Departamento de Ciência de Computadores FCUP

Introdução à Programação (CC111) 2011/2012

2º Teste Escrito (04.01.2012) *duração: 3 horas*

(Cotação: 1×2, 2.5, 5+1, 4, 1.5, 4)

N.º Nome

Na resposta às questões, sempre que for útil, pode definir funções auxiliares.

1. Em cada uma das alíneas, diga o que escreve cada fragmento de código dado, supondo que as variáveis `a`, `b` e `c` são tipo `int` e têm inicialmente os valores `23`, `4`, e `4`.

a)

```
if (b == c) b = 1;
if (c == 1){
   a = b;
   b = a;
   c = 0;
}
printf("(%d,%d)\nc=%d",a,b,c);
```

b)

```
b = 2;
a = 0;
do {
   b = b*b;
   c -= 1;
} while (++a < c);
printf("a=%d\nb=%d\nc=%d",a,b,c);
```

2. Implemente uma função `void inverte(char x[])` para inverter a sequência de caracteres bem formada (isto é, uma *string*) colocada a partir da posição dada por `x`. Por exemplo, se tiver `"Exemplificar"`, deveria ficar com `"racifilpmexE"`.

(CONTINUA, v.p.f.)

CC111 2.º Teste N.º | Nome

3. Sem usar variáveis indexadas, implemente uma função para ler da entrada padrão uma sequência de inteiros terminada por `0` e determinar a fracção dos que não são múltiplos de `n` e diferem de `p` mais do que `k` unidades, sendo `n`, `p` e `k` passados como argumentos na chamada da função e do tipo `int`. Por exemplo, para `-6 4 -7 3 8 -5 -9 16 5 8 10 14 0`, a fracção dos que diferem de `9` mais do que `5` unidades e não são múltiplos de `-2` seria `4/12`, ou seja `1/3`. Admita que a sequência tem algum elemento além do terminador `0`, e que `k` é não negativo, e `n` pode ser `0`. A fracção (valor de retorno da função) deve ser expressa por uma fracção irredutível a/b, em que b pode ser `1` se $a/b = 0$ ou $a/b = 1$. O tipo do resultado deve corresponder a `struct fraccao {int a, b;}`.
(NB: Com penalização de 1.5 valores, pode considerar que o valor de retorno é do tipo `double`).

Supondo que não será relevante considerar erros de *underflow* ou de *overflow*, justifique sucintamente a correção da implementação.

(CONTINUA, v.p.f.)

CC111 2.ºTeste N.º _____ Nome _____

4. Implemente apenas uma das duas funções seguintes. Inclua comentários para documentar as ideias principais da sua implementação.

a) `int acordo_porto(char s[])`, ou equivalentemente, `int acordo_porto(char *s)`, para substituir todas as ocorrências de `v` por `b` e de `V` por `B`, e de `cc` por `c`, e de `CC` por `C`, na sequência de caracteres (bem formada) dada por `s`. A função retorna o número de elementos da sequência compactada, que ficará colocada a partir da posição apontada por `s`. Por exemplo, se a sequência inicial fosse `"Para o Porto, vou fazer uma correccao nova, bem direccionada e intencionada"`, o resultado seria `"Para o Porto, bou fazer uma correcao noba, bem direcionada e intencionada"`. Não existirão letras acentuadas, nem `c`'s com cedilhas e nunca terá três ou mais `c`'s (ou `C`'s) consecutivos, nem junções da forma `cC` nem `Cc`.

b) `int soma(char x[],char y[],char z[],int m)` para somar dois inteiros representados na base 10 pelas sequências de caracteres dadas por `x` e `y`, se ***lidas da direita para a esquerda***. O resultado deve ficar no vetor `z` e não ocupar mais do que `m` posições. A função retorna `1` se o resultado não puder ser calculado e `0` se puder. Assim, a chamada `soma("7699","531",res,5);` deve retornar `1` porque `"20101"` requer 6 posições, mas `soma("98","135",res,5);` retorna `0`. No primeiro caso, no final, teria `"2010"` a partir de `res` e no segundo `"026"`. Note que se supõe que as sequências dadas e a obtida terminam com o caracter de código zero.

(CONTINUA, v.p.f.)

5. Na continuação do exercício anterior, compare os dois programas seguintes e explique que interesse pode ter a função `soma` para somar números positivos (não dando grande importância a `inverte` que introduzimos aqui para tornar a implementação de `soma` mais simples).

```
#include <stdio.h>
void inverte(char x[]);
int soma(char x[],char y[],char z[],int m);
int main() {
  char d1[50], d2[50], r[50];
  scanf("%s %s",d1,d2);  // seqs. de digitos
  inverte(d1);
  inverte(d2);
  soma(d1,d2,r,50);
  inverte(r);
  printf("%s\n",r);
  return 0;
}
// codigo de inverte e soma omitido
```

```
#include <stdio.h>
int main() {
  int d1, d2;
  scanf("%d %d",&d1,&d2);
  printf("%d\n",d1+d2);
  return 0;
}
```

6. Um ficheiro contém dados sobre *npols* polígonos existentes numa figura (o primeiro valor no ficheiro é *npols* e não excede 300). Para cada polígono tem o número de vértices, um identificador (inteiro), e as coordenadas dos vértices num referencial cartesiano (o.n.). Os vértices são dados no sentido anti-horário e têm coordenadas inteiras. Sabe-se que, nessas condições, se $v_0, \ldots, v_{n-1}$ for a sequência de vértices de um polígono P, com $v_i = (x_i, y_i)$, para todo i, o dobro da área de P pode ser calculado usando a expressão $2\mathcal{A}(P) = \sum_{i=0}^{n-1}(x_i y_{i+1} - y_i x_{i+1})$. Recorde que o polígono é definido pelos segmentos $v_i v_{i+1}$, para $0 \leq i < n$, e $v_{n-1}v_0$. Implemente um programa completo para re-ordenar os polígonos por ordem crescente de número de vértices, e, em caso de empate, por ordem crescente de área, e se persistir o empate, por ordem crescente de identificador. Os dados são lidos da entrada padrão e os resultados escritos na saída padrão (com redirecionamento de e para ficheiros). O ficheiro de dados não tem polígonos iguais (i.e., com o mesmo identificador) e tem um formato semelhante ao apresentado abaixo à esquerda (à direita tem o resultado esperado).

```
3
3 178
1 0 1 1 0 0
4 13
-2 3 2 3 1 5 -2 5
3 171
5 5 0 0 5 0
```

```
3
3 178
1 0 1 1 0 0
3 171
5 5 0 0 5 0
4 13
-2 3 2 3 1 5 -2 5
```

O programa tem de incluir o código seguinte, que deverá analisar e **completar**.

(CONTINUA, v.p.f.)

```
typedef struct poligono {
  int id, nv, darea;
  VERTICE *pv;  // campo guardará endereço a partir do qual tem as coordenadas dos vertices
} POLIGONO;


VERTICE *coordsverts(int n);  // ler e guardar coordenadas dos n vertices de um poligono
void carregar_dados();   // inicializa as variaveis globais Pols e NumPols
int dobroArea(VERTICE *pvert, int n);  // tem n vertices em posições consecutivas a partir de pvert
int comparar(POLIGONO p, POLIGONO q); // -1 se p antes de q, 1 se p depois de q, 0 se forem iguais
void c_ordena();  // segundo o criterio definido e usando a funcao comparar para comparar
void escrResultado();  // escreve com formato identico ao formato dos dados


POLIGONO Pols[MAXPOLS];
int NumPols;


VERTICE *coordsverts(int n) {
   VERTICE *p = malloc(sizeof(VERTICE)*n);   // espaço para n vertices do tipo VERTICE
   int i;
   for(i=0;i<n;i++)
     scanf("%d %d",&p[i].xc,&p[i].yc);
   return p;  // o endereço do primeiro vertice
}

void carregar_dados() {
   int i;
   scanf("%d",&NumPols);
   for(i=0;i<NumPols;i++) { // inicializar Pols[i]
     // ler e guardar numero de vertices, identificador, e coordenadas



     // calcular o dobro da área e guardar no campo darea


  }
}
```

// pode usar também o verso da folha (FIM)