MAC2166 – Introdução à Computação

Escola Politécnica

Segunda Prova – 19 de maio de 2015

Nome: _____

Assinatura: _____

Nº USP: _____ Turma: _____

Professor: _____

Instruções:

1. Não destaque as folhas deste caderno.
2. A prova contém 4 questões. Verifique antes de começar a prova se o seu caderno de questões está completo.
3. 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.
4. A prova pode ser feita a lápis. Cuidado com a legibilidade e, principalmente, com a TABULAÇÃO.
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 permitida a consulta a livros, apontamentos ou colegas.

DURAÇÃO DA PROVA: 2 horas

Questão	Valor	Nota
1	2,5	
2	2,5	
3	3,0	
4	2,0	
Total		

Questão 1 (valor: 2,5 pontos)

Escreva um programa na linguagem C que lê como entrada um número real *epsilon* > 0 e exibe como saída um número real que é uma aproximação de π calculada por meio da seguinte série infinita:

$$\frac{\pi}{6} = \frac{1}{2} + \left(\frac{1}{2}\right)\frac{1}{3}\left(\frac{1}{2}\right)^3 + \left(\frac{1}{2}\frac{3}{4}\right)\frac{1}{5}\left(\frac{1}{2}\right)^5 + \left(\frac{1}{2}\frac{3}{4}\frac{5}{6}\right)\frac{1}{7}\left(\frac{1}{2}\right)^7 + \left(\frac{1}{2}\frac{3}{4}\frac{5}{6}\frac{7}{8}\right)\frac{1}{9}\left(\frac{1}{2}\right)^9 + \cdots$$

Inclua na aproximação todos os termos da série até o primeiro termo de valor menor do que *epsilon*. Inclua também na soma esse último termo calculado.

Descubra o que muda de um termo para outro na soma e use essa informação para escrever sua função (ou seja, para calcular um termo da soma, você deve reaproveitar valores envolvidos no cálculo do termo anterior).

Solução 1

```c
#include <stdio.h>

int main()
{
    double soma, parte1, parte3, termo, epsilon;
    int denominador_parte2;

    printf("Digite o valor de epsilon: ");
    scanf("%lf", &epsilon);

    termo = soma = 0.5;
    parte1 = 1;
    denominador_parte2 = 1;
    parte3 = 0.5;

    while (termo >= epsilon)
    {
        denominador_parte2 += 2;
        parte1 *= (float)(denominador_parte2 - 2) / (denominador_parte2 - 1);
        parte3 /= 4.0;
        termo = parte1 * (1.0 / denominador_parte2) * parte3;

        soma += termo;
    }

    soma *= 6;
    printf("Aproximacao do pi: %f \n", soma);

    return 0;
}
```

Solução 2

```
#include <stdio.h>

int main()
{
    double soma, termo, epsilon;
    int denominador_parte2;

    printf("Digite o valor de epsilon: ");
    scanf("%lf", &epsilon);

    termo = soma = 0.5;

    denominador_parte2 = 1;

    while (termo >= epsilon)
    {
        termo = (termo * denominador_parte2) *
                 ((float)denominador_parte2 / (denominador_parte2 + 1));
        denominador_parte2 += 2;
        termo = termo / denominador_parte2 / 4.0;

        soma += termo;
    }

    soma *= 6;
    printf("Aproximacao do pi: %f \n", soma);

    return 0;
}
```

Questão 2 (valor: 2,5 pontos)

Para um número real $x \geq 0$, denotamos por $\lfloor x \rfloor$ seu *chão*, que é o maior inteiro menor ou igual a x. Por exemplo, temos $\lfloor 3 \rfloor = 3$ e $\lfloor 4.73 \rfloor = 4$.

Dado um número real $a_0 \geq 1$, podemos considerar a sequência de números a_0, a_1, a_2, ... tal que

$$a_{k+1} = \begin{cases} a_k/2 & \text{se } \lfloor a_k \rfloor \text{ é par,} \\ (3a_k+1)/4 & \text{se } \lfloor 3a_k+1 \rfloor \text{ é múltiplo de 4,} \\ (3a_k+1)/2 & \text{caso contrário,} \end{cases}$$

para $k \geq 0$.

Por exemplo, começando com $a_0 = 13$, obtemos a sequência

$$13, 10, 5, 4, 2, 1, 1 \ldots.$$

Observe que após a ocorrência do número 1, todos os demais números da sequência são iguais a 1. Começando com $a_0 = 5/2$, obtemos

$$5/2, 5/4, 19/16, 73/64, 283/256, 1105/1024, 4339/4096, \ldots.$$

Temos $4339/4096 \approx 1.05932$. Se você continuar a sequência, perceberá que os números vão ficando cada vez mais próximos de 1.

Chamamos a sequência $(a_k)_{k\geq 0}$ de *sequência de Collatz real* com *valor inicial* a_0.

(a – 1,5 pontos) Escreva uma função na linguagem C de protótipo

```
int collatz(double x, double eps);
```

que, dados números reais $x \geq 1$ e $eps > 0$, calcula a sequência de Collatz $(a_k)_{k\geq 0}$ de valor inicial $a_0 = x$ e devolve o MENOR n tal que $|a_n - 1| < eps$.

Por exemplo, se $x = 5/2$ e $eps = 0.06$, sua função deve devolver 6, já que $a_6 = 4339/4096 \approx 1.059$ e os valores anteriores são maiores ou iguais a $1105/1024 \approx 1.079$.

(b – 1 ponto) Você deve escrever um programa na linguagem C para calcular o maior valor de sua função do item (a) para o conjunto de números $x_r = 1 + r\delta$ com $0 \leq r \leq N$ inteiro, onde $\delta = (u-1)/N$.

Assim, seu programa deve ler números reais $u \geq 1$ e $eps > 0$ e um número inteiro $N \geq 1$ e imprimir na tela o número

$$\max\{\, \texttt{collatz}(x_r,\ eps) : 0 \leq r \leq N \text{ é inteiro} \,\}.$$

Você pode usar a função do item anterior mesmo que não a tenha escrito.

Solução

```
#include <stdio.h>

double absoluto(double x)
{
    if (x < 0)
        return -x;

    return x;
}

int collatz(double x, double eps)
{
    int n = 0;

    while (absoluto(x-1) >= eps)
    {
        n++;
        if (((int)x) % 2 == 0)
            x /= 2;
        else
            if (((int)(3*x+1)) % 4 == 0)
                x = (3*x + 1) / 4;
            else
                x = (3*x + 1) / 2;
    }

    return n;
}

int main()
{
    double u, eps, delta;
    int N, max, r, novo_n;

    printf("Digite os valores para u, epsilon e N: ");
    scanf("%lf %lf %d", &u, &eps, &N);

    delta = (u-1)/N;
    max = collatz(1, eps);  /* x0 = 1 + 0*delta = 1 */

    for (r = 1; r <= N; r++) {
        novo_n = collatz(1+r*delta, eps);
        if (novo_n > max)
            max = novo_n;
    }

    printf("max = %d\n", max);

    return 0;
}
```

Questão 3 (valor: 3 pontos)

Nesta questão você vai implementar uma função na linguagem C para a soma de números de ponto flutuante similar àquela que você implementou no EP2.

Neste problema vamos supor que a macro (= constante) `MAX` contenha um valor tal que uma variável do tipo `int` consiga comportar números de valor absoluto menor ou igual a $10 * \texttt{MAX}$. Nosso objetivo é **sempre** trabalhar com mantissas de valor absoluto menor ou igual a `MAX`.

Um número de ponto flutuante é representado por um par (m, e) de números inteiros, sendo m a mantissa e e o expoente. O par (m, e) representa o número

$$m \times 10^e.$$

Um número está *normalizado* se sua mantissa não possui zeros à direita. Por exemplo, o número $(12300, 5)$ não está normalizado, já o número $(123, 7)$ está normalizado.

Você deve escrever uma função de protótipo

```
int soma(int *res_m, int *res_e, int x_m, int x_e, int y_m, int y_e);
```

que recebe dois números de ponto flutuante normalizados (x_m, x_e) e (y_m, y_e) com $|x\_m|, |y\_m| \leq \texttt{MAX}$ e coloca em $*res\_m$ e $*res\_e$ a mantissa e o expoente do resultado NORMALIZADO da soma dos dois números recebidos. A função deve devolver ZERO se **ocorreu** perda de precisão durante a soma e um valor DIFERENTE DE ZERO **caso contrário**. LEMBRE-SE: devemos ter ao final $|*res\_m| \leq \texttt{MAX}$.

Para escrever sua função, lembre-se de que:

- Para fazer a soma, precisamos igualar os expoentes dos números (caso eles não sejam iguais), mas sempre com o cuidado de manter a maior quantidade possível de dígitos de precisão no resultado.
- Com os números com expoentes iguais, podemos somar suas mantissas e normalizar o resultado.
- Se ao final o valor absoluto da mantissa for maior que `MAX`, devemos dividí-la por 10 e aumentar o expoente para obter uma mantissa de valor absoluto menor ou igual a `MAX`.

Neste exercício, você **não precisa** se preocupar com a ocorrência de *overflow* no expoente de um número.

EXEMPLO: Suponha que $\texttt{MAX} = 999$. Vamos somar $a = (99, 5)$ com $b = (587, 3)$. Primeiro diminuímos o expoente de a (que é o número de maior expoente), obtendo $(990, 4)$; note que não podemos diminuir mais o expoente. Depois, aumentamos o expoente de b até obter expoente 4, obtendo $(58, 4)$ com perda de precisão. Depois, fazemos $990 + 58 = 1048 > 999$, logo temos que consertar a mantissa e perder precisão novamente. Assim, obtemos o resultado $(104, 5)$ com perda de precisão.

Solução

```
#include <stdio.h>
#define MAX 999

int soma(int *res_m, int *res_e, int x_m, int x_e, int y_m, int y_e)
{   int m_aux, e_aux, manteve_precisao = 1;

    /* guarda em x_m, x_e o operando de menor expoente */
    if (y_e < x_e )
    {   /* troca os valores de x_m, x_e e y_m, y_e */
        m_aux = x_m;
        e_aux = x_e;
        x_m = y_m;
        x_e = y_e;
        y_m = m_aux;
        y_e = e_aux;
    }

    /* Decrementa o expoente do maior numero ate que ele se iguale ao do menor,
       mas sem estourar a mantissa do maior numero  */
    while (y_e > x_e && y_m*10 <= MAX && y_m*10 >= -MAX)
    {
        y_m *= 10;
        y_e--;
    }

    /* Se y_m chegou ao numero maximo de digitos e os expoentes ainda continuam diferentes,
       entao e' preciso incrementar o expoente do menor numero cortando digitos de sua
       mantissa. Nesse processo, o menor numero pode chegar a zero. */
    while (y_e > x_e && x_m != 0)
    {
        x_m /= 10;
        x_e++;
        manteve_precisao = 0;
    }

    /* o resultado da soma ficara armazenado temporariamente em y_m, y_e */
    y_m += x_m;

    if (y_m != 0)  /* normaliza o resultado */
        while (y_m % 10  == 0)
        {
            y_m /= 10;
            y_e++;
        }

    if (y_m > MAX || y_m < -MAX)
    {  /* |y_m| > MAX, entao e' preciso cortar 1 digito da mantissa */
        y_m /= 10;
        y_e++;
        manteve_precisao = 0;
    }

/* armazena o resultado */
    *res_m = y_m;
    *res_e = y_e;

    return manteve_precisao;
}
```

Questão 4 (valor: 2 pontos)

Dizemos que o par de números (p, q) é um par de *primos gêmeos* se $q = p + 2$. Por exemplo, $(101, 103)$ é um par de primos gêmeos.

A função de protótipo `void devolvePrimosGemeos(int i, int *p1, int *p2);` devolve em `*p1` e `*p2` o i-ésimo par de primos gêmeos, onde $(3, 5)$ é o primeiro par de primos gêmeos, $(5, 7)$ é o segundo, $(11, 13)$ é o terceiro, e assim por diante.

A seguir, são apresentadas 4 diferentes implementações da função `devolvePrimosGemeos`. Em cada uma delas, você deve indicar se o código está correto ou incorreto. Ao lado de toda implementação que indicar como incorreta, escreva brevemente sobre o(s) erro(s) encontrado(s). Obs.: (i) O valor da questão será anulado caso todas as respostas forem marcadas como "Correto". (ii) Você deve supor que a função de protótipo `int ehPrimo(int num);` (que devolve 1 quando `num` é primo e 0 caso contrário) já existe.

(a) Correto ( ) Incorreto (X)

```
void devolvePrimosGemeos(int i, int *p1, int *p2) {
    int k = 0, p;
    for (p = 3; k < i; p++) {
        if (ehPrimo(p) && ehPrimo(p+2))
            k++;
    }
    p1 = p;                        <----- Após o laço, p está com 1 a mais do que deveria estar.
    p2 = p+2;                             Além disso, usa p1 e p2 quando deveria usar *p1 e *p2.
}
```

(b) Correto (X) Incorreto ( )

```
void devolvePrimosGemeos(int i, int *p1, int *p2) {
    int k = 0, p;
    for (p = 2; k < i; p++)
        if (ehPrimo(p) && ehPrimo(p+2))
            k++;
    *p1 = p-1;
    *p2 = p+1;
}
```

(c) Correto (X) Incorreto ( )

```
void devolvePrimosGemeos(int i, int *p1, int *p2) {
    int k = 1, p = 0;
    while (k <= i) {
        p++;
        if (ehPrimo(p) && ehPrimo(p+2))
            k++;
    }
    *p1 = p;
    *p2 = p+2;
}
```

(d) Correto ( ) Incorreto (X)

```
void devolvePrimosGemeos(int i, int *p1, int *p2) {
    int k = 0, p = 1;
    while (k <= i) {                    <----- O laço pára no i+1-ésimo par de primos gêmeos.
        p+=2;
        if (ehPrimo(p) && ehPrimo(p+2))
            k++;
    }
    *p1 = *p;                            <---- p não é uma variável do tipo ponteiro, portanto
    *p2 = (*p)+2;                              não faz sentido usar *p .
}
```