

**Universidade Federal de Pernambuco**

**Cin -Centro de Informática**

**Lista 4 – Introdução a Programação**

Prof: Adriano Sarmento

Data: 21.02.2013

Data de entrega: 07.03.2013

**Considerações:**

- É proibido usar a biblioteca conio.h;
- Leia a lista toda o quanto antes, para evitar más interpretações e muitas dúvidas em cima da hora;
- Envie uma prévia das questões, pelo menos um dia antes da data final da entrega, para o caso de acontecer algum imprevisto;
- A lista é para ser feita individualmente. Qualquer tentativa de cópia acarretará o zeramento da lista de todos os envolvidos;
- Em caso de dúvida, envie email para [duvidasip@googlegroups.com](mailto:duvidasip@googlegroups.com).
- Atenção para a liberação da memória no final dos programas, será cobrado que o espaço alocado no decorrer do programa seja totalmente liberado no final do mesmo.

1) Implemente as suas versões das seguintes funções que operam strings:

- Strchr, strrev, strcmp, strstr, strpbrk

Assinaturas das funções:

```
char* strchr(char* s, int c );  
char* strrev( char* s1 );  
int strcmp ( const char * str1, const char * str2 );  
char* strstr( const char *s1, const char *s2) ;  
char *strpbrk( const char *s1, const char *s2);
```

Implemente também a seguinte função:

```
char* strsplit (char* origem, char* s2, char simbolo)
```

Que divide uma string em duas a partir de um caracter.

Origem : string que irá ser dividida

S2 : string que receberá a string original até o caracter simbolo

Simbolo : caracter que dividirá a string

Retorno : ponteiro para um caracter após o símbolo

Exemplo:

```
char buffer [10];
```

```
char* ret = strsplit ("Hello World", buffer, ' ');
```

```
printf ("retorno : %s – buffer : %s", ret, buffer);
```

A saída deverá ser “retorno : World – buffer : Hello”

Obs: Não deverão ser utilizadas funções de outras bibliotecas.

Obs2: É importante saber o que as funções fazem exatamente, suas funções devem retornar exatamente o que as funções originais retornam.

Obs3: Os nomes dos parâmetros podem ser mudados caso ajude na legibilidade, mas a função deve ser a mesma.

2)Um mercado acaba de ser inaugurado e convidou os alunos de EC 2011.1 para criar um programa que organiza a quantidade de caixas necessárias para armazenar as frutas no seu estoque. Esse mercado ainda não sabe quão grande deverá ser sua remessa de frutas e para isso fez um acordo com o fornecedor de alimentos. O acordo é o seguinte: Inicialmente deverá ser dito quantas frutas serão pedidas na primeira remessa, essas frutas estarão guardadas em caixas de que cabem n frutas, esse valor n também será dado pelo usuário. A partir daí o usuário poderá fazer novos pedidos ao fornecedor dizendo a quantidade de frutas desejadas.

Essa quantidade deverá ser somada a quantidade que está no estoque. Também é possível vender as frutas, o que faz com que a quantidade da mesma no estoque diminua. Haverá um menu onde o usuário irá escolher entre pedir mais frutas ao fornecedor, vender as frutas que se encontram no seu estoque ou finalizar o programa. Cada vez que o usuário escolher um opção deverá ser mostrada a quantidade de caixas e frutas depois de realizar a operação.Essas informações também deverão ser mostradas após o pedido da primeira remessa.

Um vetor de inteiros deverá ser alocado dinamicamente da seguinte maneira: Cada posicao do vetor representa um caixa de frutas e e naquela posicao poderão ser colocadas n frutas. Quando não couber mais frutas naquela posicao o vetor deverá ser realocado para que caiba quantas caixas forem necessárias, e a quantidade restante de frutas deverá ser colocada nas novas posições. Caso seja vendida uma quantidade de frutas que deixe alguma caixa vazia o vetor também deverá ser realocado de maneira que ele tenha apenas a quantidade de caixas necessárias.

Obs: Quando não houver mais frutas o usuário só poderá pedir mais frutas ao fornecedor.

Obs2: Caso o usuário tente vender mais frutas do que há no estoque deverá aparecer a seguinte mensagem : "Não há x frutas no estoque", onde x é a quantidade que o usuário tentou vender.

Ex:

Entrada:

5 // quantidade de frutas por caixa

12 // quantidade de frutas da primeira remessa

//fim da primeira operação

pedir mais 4 frutas

//fim da segunda operação

vender 17 frutas

//fim da terceira operação

vender 12 frutas

//fim da quarta operação

Finaliza o programa

// última operação

Saida:

Você possui 3 caixas e 12 frutas //primeira operação

Você possui 4 caixas e 16 frutas //segunda operação

Não há 17 frutas no estoque! Você possui 4 caixas e 16 frutas ! //terceira operação

Você possui 1 caixa e 4 frutas //quarta operação

//sai do programa

Obs: as operações pedir e vender usadas no exemplo serão opções do menu

Demonstração do vetor:

5	5	2
---	---	---

0 1 2

5	5	5	1
---	---	---	---

0 1 2 3

4
---

0

**3)** Uma função muito útil em editores de texto é o “substituir”, que, recebendo dois textos, substitui o original por um novo em um documento. Você faz parte de uma equipe de programadores que estão desenvolvendo um editor de texto, e foi designado implementar esse recurso.

O seu objetivo é criar uma função que receba como parâmetro três elementos: Um documento de texto, e dois trechos (Original e o novo, respectivamente).

A função deve retornar um novo texto, com as devidas modificações e sem desperdício de memória (Ou seja, com um array de tamanho exato ao da string).

**Obs : Não é permitido usar a biblioteca `string.h` nessa questão**

Entrada:

O rato roeu a roupa do rei de roma.

//Texto a ser modificado

O rato

//Original

A ratazana

//Modificado

Saída:

A ratazana roeu a roupa do rei de roma.

4) Você já jogou Campo Minado? Bem, o objetivo do jogo é achar onde estão todas as minas no espaço  $N \times M$ . Para te ajudar, o jogo mostra um número num dos quadrados na qual indica quantas minas estão nos seus adjacentes. Por exemplo, no campo minado  $4 \times 4$  com 2 minas (na qual estão representadas pelo caractere "\*"):

```
* . . .  
. . . .  
. * . .  
. . . .
```

Para representar o mesmo campo substituindo cada quadrado por números como descrito acima, teremos:

```
*100  
2210  
1*10  
1110
```

Como você deve ter percebido, cada quadrado deve ter até no máximo 8 quadrados adjacentes.

A entrada consiste em receber vários campos minados. A primeira linha deverá conter dois inteiros  $N$  e  $M$  ( $0 < N, M \leq 100$ ), na qual  $N$  é o número de linhas e  $M$  o número de colunas da matriz. As próximas  $N$  linhas consistem exatamente  $M$  caracteres que representam o campo. Cada mina corresponde ao caractere "\*" e cada quadrado corresponde ao caractere ".". A entrada termina quando  $N$  e  $M$  forem iguais a zero.

A cada campo minado computado, imprima "Campo #x:", no qual  $x$  é o número do campo (inicialmente igual a 1). Sua função é criar um campo minado correspondente de acordo com a forma descrita acima, substituindo os quadrados pela quantidade de minas adjacentes.

Obs.: Você deverá acessar cada posição das matrizes usando aritmética de ponteiros OBRIGATORIAMENTE. Além disso, é proibido usar matriz estática. Utilize alocação dinâmica.

Entrada:

4 4

\*...

....

.\*. .

....

3 5

\*\*...

.....

.\*. .

0 0

Saída:

Campo #1:

\*100

2210

1\*10

1110

Campo #2:

\*\*100

33200

1\*100

5) Rebeca, Moisés e Artur são estudantes do CIn que gostam muito de jogar um popular jogo de cartas chamado UNO, e eles pedem a ajuda de você, estudante de IP 2012.2 para implementar uma versão do UNO em C, pois eles estão ocupados estudando GDI.

O seu programa deverá conter as seguintes características:

1. O usuário pode escolher o número de jogadores, que varia entre 3 e 6, inclusive.
2. Os jogadores começam com 7 cartas.
3. A mão dos jogadores deverá ser armazenada em vetores e o tamanho desses vetores deverá ser ajustado conforme necessário (quando um jogador recebe cartas ou descarta cartas, o vetor deve ser realocado para gastar o mínimo de espaço possível).
4. As cartas deverão ser representadas como números INTEIROS.
5. Não existe critério para limitar a quantidade de cartas, e as cartas especiais aparecem em igual chance em relação às cartas normais.
6. As cartas especiais são:
  - a. +1 carta para o anterior.
  - b. +2 cartas para o próximo.  
Obs: O jogador que receber +2 não joga, passando a vez.
  - c. Pular o próximo jogador.
  - d. Reverter o sentido do jogo.
7. As cartas têm 2 informações, uma delas é a categoria (A,B,C,D) e a outra é o seu número (0~9, +1, +2, P, R). Exemplos de cartas : 2A, +1D, RB, 6C, PA
8. Para poder jogar uma carta, a carta da mesa deverá ter uma das informações igual a da carta que o jogador pretende jogar.
  - a. Exemplo:  
Carta da mesa : 9A  
Só se pode jogar cartas 9 ou cartas da categoria A
9. Se o jogador não tiver nenhuma carta para jogar, ele deve receber uma carta, e se esta carta puder ser jogada, isto acontece, caso contrário, o jogador passa a vez sem jogar nada.
10. Ganha o jogo aquele jogador que descartar todas as suas cartas.
11. O valor numérico de (+1, +2, P e R) são respectivamente (10,11,12 e 13).

Observações / Dicas:

1. Utilize as operações de módulo e de divisão para obter as informações das cartas.  
Exemplos :  
Carta 8A = 08  
 $08 / 14 = 0$  (carta do tipo A)  
 $08 \% 14 = 8$  (numero da carta)  
  
Carta 4C = 32  
 $32 / 14 = 2$  (carta do tipo C)  
 $32 \% 14 = 4$  (numero da carta)  
  
Carta +2D = 53  
 $53 / 14 = 3$  (carta do tipo D)  
 $53 \% 14 = 11$  (11 representa +2)
2. Utilize a função "rand()" na hora de distribuir cartas aos jogadores.
3. Crie um vetor extra para armazenar a quantidade de cartas de cada jogador.