Exame de Programação Imperativa

Exercício 1. Rodar uma letra

Escreva uma definição da função

```
char rodar_letra(char ch);
```

para efetuar uma "rotação" por uma posição de um carater representado uma letra: `A` passa a `B`, `B` passa a `C`, etc. até `Z` que passa a `A`.

$$\begin{array}{cccccc} A & B & C & \cdots & Y & Z \\ \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow \\ B & C & D & \cdots & Z & A \end{array}$$

A função deve retornar o código do carater rodado e deve funcionar para letras maiúsculas e minúsculas sem acentos; para carateres que não são letras o resultado deve ser o caratere original.

Exemplos: `rodar_letra('B')` dá `'C'`; `rodar_letra('c')` dá `'d'`; `rodar_letra('?')` dá `'?'`.

Exercício 2. Próximo número primo

Recorde que um número inteiro $n \geq 0$ diz-se *primo* se tem exatamente dois divisores inteiros positivos: 1 e n. Note que 1 não é primo porque tem apenas um divisor positivo.

Escreva uma definição da função

```
int proximo_primo(int n);
```

cujo resultado é o primeiro número primo maior do que `n`.

Exemplos: `proximo_primo(0)` dá resultado 2; `proximo_primo(2)` dá resultado 3; `proximo_primo(5)` dá resultado 7.

Sugestão: começe por definir uma função auxiliar para testar se um inteiro é primo (resultado 0 ou 1).

Exercício 3. Contar montanhas

Um alpinista efetua as suas caminhadas usando um pedómetro muito peculiar que regista sequências de letras:

- uma letra `S` cada vez que dá um passo a subir;
- uma letra `D` cada vez que dá um passo a descer.

O alpinista começa sempre ao nível do mar (altitude 0). Cada passo representa uma mudança de 1 unidade de altitude (positiva ou negativa, conforme a letra).

Vamos considerar que uma **montanha** é uma sequência consecutiva de passos acima do nível do mar (isto é, com altitude positiva), começando num passo ascendente a partir do nível do mar e terminando num passo descendente para o nível do mar.

Exemplo: se a sequência registada for

`SSDSDDSDDDSS`

então o alpinista sobe duas vezes para uma montanha de altura 2; desce e sobe ainda na mesma montanha; depois desce ao nível do mar e sobe uma outra montanha de altura 1; por último, desce para altura -2 e volta a subir para o nível do mar. Logo, nesta caminhada encontrou apenas duas montanhas (ver figura).

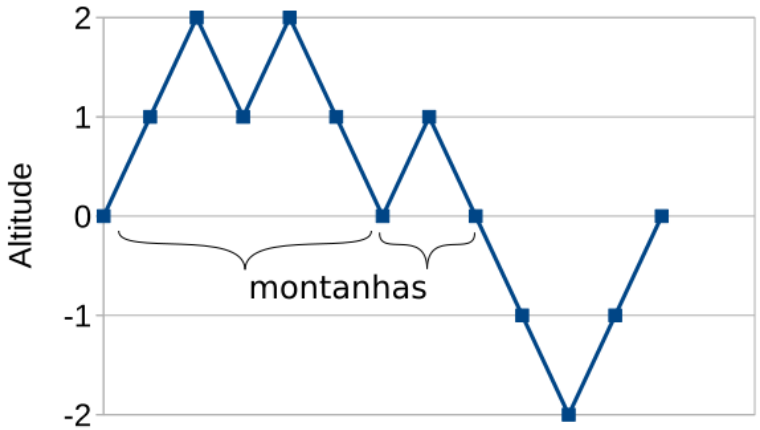

Figure 1:

Defina uma função

```
int montanhas(char str[]);
```

que conta o número de montanhas encontradas numa caminhada. O argumento `str` é uma cadeia com carateres `S` e `D` e terminada pelo carater `\0`.

O resultado retornado deve ser o número de montanhas encontradas (um inteiro não-negativo). Pode assumir que o percurso começa e termina sempre no nível do mar (altitude 0).

Exemplo: `montanhas("SSDSDDSDDDSS")` dá resultado 2.

Exercício 4. Ordenação "bubblesort"

O "bubblesort" é um algoritmo simples para ordenação que não foi apresentado nas aulas. A ideia deste algoritmo é efetuar múltiplas passagens sobre a sequência de valores, comparando pares adjacentes de valores e trocando-os quando estão fora de ordem.

Sejam $v[0], v[1], v[n-1]$ a sequência de n valores; então uma passagem do "bubblesort" corresponde ao seguinte pseudo-código:

repetir para i de 1 até $n-1$ inclusivé :
  se $v[i-1] > v[i]$ então trocar $v[i-1]$ com $v[i]$

Para fazer a ordenação completa basta repetir passagens até que deixem de ocorrer trocas; então todos os valores estarão na ordem correta.

Implemente este algoritmo em C como uma função

```
void bubblesort(int vec[], int n);
```

Questionário de escolha múltipla

- Para cada uma das seguintes questões apenas uma das alternativas é correta
- Não responda se acha que não sabe: **respostas erradas descontam metade da cotação de uma resposta certa**
- Não obterá "feedback" das respostas corretas durante o exame; o resultado será apenas *Accepted*
- Pode responder múltiplas vezes; apenas a última submissão será cotada

Quais os valores das variáveis no final do seguinte fragmento de programa?

```
int a, b, c;
a = b = c = 1;
a += 2;
b += a+b;
c += a+b;
```

(a) a=3, b=5, c=9
(b) a=2, b=2, c=2
(c) a=3, b=4, c=4

Suponha que `i` e `j` são variáveis inteiras. Qual das seguintes expressões exprime a condição "`i` está entre 0 e `j`"?

(a) `i>=0 || i<=j`
(b) `i>=0 && i<=j`
(c) `0 <= i <= j`

Qual dos seguintes fragmentos de programa calcula a soma $s = 1^2 + 2^2 + 3^2 + \cdots + n^2$ para $n \geq 0$? (Assuma que não ocorre *overflow*).

(a)
```
int s = 0;
int i = 1;
while (i<n) {
    s = s + i*i;
    i++;
}
```

(b)
```
int s = 0;
int i = 1;
while (i<=n) {
    s = s + i*i;
    i ++;
}
```

(c)
```
int s = 1;
int i = 0;
while(i < n) {
   s = s + s*s;
   i++;
}
```

O seguinte fragmento de programa imprime alguns inteiros dentro de um intervalo $[a, b]$:

```
// a, b são inteiros
for(int k = a; k <= b; k++) {
    if(k%2==0 && k%3!=0)
        printf("%d\n", k);
}
```

Indique qual das seguintes opções descreve os valores impressos.

(a) os inteiros impares que são múltiplos de 3
(b) os inteiros pares que não são múltiplos de 3
(c) o primeiro inteiro impar que não é múltiplo de 3

Considere a seguinte função "mistério":

```
int fun(int n) {
  int r = 1;
  for(int i = 0; i<n; i++) {
    r = r*2;
  }
  return r;
}
```

Qual das seguintes expressões representa o resultado calculado pela função quando $n \geq 0$? (Assuma que não ocorre *overflow*.)

(a) $\underbrace{2 \times 2 \times \cdots \times 2}_{n \text{ fatores}}$ ou seja 2^n

(b) $\underbrace{2 + 2 + \cdots + 2}_{n \text{ termos}}$ ou seja $2 \times n$

(c) $\underbrace{1 \times 2 \times \ldots \times n}_{n \text{ fatores}}$ ou seja $n!$

Considere a seguinte definição incompleta de uma função que calcula o mdc pelo algoritmo de Euclides usando subtrações.

```
int mdc(int a, int b) {
   while (..?..) {   // completar
      if (a > b)
         a = a - b;
      else
         b = b - a;
   }
   return a;
}
```

Indique qual deve ser a condição do ciclo:

(a) `a == b`

(b) `a != b`

(c) `a >= b`

Considere o seguinte fragmento de programa que processa uma linha de texto da entrada-padrão:

```
int ch, n = 0;
while((ch=getchar()) != '\n') {
   if (isalpha(ch) || isdigit(ch))
       n++;
```

```
}
printf("%d", n);
```

Indique a alternativa que descreve o que este fragmento faz:

(a) conta o número de carateres que são letras ou algarismos
(b) conta o número de carateres que não são letras nem algarismos
(c) soma os códigos dos carateres que são letras ou algarismos

Considere a seguinte definição incompleta de uma função que implementa pesquisa sequencial da primeira ocorrência de um valor `x` numa variável indexada `vec[]` com `n` elementos.

```
int pesquisa(int vec[], int n, int x) {
   int i = 0;
   while(i < n && ..?..)   // completar
     i++;
   if(i < n)
     return i;  // encontrou
   else
     return -1; // não encontrou
}
```

Qual das alternativas completa a condição em falta?

(a) `vec[i] == x`
(b) `vec[i] != x`
(c) `vec[i] < x`

Considere a seguinte definição incompleta de uma função que implementa **ordenação por seleção**:

```
void select_sort(int vec[], int n) {
  int i, j;
  for(i = 0; i < n; i++) {
    int imin = i; // índice inicial do mínimo
    for(j = i+1; j < n; j++) {
      if(..?..) imin = j; // completar
    }
    // trocar o mínimo com vec[i]
    if(imin != i) {
      int temp = vec[i];
      vec[i] = vec[imin];
      vec[imin] = temp;
    }
  }
}
```

Qual das alternativas completa a condição em falta?

(a) `vec[j] < vec[i]`
(b) `vec[j] > vec[imin]`
(c) `vec[j] < vec[imin]`

Considere a seguinte função "mistério":

```
int fun(char *ptr) {
   int n = 0;
   while(*ptr != '\0') {
      if (*ptr != ' ' && *ptr != '\n')
          n++;
      ptr++;
   }
   return n;
}
```

Indique qual das seguintes alternativas descreve a operação implementada por esta função:

(a) conta o número de espaços e mudanças de linha numa cadeia
(b) conta o número de carateres diferentes de espaço e mudança de linha
(c) verifica se uma cadeia contém algum espaço ou mudança de linha