Fundamentos de Programação

Filipe Francisco

26 de abril de 2018

Comentários

- Trabalhando com vetores:
  - utiliza loops para ler, trabalhar, e imprimir
  - o ideal é ter um loop para ler, um para imprimir, e um (ou mais) para trabalhar com os elementos!
- Quantos se repetem mais:
  - início mais fácil da solução: ordenar o vetor, contar quantas vezes o elemento se repete mais vezes
  - problema: armazenar o elemento que se repete mais
    - pode ser que mais de um elemento se repita "mais vezes"
  - portanto, após contar a quantidade, devemos percorrer o vetor novamente e procurar pelos elementos que se repetem aquela quantidade de vezes!

Comentários

- Mercantil:
  - há 5 casos possíveis:
    - preço<chute1 e chute2='m': jogador 2 ganha
    - preço>chute1 e chute2='M': jogador 2 ganha
    - preço=chute1: jogador 1 ganha
    - preço<chute1 e chute2='M': jogador 1 ganha
    - preço>chute1 e chute2='m': jogador 1 ganha
  - em vez de cobrir cada caso, podemos cobrir apenas os casos para um dos jogadores!
    - se não cair em nenhum destes, significa que o outro jogador ganhou
  - esta ideia de cobrir apenas alguns casos (e deixar os outros como um "else") é uma ideia útil para alguns outros problemas

Comentários

- Ainda no problema Mercantil:
  - lembrem de usar double para receber os valores de preço e chute!
    - para evitar casos como o que aconteceu em Soldados Pequenos e Grandes
- Será que ganham ponto:
  - para este problema, utilizamos dois loops
  - cada loop seleciona um aluno, e verificamos se a soma dos dois números é igual ao número sorteado
  - problema: não podemos somar o número de um aluno duas vezes
    - isso seria basicamente selecionar o mesmo aluno duas vezes
  - para resolver esse caso, basta verificarmos se os dois alunos são diferentes!
  - ou seja, basta verificarmos se $i = j$
- E cuidado com ponto e vírgula na linha do for!
  - **for**(i=0;i<n;i++);{ ... }

Comentários

- No problema Mercantil, comentamos que no scanf, deveríamos inserir um espaço após cada "%lf"
  - scanf("%lf ",&valor[i]);
  - scanf("%c ",&chute2[i]);
- Isso relembra um problema anterior, da Calculadora, que exigia um
  - scanf("%d %d %c ",&v1,&v2,&op);
- Por que devemos fazer isso?
  - porque espaço e enter também são caracteres!
  - mais a frente, comentaremos sobre o buffer
- Como isso acontece?

Comentários

- Note que ao inserir um número antes de um caractere, você dá um espaço ou enter
- Problema: espaço e enter também são caracteres!
- Considere esse caso:
  - scanf("%d %d %c",&v1,&v2,&op);
- Se você não colocar espaços antes do %c, o programa lerá o espaço (ou enter) como o caractere!
- Portanto, a solução não é exatamente botar um espaço depois de lermos o número, mas sim colocar um espaço antes de lermos o caractere!
- Exemplo:
  - scanf("%d",&numero);
  - scanf("%c",&letra);
- O código acima não funcionará porque falta um espaço antes do %c

Comentários

- Vimos que sem colocar espaço antes do %c, acabamos lendo um espaço ou enter
- Pergunta curiosa: para onde vão os caracteres que eu digito e não são armazenados?
- Resposta: para o buffer do teclado!

- O que é o buffer?
  - é uma memória temporária que armazena o que foi digitado e ainda não foi utilizado (ou armazenado)
- Exemplo:
  - crie um programa que leia dois caracteres e imprima-os na ordem inversa de leitura

Comentários

- Exemplo:
  - crie um programa que leia dois caracteres e imprima-os na ordem inversa de leitura
- Lendo os caracteres:
  - scanf("%c",&letra1);
    scanf("%c",&letra2);
- Porém, a solução acima está incorreta. Por que?
  - sem o espaço antes do segundo %c, o segundo scanf lerá o espaço/enter!
- Como consertar? Duas opções:
  - scanf("%c",&letra1);
    scanf(" %c",&letra2);
  - scanf("%c %c",&letra1,&letra2);
- Boa prática: sempre colocar um espaço no scanf antes de qualquer %c

Strings

- E por que comentar sobre o buffer?
- Porque a partir de agora, o buffer vai ser uma coisa chata na nossa vida
- A partir de hoje, começamos a trabalhar com strings

- O que são strings?
  - são cadeias de caracteres
  - ou seja, são vetores de caracteres!
- Assim como em vetores, para criarmos uma string precisamos primeiro saber quantas letras precisamos armazenar!
- Porém, para trabalharmos com strings, é bom termos antes uma revisão sobre caracteres

Strings

- Caracteres são basicamente letras e símbolos que podemos inserir
- A linguagem C é *case sensitive*, ou seja, diferencia maiúsculas de minúsculas
- Cada caractere pode ser tratado como um inteiro para alguns fins
- Também temos "tipos" de caracteres:
  - dígitos: 0-9
  - dígitos hexadecimais: 0-9, A-F, a-f
  - letras minúsculas: a-z
  - letras maiúsculas: A-Z
  - letras: conjunto de letras maiúsculas e minúsculas
  - caractere alfanumérico: conjunto de dígitos, maiúsculas e minúsculas
  - dentre outros (como pontuação e espaçamento)

Strings

- A linguagem C também tem maneiras estranhas de trabalhar com caracteres
- Exemplo: checar se o caractere é maiúsculo ou minúsculo
  - if(c>='a' && c<='z') ...
  - if(c>='A' && c<='Z') ...
- Exemplo: transformar um caractere em minúsculo:
  - if(c>='A' && c<='Z') c=c+('a'-'A');
- Exemplo: transformar um caractere em maiúsculo:
  - if(c>='a' && c<='z') c=c-('a'-'A');
- Felizmente, temos uma biblioteca de funções que nos ajuda a evitar estas aberrações!

Strings

- Biblioteca "ctype.h"

- Exemplo: checar se o caractere é maiúsculo
  - if(isupper(c)) ...
- Exemplo: checar se o caractere é minúsculo
  - if(islower(c)) ...
- Exemplo: transformar um caractere em minúsculo:
  - c=tolower(c);
- Exemplo: transformar um caractere em maiúsculo:
  - c=toupper(c);

Strings

- Biblioteca "ctype.h"

- Exemplo: checar se o caractere é alfabético
  - if(isalpha(c)) ...
- Exemplo: checar se o caractere é alfanumérico
  - if(isalnum(c)) ...
- Exemplo: checar se o caractere é um dígito
  - if(isdigit(c));
- Exemplo: checar se o caractere é um dígito hexadecimal
  - if(isxdigit(c));
- Dentre outras

Strings

- Comentamos um pouco sobre caracteres porque, como vimos, uma string é um vetor de caracteres
- Portanto, trabalhar com letras de uma string é basicamente trabalhar com caracteres em um vetor
- Sabendo trabalhar com vetores, sabemos também trabalhar com os caracteres de uma string
- O que precisamos aprender de importante?
  - como criar uma string
  - como ler uma string
  - como imprimir uma string

String

- Criar uma string é algo intuitivo
  - uma string é um vetor de caracteres
  - portanto, basta criar um vetor de caracteres!
  - **char** str[n];
  - às vezes, também é útil criarmos uma string com quantidade fixa de caracteres
    - em alguns problemas, especificamos que a string terá no máximo *n* caracteres
  - **char** str[100];
- Imprimir uma string também é fácil
  - podemos imprimir caractere a caractere, mas é desnecessário
  - podemos imprimir utilizando um printf utilizando "%s"!
  - printf("%s",str);

String

- A maior dificuldade com strings em C é, pasmem, ler a string
- Vamos mostrar algumas maneiras de fazer, e problemas que podem acontecer
- Maneiras de ler uma string:
  - scanf(" %s",&str);
    - para ao encontrar um espaço ou enter
    - problema: estoura o tamanho da string caso o tamanho inserido seja maior que o tamanho da estrutura
    - obs: para gravar n caracteres, devemos criar uma string de tamanho $n+1$ (para guardar um caractere nulo no final)
    - obs: note o espaço no scanf!
  - gets(str);
    - para somente ao encontrar um enter
    - problema: estoura o tamanho da string caso o tamanho inserido seja maior que o tamanho da estrutura
    - obs: para gravar n caracteres, devemos criar uma string de tamanho $n+1$ (para guardar um caractere nulo no final)

String

- Mais maneiras de ler uma string:
  - fgets(str,**sizeof**(str),stdin);
    - a leitura para somente ao encontrar um enter
    - eficiente, sem estourar o tamanho
    - problema: complexo (sizeof, stdin)
    - problema: devemos criar uma string de tamanho $n + 2$ para armazenar o \n e o caractere nulo
  - scanf(" %100[^\n]",str);
    - lê até 100 caracteres, ou até encontrar um '\n'
    - neste estilo, a leitura para somente ao encontrar um enter
    - também é eficiente, não estoura tamanho
    - problema: útil apenas para casos com tamanho fixo da string
    - porém, é muito útil em casos onde a string tem até n elementos!
    - obs: para gravar n caracteres, devemos criar uma string de tamanho $n + 1$ (para guardar um caractere nulo no final)
    - obs: não colocar &!

String

- IMPORTANTE! Qual maneira usar no Moodle?

- Vimos que o scanf("%s") para ao ler um espaço, portanto não o usaremos
- O ideal seria usar o gets, pela simplicidade e por ele resolver nossos problemas
- Porém, como comentamos, o gets tem o problema de estourar o tamanho da string
  - isso significa que ele modifica posições da memória que ele não deveria!
- Portanto, o Moodle proíbe o uso do gets! :(
  - "warning: the 'gets' function is dangerous and should not be used."
- A segunda versão do scanf é útil, mas temos outro problema

String

- IMPORTANTE! Qual maneira usar no Moodle?

- A segunda versão do scanf é útil, mas temos outro problema
  - scanf(" %100[^\n]",str);
    - lê até 100 caracteres, parando ao encontrar um '\n'
  - problema: o '\n' é jogado no buffer!
  - ou seja, teremos problema para ler mais de uma string
  - isto é relativamente fácil de resolver, se tivermos de ler strings em um loop, fica mais complicado
- Portanto, utilizaremos o fgets!
  - fgets(str,num,stdin);
    - lê até *num* caracteres e armazena na string *str*
    - o fgets só para ao encontrar um '\n'
    - stdin: entrada do computador (basicamente é o que você digita!)

Strings

- Por que temos que criar strings com um (ou dois) caracteres a mais?
- A resposta envolve dois motivos
- Primeiro: caractere nulo no final da string
  - ao final de qualquer string, sempre é inserido um caractere nulo ('\0')
  - no caso de não termos uma posição sobrando, este caractere nulo irá sobrescrever um caractere da string!
    - ex: se lemos uma palavra de 10 caracteres e criamos uma string de tamanho igual a 10, o 10o caractere será apagado da string e no seu lugar, terá um caractere nulo!
- Segundo: caractere '\n'
  - se não armazenamos o "enter", ele será guardado no buffer
  - deste modo, quando lermos outro caractere (ou string), o buffer adicionará o caractere armazenado à string a ser lida!
- Obs: também temos uma biblioteca de funções para strings (string.h)!

Strings

- Biblioteca "string.h"

- Exemplo: comparar duas strings em ordem alfabética:
  - strcmp(str1,str2);

  - comp=strcmp(str1,str2);
    if(comp==0) printf("iguais"); //retorna 0
    if(comp<0) printf("str1 vem antes"); //retorna -1
    if(comp>0) printf("str2 vem antes"); //retorna 1

  - obs: caracteres maiúsculos são "menores" (vem antes) do que caracteres minúsculos
    - se compararmos "ABCDEF" e "abcdef", a função retorna -1
  - portanto ao comparar duas strings, as duas devem ter a mesma disposição de maiúsculas e minúsculas!
  - obs: espaços também contam como caracteres!
    - uma string "abcd " é maior que uma string "abcd"

Strings

- Biblioteca "string.h"

- Exemplo: calcular o tamanho de uma string
  - tam=strlen(str);

  - calcula o tamanho da string até (mas não incluindo) o caractere nulo (caractere que indica o fim da string), e retorna este tamanho
  - qual a diferença entre "**sizeof**(str)" e "strlen(str)"?
    - "**sizeof**(str)": tamanho do vetor criado para armazenar a string
    - inclui todas as posições

    - "strlen(str)": tamanho da string armazenada no vetor
    - inclui apenas posições preenchidas por caracteres até a primeira ocorrência de um caractere nulo, desconsiderando as demais posições

Strings

- Biblioteca "string.h"

- Exemplo: concatenar uma string com outra
  - strcat(str1,str2);

  - concatenar: unir duas strings em uma
  - str2 será copiada logo após str1
  - argumentos da função:
    1. primeira string
    2. segunda string
  - retorna a string resultante do concatenamento
  - importante: o resultado é gravado sobre str1!

Strings

- Biblioteca "string.h"

- Exemplo: concatenar uma string com **parte** de outra
  - strncat(str1,str2,n);

  - os *n* primeiros caracteres de str2 serão "colados" após str1
  - argumentos da função:
    1. primeira string
    2. segunda string
    3. quantidade *n* de caracteres de str2 que deve ser concatenada a str1 (o concatenamento utilizará os *n* primeiros caracteres)
       - obs: se quisermos concatenar a string 2 inteira, podemos utilizar a função strcat, ou então escolher a quantidade *n* como "strlen(str2)" ou "**sizeof**(str2)"
  - retorna a string resultante do concatenamento
  - importante: o resultado é gravado sobre str1!

Strings

- Biblioteca "string.h"

- Exemplo: procurar por um caractere em uma string
  - strchr(str,c);

  - busca pelo caractere *c* dentro da string str
    - se encontrar, retorna o próprio caractere *c*
    - se não encontrar, retorna NULL
  - como trabalhar com o NULL?
    - **if**(strchr(str,c)==NULL) printf("nao tem %c",c);
      else printf("tem %c sim!",c);

Strings

- Biblioteca "string.h"

- Exemplo: checar se uma string é substring de outra
  - strstr(str1,str2);

  - checa se str2 é substring de str1
  - o que é substring?
    - é uma string que, completa, é parte de outra
    - exemplo: "paga" é substring de "apagador"
    - obs: não compara caracteres nulos!
  - retorno da função:
    - se str2 não é substring de str1, retorna NULL
    - se str2 é substring de str1, retorna a string str1 a partir do ponto onde as duas são iguais
    - exemplo: se str1="qwertasdfg" e str2="rtas", strstr(str1,str2) retornará a string "rtasdfg"

Strings

- Biblioteca "string.h"

- Exemplo: copiar uma string
  - strcpy(str1,str2);

  - copia a str2 para a str1
    - "apaga" tudo o que estiver em str1
    - ex: se str1="qwertyuiop" e str2="123", esta função primeiro apaga str1, e depois escreve str2, logo str1 será igual a "123"
  - obs: str1 é sobrescrita neste processo, portanto perdemos a string anterior!