][81])
{
   int i;
   for (i=0; i<n; i++)
       printf ("%s\n", alunos[i]);
}
```

Vetor de cadeias de caracteres

- Com alocação dinâmica
  - Declara-se um vetor de ponteiros
  - Aloca-se dinamicamente cada elemento

```
#define MAX 50
char* alunos[MAX];

char* lelinha (void)
{
   char linha [121];
   printf (“Digite um nome: “);
   scanf (“ %120[^\n]”, linha);
   return duplica (linha);
}
```

Vetor de cadeias de caracteres

```
int lenomes (char** alunos)
{
    int i;
    int n;
    do {
        printf (“Digite o numero de alunos: “);
        scanf (“%d”, &n);
    } while (n>MAX);

    for (i=0; i<n; i++)
         alunos[i] = lelinha();
    return n;
}
```

Vetor de cadeias de caracteres

- Função para liberar os nomes alocados

```
void liberanomes (int n, char** alunos)
{
    int i;
    for (i=0; i<n; i++)
        free(alunos[i]);
}
```

- Função que imprime os nomes dos alunos

```
void imprimenomes (int n, char** alunos)
{
    int i;
    for (i=0; i<n; i++)
        printf ("%s\n", alunos[i];
}
```

Vetor de cadeias de caracteres

- Programa que faz uso das funções anteriores

```
#define MAX 50

int main (void)
{
    char* alunos[MAX];
    int n = lenomes(alunos);
    imprimenomes(n, alunos);
    liberanomes(n, alunos);
    return 0;
}
```

Parâmetros da função main

- Função main
  - Zero parâmetros
  - Dois parâmetros (argc e argv)
- argc
  - Número de argumentos passados para o main
- argv
  - Vetor de cadeias de caracteres com os nomes passados como argumento

```
int main (int argc, char** argv)
{
    ...
}
```

Parâmetros da função main

- Considere um programa executável *mensagem*

  > mensagem estruturas de dados

- argc receberá o valor 4
- argv será inicializado com
  - argv[0] = “mensagem”
  - argv[1] = “estruturas”
  - argv[2] = “de”
  - argv[2] = “dados”

Parâmetros da função main

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

- Qual será a saída desse programa?