Министерство науки и образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
"Московский институт электронной техники"
(МИЭТ)

Отчет по лабораторной работе № 5

" Работа с числами с плавающей запятой"

Выполнили: студенты ПМ - 31

Алтухова Анна Валерьевна

Мартынова Мария Олеговна

2021 г.

Задание Л5. №1. Разработайте программу на языке C/C++, выполняющую вычисления над числами с плавающей запятой одинарной точности ($float$). Проверьте, что программа действительно работает с операндами одинарной точности, а не приводит к типу $float$ окончательный результат. Для частичной суммы гармонического ряда $S(N) = \sum_{i=1}^{N} \frac{1}{i} \in R$ найдите две её оценки: $S_d(N)$ — последовательно складывая члены, начиная от 1 и заканчивая N, и $S_a(N)$ — от N к 1. Сравните $S_d(N)$ и $S_a(N)$ для различных значений N: 10^3 , 10^6 , 10^9 . Объясните результат.

Листинг:

```
void task1()
{
    cout << "--------------Задание 1---------------" << endl;

    cout << "Результат для входных переменных типа float:" << endl;

    float Sd_float [3];
    float Sa_float [3];

    int n[3] = {1000,1000000,1000000000};

    for(int k = 0; k < 3; k++)
    {
        for(int i = 1; i <= n[k]; i++)
            Sd_float[k] += static_cast<float>(1)/i;
        for(int i = n[k]; i > 0; i--)
            Sa_float[k] += static_cast<float>(1)/i;
        cout << "n = " << n[k] << " : Sd = " << Sd_float[k] << ", Sa = " << Sa_float[k] << endl;
    }

    cout << "Результат для входных переменных типа double:" << endl;

    double Sd_double [3];
    double Sa_double [3];

    for(int k = 0; k < 3; k++)
    {
        for(int i = 1; i <= n[k]; i++)
            Sd_double[k] += static_cast<double>(1)/i;
        for(int i = n[k]; i > 0; i--)
            Sa_double[k] += static_cast<double>(1)/i;
        cout << "n = " << n[k] << " : Sd = " << Sd_double[k] << ", Sa = " << Sa_double[k] << endl;
    }

    cout << endl;
}
```

Вывод:

```
--------------Задание 1--------------
Результат для входных переменных типа float:
n = 1000 : Sd = 7.48548, Sa = 7.48547
n = 1000000 : Sd = 14.3574, Sa = 14.3927
n = 1000000000 : Sd = 15.4037, Sa = 18.8079
Результат для входных переменных типа double:
n = 1000 : Sd = 7.48547, Sa = 7.48547
n = 1000000 : Sd = 14.3927, Sa = 14.3927
n = 1000000000 : Sd = 21.3005, Sa = 21.3005
```

С увеличением количества членов различие между S_d и S_a одинарной точности возрастает. Это связано, что одинарная точность учитывает только 7-8 знаков после запятой. Когда мы суммируем от 1 до N, то прибавляем к большему числу меньшее и часть последних членов суммы попросту отбрасывается в результате округления. Таким образом, S_a является более точным результатом, потому что учитывает самые маленькие значения, суммируя их первыми. Для чисел с двойной точностью различие между S_d и S_a меньше, так как учитывается 16 знаков после запятой.

Задание Л5.№2. Разработайте программу на языке C/C++, рассчитывающую с заданной точностью ε сумму лейбницевского ряда:

1	$S = \sum_{i=2}^{\infty} (-1)^i \frac{i+1}{i^2-1}$

Листинг:

```
double i_sequence_term2(unsigned long long int i)
{
    return pow(-1,i)*((i+1)/(pow(i,2)-1));
}

double two_sum(double& t, double a, double b)
{
    double s = a+b; // сумма чисел a и b

    // вычисление погрешности нахождения суммы
    double bs = s-a;
    double as = s-bs;
    t = (b-bs)+(a-as);

    return s;
}
void task2()
{
    cout << "--------------Задание 2---------------" << endl;
    double eps = pow(10,-6);

    double sum1 = 0; // текущее значение суммы для первого алгоритма
    double t = 0; // текущее значение погрешности найденной суммы
    double t_i; // погрешность на i-ой итерации

    // Rump-Ogita-Oishi algorithm
    for(int i = 2; abs(i_sequence_term2(i+1)) > eps; i++)
    {
        sum1 = two_sum(t_i,sum1,i_sequence_term2(i));
        t += t_i;
    }

    cout <<"sum = " << sum1+t << ", eps = " << eps << endl;
    cout << endl;
}
```

Вывод:

```
--------------Задание 2---------------
sum = 0.693147, eps = 1e-06


Process returned 0 (0x0)   execution time : 0.563 s
Press any key to continue.
```

Задание Л5.№3. Бонус (+2 балла). Разработайте программу на языке C/C++, рассчитывающую с заданной точностью ε сумму обобщённого гармонического ряда $S = \sum_{i=1}^{\infty} \frac{1}{i^{\alpha}}$.

Листинг:

```
double i_sequence_term3(unsigned long long int i, double alpha)
{
    return 1.0/pow(i,alpha);
}

void task3()
{
    double eps = pow(10,-6);
    double alpha;
    cout<<"Введите значение alpha:"<<endl;
    cin>>alpha;
    if (alpha<2)
    {
        cout<<"Ряд расходится"<<endl;
        return ;
    }

    double sum1 = 0; // текущее значение суммы для первого алгоритма
    double a_i = i_sequence_term3(1,alpha);
    double t = 0; // текущее значение погрешности найденной суммы
    double t_i; // погрешность на i-ой итерации

    for(int i = 1; a_i + i_sequence_term3(i+1,alpha-1)/(alpha-1) > eps; i++)
    {
        sum1 = two_sum(t_i,sum1,i_sequence_term3(i,alpha));
        t += t_i;
        a_i = i_sequence_term3(i+1,alpha);
    }

    cout << "sum = " << sum1+t << ", eps = " << eps << endl;
    cout << endl;
}
```

Вывод:

```
--------------Задание 3---------------
Введите значение alpha:
4
sum = 1.08232, eps = 1e-06


Process returned 0 (0x0)   execution time : 3.045 s
Press any key to continue.
```

Задание Л5.№4. Рассчитайте (используя FPU) значение выражения от числа x с плавающей запятой ($double$):

1	$\frac{7{,}9 - \sin(x)}{1 - \cos(x)}$

Листинг:

```
void task4()
{
  cout << "--------------Задание 4---------------" << endl;
  double x;
  cout << "Введите значение x: ";
  cin >> x;

  double result;
  const double _7_9 = 7.9;
  asm volatile (
    "   fldl    %[x]"           "\n"    // st(0) = x
    "   fsin"                "\n"    // st(0) = sin(x)
    "   fsubrl  %[_7_9]"        "\n"    // st(0) = 7.9 - sin(x)
    "   fld1"                "\n"    // st(0) = 1.0 | st(1) = 7.9 - sin(x)
    "   fldl    %[x]"           "\n"    // st(0) = x | st(1) = 1.0 | st(2) = 7.9 - sin(x)
    "   fcos"                "\n"    // st(0) = cos(x) | st(1) = 1.0 | st(2) = 7.9 - sin(x)
    "   fsubrp"              "\n"    // st(0) = 1.0 - cos(x) | st(1) = 7.9 - sin(x)
    "   fdivrp"              "\n"    // st(0) = (7.9 - sin(x)) / (1.0 - cos(x))
    "   fstpl   %[result]"      "\n"    // запись результата, стек пуст
    : [result]"=m"(result)          // выходные операнды
    : [x]"m"(x), [_7_9]"m"(_7_9)        // входные операнды
  );
  cout << "Решение с помощью FPU: y= " << std::setprecision(6) << result << endl;
  cout<<"Проверка на C++: y="<<(7.9-sin(x))/(1.0-cos(x))<<endl;
}
```

Вывод:

```
--------------Задание 4---------------
Введите значение x: 5
Решение с помощью FPU: y= 12.367
Проверка на C++: y=12.367

Process returned 0 (0x0)   execution time : 2.198 s
Press any key to continue.
```

Задание Л5.№5. Рассчитайте (используя AVX) для массивов ($x0$, ...$x3$) и ($y0$, ...$y3$) из четырёх чисел с плавающей запятой ($double$), аналогичный массив ($z0$, ...$z3$), где $zi = (xi + yi) \cdot (xi - yi)$. Выделение памяти под x, y, z и заполнение массивов x, y может быть выполнено на C/C++.

Листинг:

```
void task5()
{
  cout << "--------------Задание 5---------------" << endl;
  double x[4], y[4], z[4];
  cout << "Введите массив x (через enter): ";
  cin >> x[0] >> x[1] >> x[2] >> x[3];
  cout << "Введите y (через enter): ";
  cin >> y[0] >> y[1] >> y[2] >> y[3];

  // Реализация через AVX для double
  asm volatile (
    "   vmovupd (%[x]), %%ymm0"      "\n"    // ymm0 = массив x
    "   vmulpd  (%[y]), %%ymm0, %%ymm0" "\n"    // ymm0 *= массив y
    "   vmovupd %%ymm0, (%[z])"      "\n"    // запись результата
    :                           // выходные операнды
    : [x]"r"(x), [y]"r"(y), [z]"r"(z)       // входные операнды
    : "%ymm0", "memory"             // разрушаемые регистры (memory - память, cc - флаги)
  );
  cout << "z = [" << std::setprecision(16) << z[0] << ", " << z[1] << ", " << z[2] << ", " << z[3] <<"]"<< endl;
}
```

Вывод:

```
--------------Задание 5---------------
Введите массив x (через enter): 1.1
2.2
3.3
4.4
Введите y (через enter): 5.5
6.6
7.7
8.8
z = [6.05, 14.52, 25.41, 38.72]

Process returned 0 (0x0)   execution time : 15.476 s
Press any key to continue.
```

Задание Л5.№6. Разработайте программу, целиком написанную на ассемблере, вычисляющую (вызывая функции libc) выражение:

1	$4{,}6 \cdot x^y - 1{,}3$

Листинг:

```
task6_asm:

    sub $16, %rsp

    #define mem_x (%rsp)
    #define mem_y 8(%rsp)
    movq %xmm0, mem_x
    movq %xmm1, mem_y

    // 4.6*x^y-1.3

    fldl mem_y          // загрузка y
    fldl mem_x          // загрузка x
    fyl2x               // y*log_2(x) <=> log_2(x^y)
    fstpl mem_x         // выгрузка значения из стека

    fld1                // загрузка единицы
    fldl mem_x          // загрузка log_2(x^y)
    fprem               // получение дробной части log_2(x^y)
    f2xm1               // 2^{log_2(x^y)}-1
    fld1                // загрузка единицы
    fadd                // 2^{log_2(x^y)}
    fxch                // теперь на вершине стека лежит единица
    fldl mem_x          // загрузка log_2(x^y)
    fxch                // теперь на вершине стека лежит единица
    fscale              // 1*2^[log_2(x^y)]
    fxch                // теперь на вершине стека лежит log_2(x^y)
    fstpl mem_x         // выгрузка x из стека
    fmul                // 2^[log_2(x^y)]*2^{log_2(x^y)} = 2^(log_2(x^y)) = x^y

    fldl a              // загрузка a
    fmul                // a*x^y
    fldl b              // загрузка b
    fsubr               // a*x^y-b

    fstpl mem_x
    movq mem_x, %xmm0

    #undef mem_x
    #undef mem_y
    add $16, %rsp
    ret
```

```
main:

    sub $8, %rsp

    mov x, %rcx
    mov y, %rdx
    movq %rcx, %xmm0
    movq %rdx, %xmm1

    sub $32, %rsp
    call task6_asm
    add $32, %rsp

    lea s_asm(%rip), %rcx
    movq %xmm0, %rdx
    sub $32, %rsp
    call printf
    add $32, %rsp

    mov x, %rcx
    mov y, %rdx
    movq %rcx, %xmm0
    movq %rdx, %xmm1

    sub $32, %rsp
    call task6_cpp
    add $32, %rsp

    lea s_cpp(%rip), %rcx
    movq %rax, %rdx
    sub $32, %rsp
    call printf
    add $32, %rsp


    add $8, %rsp
    xor %eax, %eax
    ret
```

Вывод:

```
Assembler: z = 272.925999
CPP: z = 272.925999

Process returned 0 (0x0)   execution time : 0.143 s
Press any key to continue.
```