Fundamentos de Programação

Filipe Francisco

03 de maio de 2018

Problemas

- Alguns dos problemas desta semana trabalham com caracteres, outros com strings
- Daí, surge a dúvida: quando ler caractere e quando ler string?
  - caractere: uma única letra, dígito, símbolo, ...
  - nos problemas, normalmente representa uma letra ou uma operação a ser feita sobre outros dados lidos
    - maiúsculo, minúsculo ou dígito, e cifra: letra
    - calculadora, e meu word quebrou: operação
  - string: uma sequência de uma ou mais caracteres
  - será uma palavra ou frase sobre a qual trabalharemos

Problemas

- Valdiskey e a cifra V1!
  - comentamos que em C, um caractere é basicamente um byte (8 bits)
  - também comentamos que a linguagem C transforma de char para int (e vice versa) implicitamente
    - para isso, basta transformar os 8 bits em um valor decimal!
  - portanto, se temos um caractere c quisermos andar uma certa quantidade n de casas, basta fazermos $c = c + n$
  - problemas: extrapolar (ficar antes do 'a', ou depois do 'z')
    - porém, é um caso fácil de resolver!
- Gritando em Caixa Alta - Invertendo o Case da frase!
  - neste problema, podemos usar alguns métodos que comentamos
    - isupper, islower, toupper, tolower
  - porém, os métodos não funcionam se aplicarmos sobre a string *str*
    - estes métodos trabalham sobre um único caractere!
  - para isto, basta aplicarmos a análise caractere a caractere, ou seja, para cada elemento do vetor

Problemas

- Substring na decoração - Obter Substrings!
  - recebemos uma string, uma posição inicial *a* e uma quantidade *n*
  - devemos imprimir *n* caracteres a partir da posição *a*
  - problema: e se a string não tiver tantos caracteres assim?
    - caso 1: *a* é maior que a quantidade de caracteres da string
    - caso 2: imprimir *n* caracteres a partir de *a*, sendo que partindo de *a* nós temos menos de *n* caracteres
  - tratando estes dois casos, resolvemos o problema!
  - dica: utilize a função strlen(str)
- MeU WoRd QuEbRoU - Formatação de Case!
  - lemos uma string e um caractere, e aplicamos uma certa transformação
    - o tipo de transformação é determinado pelo caractere
  - assim como no problema "Gritando em Caixa Alta", podemos usar as funções para trabalhar com caracteres!
    - obs: também devemos trabalhar caractere a caractere
  - dica: uma palavra só tem uma letra se há um espaço antes e depois dela (exceção: primeiro caractere da frase)

Comentários

- Vimos três maneiras um pouco ruins para ler uma string:

  scanf("%s",&str);
  - problema: para ao encontrar um espaço, ou seja, só lê uma palavra
  - ex: ao digitar "oi cara de boi", str="oi"

  gets(str);
  - problema: não tem controle da memória
  - ex: se str tem tamanho 10 e digitamos 100 caracteres, ele expande
  - Moodle: the 'gets' function is dangerous and should not be used.

  fgets(str,sizeof(str),stdin);
  - problema: armazena o enter ('\n') antes do '\0'
  - ex: ao digitar "abc", str="abc\n"
  - ou seja, se quisermos utilizar o fgets, temos que trocar o '\n' por '\0'

Comentários

- Portanto, utilizaremos a versão a seguir

  scanf("%100[^\n]",str);
  - lê até 100 caracteres, até ler um '\n'
  - por que usar esta versão?
    - tem controle da memória
    - lê espaços (ou seja, lê frases - só para com enter)
    - não guarda '\n'
  - podemos ter problemas ao ler duas strings, ou uma string e um caractere
    - ao lermos uma string desta maneira, guardamos todos os caracteres, menos o '\n'
    - como não guardamos o '\n', ele vai pro buffer
    - com um '\n' no buffer, o caractere/string seguinte não é lido!

Comentários

- Felizmente, este caso é simples de se resolver

  scanf("%100[^\n]\n",str);
  - lê até 100 caracteres, até ler um '\n'
    - importante: grava até encontrar um '\n', mas captura o '\n', ou seja, não o joga pro buffer!
  - obs: não colocamos este '\n' para a leitura da última string
    - se fizer isso, o compilador espera que você insira algo mais
- Para ler duas strings:

  scanf("%100[^\n]\n",str1);
  scanf("%100[^\n]",str2);
- Para ler uma string e um caractere:

  scanf("%100[^\n]\n",str);
  scanf("%c",letra);

Comentários

- Problema: leitura de strings em um loop
  - pelo que acabamos de mostrar, não podemos botar o '\n' adicional no último scanf
  - sendo assim, não podemos botar um scanf com '\n' em um loop
    - e vimos que não fazendo isso, o '\n' vai pro buffer
- Sendo assim, como fazer para ler várias strings em um loop?
- Obs: nos exercícios do Moodle, não precisaremos fazer isto, mas é bom comentar!

- Para este caso em específico, utilizaremos o fgets!
  - o problema do fgets é armazenar o '\n' antes do '\0'
  - se resolvermos este problema, podemos utilizá-lo tranquilamente!
- Como fazer?

Comentários

- Ideia: substituir o '\n' por '\0'
- Maneira 1: utilizando a função strchr:

```
char *pos;
pos=strchr(str,'\n');
*pos='\0';
```

  - ideia: utilizar strchr para procurar pelo caractere '\n' na string
  - a função retorna um ponteiro para a posição onde está o '\n'
    - ponteiro: diz a posição na memória onde está a informação
  - a partir disto, vamos nesta posição e alteramos para '\0'
- Maneira 2: utilizando a função strlen:

```
int tam;
tam=strlen(str);
str[tam-1]='\0';
```

  - ideia: utilizar strlen para saber o comprimento da string
  - sabemos que o '\n' será o último caractere não-nulo da string
    - última posição: posição $tam - 1$
  - a partir disto, vamos nesta posição e alteramos para '\0'

Comentários

- Uma pergunta que pode surgir a partir dos últimos comentários é: quando isso é útil?
- Para responder, vamos apresentar um exemplo de problema no qual podemos aplicar esta ideia:
  - você recebe várias strings e deve contar quantas destas strings tem pelo menos uma letra 'a'
- Mesmo que não armazenemos todas as strings, podemos trabalhar com elas assim que lemos!
  - afinal, basta verificar se a string lida tem algum 'a'
- Pergunta: e se quiséssemos ler e armazenar todas as strings para só depois trabalharmos com elas?

Matriz

- Relembrando: uma string é um vetor de caracteres
- Se queremos armazenar várias strings, podemos ter uma ideia de criarmos um vetor de strings
- Porém, a string em si já é um vetor, portanto de fato estaríamos criando um vetor de vetores de caracteres
- Esta é a ideia de uma matriz!

- Vimos que um vetor é basicamente uma sequência de elementos
- Algo que não comentamos: esta sequência é estritamente unidimensional
  - só "cresce" em uma dimensão, só precisamos de um índice
- As matrizes generalizam essa ideia, permitindo que trabalhemos com quantas dimensões quisermos

Matriz

- Algumas aplicações de matrizes:
  - arquivos de imagem (bitmap) e tela de um computador
    - ex: a resolução 1920x1080 representa uma matriz de pixels de 1080 linhas por 1920 colunas
  - transformações geométricas na computação gráfica
    - ex: escala, rotações, translações
    - basicamente, cada uma destas operações é realizada por uma multiplicação de matrizes
  - programas de edição de imagem
    - em processamento de imagens, alguns filtros que aplicamos sobre uma foto se utilizam de uma matriz
  - representação de tabelas e dados
    - em grande maior parte dos casos, tabelas são estruturas bidimensionais como matrizes, com linhas e colunas
    - algoritmos que trabalham sobre um conjunto de dados na verdade trabalham sobre uma matriz com estes dados

Matriz

- Matrizes são bem parecidas com vetores, não devemos ter dificuldade
  - para trabalhar com vetores, tinhamos um loop para fazer cada coisa: ler, trabalhar, imprimir
  - aqui também teremos loops separados, mas desta vez teremos loops aninhados!
    - loops aninhados: um loop dentro de outro
    - obs: já tivemos exemplos que usaram isto!
- Matrizes também podem ajudar em problemas de vetores!
  - em alguns problemas, em vez de criarmos dois vetores, podemos criar uma matriz de duas linhas e armazenar um vetor em cada linha!
- Novamente, analisaremos os pontos essenciais para se trabalhar com esta estrutura:
  - como criar uma matriz?
  - como preencher uma matriz?
  - como trabalhar/acessar uma matriz?
  - como imprimir (os elementos de) uma matriz?

Matriz

- Como criar uma matriz?
  - para criar um vetor, precisávamos de três coisas:
    - tipo
    - identificador
    - tamanho do vetor
  - para criar uma matriz, como podemos ter mais de uma dimensão, precisamos definir uma informação a mais
  - portanto, para criar uma matriz, precisamos de quatro informações:
    - tipo
    - identificador
    - quantidade de dimensões
    - tamanho de cada uma das dimensões do vetor
  - por exemplo, criar uma matriz de dimensões 2×3 é diferente de criar uma matriz com dimensões 3×4

Matriz

- Como criar uma matriz? (continuação)
  - pseudocódigo:

    inteiro M1[1..m,1..n]
    - exemplo 1: criação de uma matriz $m \times n$ de inteiros

    caractere M2[1..5,1..100]
    - exemplo 2: criação de uma matriz 5×100 de caracteres
    - obs: como vimos, esta é a ideia de um vetor de strings!
  - C:

    int M1[m][n];
    - exemplo 1: criação de uma matriz $m \times n$ de inteiros

    char M2[5][100];
    - exemplo 2: criação de uma matriz 5×100 de caracteres
    - obs: também podemos criar M3[m][100] ou M4[5][n] (ou seja, usando apenas uma variável)

Matriz

- Como preencher uma matriz?
  - para isto, suponha que temos uma matriz M de inteiros com dimensões $m \times n$ (m linhas e n colunas)
  - para preencher todas as posições, precisamos de dois loops!
  - pseudocódigo:

```
para i de 0 a m − 1 passo 1 faça
    para j de 0 a n − 1 passo 1 faça
        leia M[i,j]
```

  - C:

```
for(i=0;i<m;i++) {
    for(j=0;j<n;i++)
        scanf("%d",&M[i][j]);
    }
}
```

Matriz

- Como preencher uma matriz? (continuação)
  - bônus: se temos uma matriz de caracteres, podemos ler cada linha desta matriz como uma string!
  - para isto, suponha uma matriz S de caracteres com dimensões $m \times 100$
  - pseudocódigo:

    para i de 0 a $m-1$ passo 1 faça
        leia M[i]

  - C:

```
char *pos;
for(i=0;i<m;i++) {
    fgets(S[i],sizeof(S[i]),stdin);
    pos=strchr(S[i],'\n');                    //ou tam=strlen(S[i]);
    *pos='\0';                                  //ou S[i][tam]='\0';
}
```

Matriz

- Como preencher uma matriz? (continuação)
  - bônus 2: também podemos inicializar uma matriz no momento que a criamos
  - em C, podemos fazer:

    int M[2][3]={1,2,3,4,5,6};
  - este tipo de inicialização preenche a matriz linha a linha
    - quando preenchemos uma linha completa, pulamos para a próxima
  - fazendo "M[2][3]={1,2,3,4,5,6}", a matriz resultante será:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

  - fazendo "M[3][2]={1,2,3,4,5,6}", a matriz resultante será:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

Matriz

- Como trabalhar/acessar uma matriz?
  - relembrando: para acessar uma posição no vetor, precisamos apenas do índice referente à posição
  - de modo semelhante, para acessar uma posição de uma matriz, precisamos apenas dos índices referentes a cada uma das dimensões
    - assim como no vetor, os índices de uma dimensão começam com 0
  - nos exemplos abaixo, estamos somando duas matrizes A e B em uma matriz C
  - pseudocódigo:

```
para i de 0 a m − 1 passo 1 faça
    para j de 0 a n − 1 passo 1 faça
        C[i,j]=A[i,j]+B[i,j]
```

  - C:

```
for(i=0;i<m;i++) {
    for(j=0;j<n;i++)
        C[i][j]=A[i][j]+B[i][j];
    }
}
```

Matriz

- Como imprimir (os elementos de) uma matriz?
  - para imprimir uma matriz, precisamos imprimir todas as posições, portanto também precisamos de dois loops!
  - para isto, suponha que temos uma matriz M de inteiros com dimensões $m \times n$ (m linhas e n colunas)
  - pseudocódigo:

```
para i de 0 a m − 1 passo 1 faça
    para j de 0 a n − 1 passo 1 faça
        imprima M[i,j]
```

  - C:

```
for(i=0;i<m;i++) {
    for(j=0;j<n;j++)
        printf("%d ",M[i][j]);
    }
    printf("\n");
}
```

Matriz

- Como imprimir (os elementos de) uma matriz? (continuação)
  - bônus: se nossa matriz é utilizada para armazenar strings (uma em cada linha), podemos imprimir cada linha como uma string!
  - para isto, suponha uma matriz S de caracteres com dimensões $m \times 100$
  - pseudocódigo:

    para i de 0 a $m - 1$ passo 1 faça
        imprima M[i]

  - C:

```
for(i=0;i<m;i++) {
    printf("%s\n",M[i]);
}
```