MAC2166 – Introdução à Computação para Engenharia

Escola Politécnica

Segunda Prova – 21 de maio de 2012

Nome: _____

Assinatura: _____

Nº USP: _____ Turma: _____

Professor: _____

Instruções:

1. Não destaque as folhas deste caderno.
2. A prova consta de 3 questões. Verifique antes de começar a prova se o seu caderno de questões está completo.
3. A prova pode ser feita a lápis. Cuidado com a legibilidade.
4. Qualquer questão pode ser resolvida em qualquer página. Se a questão não está na página correspondente ao enunciado basta indicar isto na página e escrever QUESTÃO X em letras ENORMES antes da solução.
5. Não é necessário apagar rascunhos no caderno de questões.
6. Não é permitido o uso de folhas avulsas para rascunho.
7. Não é permitido o uso de equipamentos eletrônicos.
8. Não é permitido a consulta a livros, apontamentos ou colegas.

DURAÇÃO DA PROVA: 2 horas

Questão	Valor	Nota
1	2,5	
2(a)	2,0	
2(b)	0,5	
3(a)	0,5	
3(b)	0,5	
3(c)	1,0	
3(d)	3,0	
Total	10,0	

QUESTÃO 1 (vale 2,5 pontos)

Simule a execução do programa abaixo, destacando a sua saída. A saída do programa consiste de tudo que resulta dos comandos `printf`. **Para efeito de correção só será considerada a saída do programa**.

```
#include <stdio.h>

float f1(int a, int b) ;
int f2(int *a, int *b) ;


int main() {
  int nusp, a, b, c;
  float x ;
  char ch ;

  printf("Digite o seu No. USP: ") ;
  scanf("%d", &nusp) ;

  a = nusp%5 + 2;
  b = 7 - a ;
  printf("a=%d b=%d\n", a, b) ;
  x = f1( a , b ) ;
  c = x + 2 ;
  printf("a=%d b=%d c=%d x=%f\n",
          a, b, c, x) ;

  a = nusp%5 + 2;
  b = 7 - a ;
  c = 2*b - 1;
  printf("a=%d b=%d c=%d\n", a, b, c) ;
  b = f2( &a , &c ) ;
  printf("a=%d b=%d c=%d\n", a, b, c) ;

  ch = 'a' + b ;
  printf("ch=%c\n", ch) ;

  return 0 ;
}
```

```
float f1(int a, int b) {
  int c ;
  float x, y;

  c = (a+b)/2 ;
  x = (a+b)/2 ;
  y = a ;
  printf("c=%d  x=%f  y=%f\n", c, x, y) ;

  y = (y+b)/2 ;
  printf("y=%f\n", y) ;

  return y ;
}
```

```
int f2(int *a, int *b) {
  int c ;

  if(*a%2==0) *a=*a+1 ;
  else *a=*a+2 ;
  c = (*b) + (*a) + 1;
  *b = c/2 ;
  printf("*a=%d *b=%d c=%d\n", *a, *b, c) ;

  return c%3+1 ;
}
```

Escreva a saída do programa no espaço abaixo. As demais áreas brancas podem ser usadas livremente para fazer a simulação.

A solução depende do resto da divisão do número USP por 5 `nusp%5`.

Saída do programa

```
Digite o seu No. USP: "nusp final 0 ou 5"
a=2 b=5
c=3  x=3.000000  y=2.000000
y=3.500000
a=2 b=5 c=5 x=3.500000
a=2 b=5 c=9
*a=3 *b=6 c=13
a=3 b=2 c=6
ch=c

Digite o seu No. USP: "nusp final 1 ou 6"
a=3 b=4
c=3  x=3.000000  y=3.000000
y=3.500000
a=3 b=4 c=5 x=3.500000
a=3 b=4 c=7
*a=5 *b=6 c=13
a=5 b=2 c=6
ch=c

Digite o seu No. USP: "nusp final 2 ou 7"
a=4 b=3
c=3  x=3.000000  y=4.000000
y=3.500000
a=4 b=3 c=5 x=3.500000
a=4 b=3 c=5
*a=5 *b=5 c=11
a=5 b=3 c=5
ch=d

Digite o seu No. USP: "nusp final 3 ou 8"
a=5 b=2
c=3  x=3.000000  y=5.000000
y=3.500000
a=5 b=2 c=5 x=3.500000
a=5 b=2 c=3
*a=7 *b=5 c=11
a=7 b=3 c=5
ch=d


Digite o seu No. USP: "nusp final 4 ou 9"
a=6 b=1
c=3  x=3.000000  y=6.000000
y=3.500000
a=6 b=1 c=5 x=3.500000
a=6 b=1 c=1
*a=7 *b=4 c=9
a=7 b=1 c=4
ch=b
```

QUESTÃO 2 (vale 2,5 pontos)

ITEM 2(a) (vale 2,0 pontos) Escreva uma função de protótipo

```
float seno(float x, float eps) ;
```

que recebe um número real `x` (ângulo em radianos) e um número real `eps` (precisão), e devolve uma aproximação de `seno(x)` obtida pela soma de todos os termos da série abaixo até que $\left|(-1)^k \frac{x^{2k+1}}{(2k+1)!}\right| <$ `eps`. O primeiro termo menor que `eps` deve ser incluído na aproximação.

$$\texttt{seno(x)} = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots + (-1)^k \frac{x^{2k+1}}{(2k+1)!} + \ldots$$

Caso ache conveniente, escreva e use funções adicionais. Não é permitido o uso de funções da biblioteca matemática da C.

```
/*
 * funcao que recebe um angulo x em radianos e um
 * numero real eps e devolve a aproximacao de
 * sen(x) com precisao eps.
 *
 */

float sen(float x, float eps)
{
  float termo;  /* armazena um termo da serie */
  float seno;   /* aproximacao de seno(x) */
  int i;        /* armazena 2k */

  /* 1. inicialize as variaveis */
  termo = x;
  seno  = x;
  i     = 2;

  /* 2. calcule a expressao que aproxima o seno */
  while (termo <= -eps || eps <= termo)
    {
      /* 2.1. determine o proximo termo */
      termo = -termo*x*x/(i*(i+1));

      /* 2.2. adicione o k-esimo termo a aprox. do seno */
      seno = seno + termo;

      /* 2.3. atualize o valor de 2k */
      i = i + 2;
    }

  return seno;
}
```

ITEM 2(b) (vale 0,5 pontos)

Escreva um programa em C que lê um número real θ (ângulo em graus) e imprime uma aproximação do seno de θ, usando `eps` = `0.000001`. Use a função do item anterior, mesmo que você não a tenha feito. Não é necessário transcrever aqui nem o protótipo nem a função do item anterior.

Para converter o ângulo de graus para radianos, considere o valor de π definido pela constante PI:

```
#define PI  3.1415926


#define EPS 0.000001

/*
 *  MAIN
 *
 */
int main()
{
  float teta;     /* valor em graus do seno a ser calculado */
  float rad;      /* valor em radianos do seno a ser calculado  */

  /* 1. leia o valor do angulo teta */
  printf("Entre com angulo teta: ");
  scanf("%f", &teta);

  /* 2. tranforme o angulo teta para radianos */
  rad = (PI*teta)/180;

  /* 3. imprima a aproximacao calculada */
  printf ("sen(%f,%f) = %f \n", rad, EPS, sen(rad,EPS));

  return 0;
}
```

QUESTÃO 3 (vale 5,0 pontos)

Esta questão consiste de quatro itens e em cada item deve ser escrita uma função. Todas as funções são semelhantes a alguns trechos de código que você escreveu para o seu EP3. Os nomes das funções são `distancia`, `colisao`, `atualiza` e `main`.

Considere dada a função

```
float raizquadrada(float x);   /* calcula a raiz quadrada de um numero */
```

Em cada item podem ser utilizadas as funções dadas ou de itens anteriores. Não há necessidade de transcrever os protótipos ou as funções dos itens anteriores. As funções de itens anteriores podem ser usadas mesmo que elas não tenham sido escritas.

Angry Bixos Episódio II: O Império dos Veteranos Contra-Ataca

Os veteranos da Poli desenvolveram um sistema anti-bixos, que consiste no lançamento de um projétil de interceptação. Agora temos dois projéteis, um disparado pelos bixos, e outro disparado pelos veteranos. Os projéteis possuem formato circular e são descritos pelas coordenadas do seu centro e pelo seu raio.

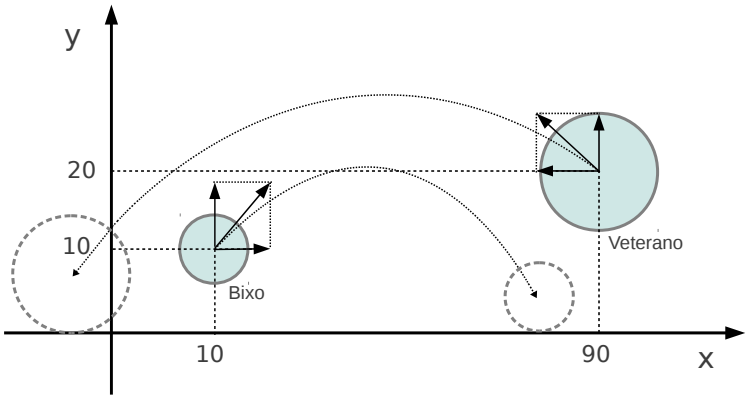

Figura 1: Representação dos projéteis dos bixos e veteranos em posição de batalha.

ITEM 3(a) (vale 0,5 ponto)

Escreva uma função de protótipo:

```
float distancia(float x1, float y1, float x2, float y2);
```

que recebe as coordenadas de dois pontos em `(x1,y1)` e `(x2,y2)` e devolve a distância euclidiana entre eles.

A sua função `distancia` deve usar a função `raizquadrada` dada.

```
/*
 * Solucao curta e grossa
 */
float distancia(float x1, float y1, float x2, float y2)
{
   return raizquadrada((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2));
}


/*
 * Solucao com uma variavel auxiliar
 */
float distancia(float x1, float y1, float x2, float y2)
{
   float dist;

   dist = raizquadrada((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2));

   return dist;
}
```

ITEM 3(b) (vale 0,5 ponto)

Escreva uma função de protótipo:

```
int colisao(float x1, float y1, float r1, float x2, float y2, float r2);
```

que recebe em `(x1, y1)` e `(x2, y2)` as coordenadas dos centros de dois projéteis e em `r1` e `r2` os seus repectivos raios. A função deve devolver **1** se nessas condições há uma colisão (alguma sobreposição) entre eles e deve devolver **0** em caso contrário.

A sua função `colisao` deve usar **obrigatoriamente** a função `distancia` do item 3(a).

```
/*
 * Solucao 1: A solucao podia usar '<' no lugar de '<='
 *   Com float a comparacao == nao faz sentido...
 */
int colisao(float x1, float y1, float r1, float x2, float y2, float r2)
{
   if (distancia(x1,y1,x2,y2) <= r1+r2)
     return 1;

   return 0;
}

/*
 * Solucao 2: A solucao podia usar '<' no lugar de '<='
 *   Com float a comparacao == nao faz sentido...
 */
int colisao(float x1, float y1, float r1, float x2, float y2, float r2)
{
   int resp;

   resp = 0;
   if (distancia(x1,y1,x2,y2) <= r1+r2)
     resp = 1;

   return resp;
}
```

ITEM 3(c) (vale 1,0 ponto) Seja

```
#define GT 9.807 /* Aceleracao da gravidade da Terra */
```

Considere as coordenadas $(\mathtt{x}^{\mathtt{t-dt}}, \mathtt{y}^{\mathtt{t-dt}})$ e o vetor velocidade $(\mathtt{velX}^{\mathtt{t-dt}}, \mathtt{velY}^{\mathtt{t-dt}})$ de um projétil em um instante $\mathtt{t-dt}$. O projétil movimenta-se a uma velocidade horizontal constante e a uma velocidade vertical que depende da força gravitacional agindo sobre ele. As coordenadas $(\mathtt{x}^{\mathtt{t}}, \mathtt{y}^{\mathtt{t}})$ e o vetor velocidade $(\mathtt{velX}^{\mathtt{t}}, \mathtt{velY}^{\mathtt{t}})$ do projétil no instante $\mathtt{t}$ podem ser calculadas da seguinte forma:

$$\begin{aligned} \mathtt{x}^{\mathtt{t}} &= \mathtt{x}^{\mathtt{t-dt}} + \mathtt{velX}^{\mathtt{t-dt}} \times \mathtt{dt}\,, \\ \mathtt{y}^{\mathtt{t}} &= \mathtt{y}^{\mathtt{t-dt}} + \mathtt{velY}^{\mathtt{t-dt}} \times \mathtt{dt} - 0.5 \times \mathtt{GT} \times \mathtt{dt} \times \mathtt{dt}\,, \\ \mathtt{velY}^{\mathtt{t}} &= \mathtt{velY}^{\mathtt{t-dt}} - \mathtt{GT} \times \mathtt{dt}\,, \end{aligned}$$

e $\mathtt{velX}^{\mathtt{t}} = \mathtt{velX}^{\mathtt{t-dt}}$ se o projétil não tocou o chão. Caso o projétil toque o chão, ele deve permancer parado; isto é, $\mathtt{velX}^{\mathtt{t}} = 0$, $\mathtt{velY}^{\mathtt{t}} = 0$ e sua coordenada vertical deve ser ajustada para que ele fique tangente ao chão (o projétil não afunda no chão). O chão tem coordenada `y` igual a zero.

Escreva uma função de protótipo:

```
void atualiza(float *x, float *y, float *velX, float *velY, float r, float dt);
```

que recebe as coordenadas `(*x,*y)` e o vetor velocidade `(*velX, *velY)` de um projétil em um instante $\mathtt{t-dt}$. A função recebe ainda o raio `r` do projétil e um intervalo de tempo `dt` (passo da simulação). A função deve devolver em `(*x,*y)` a posição e em `(*velX, *velY)` o vetor velocidade do projétil no instante `t` conforme descrito acima.

```
/*
 * POSSIVEL solucao
 */
void atualiza(float *x, float *y, float *velX, float *velY, float r, float dt)
{
  *x    = *x + *velX*dt;
  *y    = *y + *velY*dt - 0.5*GT*dt*dt;
  *velY = *velY - GT*dt;

  /* o projetil tocou o chao? */
  if (*y <= r)
    {
      *velX = 0;
      *velY = 0;
      *y    = r;
    }
}
```

ITEM 3(d) (vale 3,0 pontos)

Escreva uma função `main` que **lê**

(a) o raio `rB` do projétil dos bixos;

(b) o vetor velocidade `(velXB, velYB)` de lançamento do projétil dos bixos;

(c) o raio `rV` do projétil dos veteramos;

(d) o vetor velocidade `(velXV, velYV)` de lançamento do projétil dos veteranos;

(e) o intervalo de tempo `dt` considerado entre as atualizações (passo da simulação).

Todos esses dados são do tipo real (`float`).

Suponha que $(10, 10)$ são as coordenadas iniciais do projétil dos bixos e que $(90, 20)$ são as coordenadas iniciais do projétil dos veteranos e que no momento inicial os projéteis não estão tocando o chão e que também não se encontram colididos. Tão logo os bixos realizam um lançamento, digamos em $\texttt{t} = 0$, os veteranos também realizam um lançamento no instante imediatamente seguinte $\texttt{t} + \texttt{dt}$ visando interceptar o projétil dos bixos. Suponha que as coordenadas e velocidades dos projéteis são atualizadas conforme descrito no item 3(c) desta questão.

Dadas essas condições, o seu programa **deverá simular a trajetória dos dois projéteis** considerando o passo de simulação `dt` dado e **deverá imprimir em cada instante da simulação**:

- as coordenadas `(xB,yB)` do protétil do bixos,
- as coordenadas `(xV,yV)` do projétil do veteranos, e
- a distância entre as superfícies dos projéteis (em caso de colisão, essa distância deve ser condiderada igual a zero).

A simulação deverá terminar se uma colisão entre os projéteis for detectada ou quando ambos os projéteis tocarem o chão. Note que um projétil, ao tocar o chão, não afunda no chão (sua coordenada vertical é tal que ele fica tangente ao chão) e **permanece parado até o final da simulação**. Um projétil pode colidir com outro que já esteja no chão. Ao final da simulação **o programa deve imprimir**:

- se houve ou não uma colisão, e
- a **menor distância** registrada **entre as superfícies dos projéteis** durante a simulação.

Para a leitura dos dados de entrada, suponha que é dada e utilize a função:

```
void LeiaDados(float *rB, float *velXB, float *velYB,
               float *rV, float *velXV, float *velYV, float *dt) ;
```

A sua função `main` deve usar **obrigatoriamente** as funções `distancia` do item 3(a), `colisao` do item 3(b) e `atualiza` do item 3(c).

```
/*
 * UMA POSSIVEL solucao, HA MUITAS outras aceitaveis
 */

int main()
{
  float rB;           /* raio do projetil dos bixos */
  float xB, yB;       /* coordenadas do centro do projetil dos bixos  */
  float velXB, velYB; /* vetor velocidade do projetil dos bixos */
  float rV;           /* raio do projetil dos veterano */
  float xV, yV;       /* coordenadas do centro do projetil dos veteranos  */
  float velXV, velYV; /* vetor velocidade do projetil dos veteranos */
  float t;            /* intante da simulacao */
  float dt;           /* passo da simulacao */
  float dist;         /* distancia entre os projeteis */
  float mindist;      /* menor distancia entre os projeteis */

  int colidiu ;       /* indica se houve colisao */


  LeiaDados(&rB, &velXB, &velYB, &rV, &velXV, &velYV, &dt);

  /* no instante t=0 os projeteis estao parados */
  t = 0;
  xB = 10;  yB = 10;
  xV = 90;  yV = 20;
  mindist = distancia(xB,yB,xV,yV) - (rB+rV);
  printf("t=%f   (%f,%f)   (%f,%f)   %f\n",
          t,      xB,yB,    xV,yV,   mindist) ;

  /* no instante t=dt o projetil dos bixos ja se movimentou */
  t = t+dt;

  /* atualize posicao e velocidade projetil dos bixos */
  atualiza(&xB,&yB,&velXB,&velYB,rB,dt) ;

  /* verifique colisao e se nao houve calcule distancia */
  colidiu = colisao(xB,yB,rB,xV,yV,rV) ;
  if (colidiu == 0)
    dist = distancia(xB,yB,xV,yV) - (rB+rV);
  else
    dist = 0;

  /* atualize menor distancia */
  if (dist < mindist)
    mindist = dist ;

  /* imprima posicoes e distancia */
  printf("t=%f   (%f,%f)   (%f,%f)   %f\n",
          t,      xB,yB,    xV,yV,   dist) ;
```

```
  while (colidiu==0 && (yB > rB || yV > rV))
    {
      /* daqui para a frente começa a simulação simultanea dos dois projéteis */
      t = t+dt ;

      /* atualize posicoes e velocidades */
      atualiza(&xB,&yB,&velXB,&velYB,rB,dt);
      atualiza(&xV,&yV,&velXV,&velYV,rV,dt);

      /* verifique colisao e se nao houve calcule distancia */
      colidiu = colisao(xB,yB,rB,xV,yV,rV) ;
      if (colidiu == 0)
        dist = distancia(xB,yB,xV,yV) - (rB+rV);
      else
        dist = 0;

      /* atualize menor distancia */
      if (dist < mindist)
        mindist = dist ;

      /* imprima posicoes e distancia */
      printf("t=%f   (%f,%f)   (%f,%f)   %f\n",
              t,      xB,yB,    xV,yV,   dist) ;
     }

  if (colidiu==1)
    printf("Houve colisão em  (%f,%f)   (%f,%f)\n", xB, yB, xV, yV ) ;
  else
    printf("Não houve colisão\n") ;

  printf("A menor distância registrada entre as superfícies dos projeteis"
         " foi %f\n", mindist) ;

  return 0 ;
}
```