Домашнее задание № 4

по курсу «Архитектура ЭВМ и язык ассемблера»
для студентов групп 231, 232, 233 БПИ ФКН ВШЭ

Тема: Сборка многомодульных программ. Вычисление корней уравнений и определенных интегралов.
Язык программирования: Си, ассемблер NASM.

1. Постановка задачи

С заданной абсолютной точностью ε вычислить площадь плоской фигуры, ограниченной тремя кривыми, уравнения которых $y = f_1(x)$, $y = f_2(x)$ и $y = f_3(x)$ либо заранее определены вариантом задания, либо задаются в текстовом виде на этапе сборки программы. Во втором случае необходимо разработать две программы: основную — для вычисления площади и вспомогательную — для построения исполняемого кода, вычисляющего значения функций.

При решении задачи необходимо следующее.

- С некоторой точностью ε_1 вычислить абсциссы точек пересечения кривых, используя предусмотренный вариантом задания метод приближенного решения уравнения $F(x) = 0$.
  - В вариантах задания, где уравнения кривых фиксированы, отрезки, где программа будет искать точки пересечения, и где применим используемый метод, следует определить самостоятельно. Выбор отрезка должен иметь строгое математическое обоснование.
  - В вариантах задания, где уравнения задаются в текстовом виде, отрезки для поиска пересечения задаются вместе с уравнениями.
- Представить площадь заданной фигуры как алгебраическую сумму определенных интегралов и вычислить эти интегралы с некоторой точностью ε_2 по квадратурной формуле, предусмотренной вариантом задания.
- Величины ε_1 и ε_2 подобрать самостоятельно так, чтобы гарантировалось вычисление площади фигуры с точностью $\varepsilon = 0.001$. Выбор значений ε_1 и ε_2 должен иметь строгое математическое обоснование.

2. Варианты задания

Вариант определяется следующими параметрами: (1) способом задания кривых и их набором для вариантов 1-10, (2) методом приближенного решения уравнений, (3) квадратурными формулами.

I. Уравнения кривых

1.	$f_1 = 2^x + 1,$	$f_2 = x^5,$	$f_3 = \frac{1-x}{3}$	
2.	$f_1 = 3\left(\frac{0.5}{x+1} + 1\right),$	$f_2 = 2.5x - 9.5,$	$f_3 = {}^5\!/_x,$	$x > 0$
3.	$f_1 = e^{-x} + 3,$	$f_2 = 2x - 2,$	$f_3 = {}^1\!/_x$	
4.	$f_1 = e^x + 2,$	$f_2 = -{}^1\!/_x,$	$f_3 = \frac{-2(x+1)}{3}$	
5.	$f_1 = 0.35x^2 - 0.95x + 2.7,$	$f_2 = 3x + 1,$	$f_3 = \frac{1}{x+2}$	
6.	$f_1 = 0.6x + 3,$	$f_2 = (x-2)^3 - 1,$	$f_3 = {}^3\!/_x,$	$x > 0$
7.	$f_1 = \ln x,$	$f_2 = -2x + 14,$	$f_3 = \frac{1}{(2-x)} + 6$	
8.	$f_1 = e^x + 2,$	$f_2 = -2x + 8,$	$f_3 = {}^{-5}\!/_x$	
9.	$f_1 = \frac{3}{(x-1)^2+1},$	$f_2 = \sqrt{x + 0.5},$	$f_3 = e^{-x}$	
10.	$f_1 = 1 + \frac{4}{x^2+1},$	$f_2 = x^3,$	$f_3 = 2^{-x}$	

11. *(вариант повышенной сложности, +2 или +3 задачи)* Плоская фигура задается во время сборки программы в виде текстового описания. См. Приложение А.

II. Методы приближенного решения уравнений

1. Метод деления отрезка пополам.
2. Метод хорд (секущих).
3. Метод касательных (Ньютона).
4. Комбинированный метод (хорд и касательных).
5. *(усложнённый вариант, +1 задача)* Реализуется два различных метода решения уравнений. Для каждого метода строится исполняемый файл, выбор метода происходит на этапе сборки программы (см. раздел 3.2.).

III. Квадратурные формулы

1. Формула прямоугольников.
2. Формула трапеций.
3. Формула Симпсона (парабол).

3. Требования к программе

3.1. Вычисление площади

1. Функции, вычисляющие значения f_1, f_2, f_3 и их производных (в случае необходимости производных) реализуются на языке ассемблера с соглашением вызова cdecl. Все остальные функции программы реализуются на языке Си.

2. Вычисление с точностью ε_1 корня x уравнения $f(x) = g(x)$ на отрезке $[a, b]$ должно быть реализовано в отдельной Си-функции `root(f, g, a, b, eps1)`. Если используется метод касательных или комбинированный метод, то у `root` должно быть еще два параметра функционального типа, позволяющие вызывать производные функций f и g. Никаких других параметров у функции `root` быть не должно. Вызывать вычисляющие значения или производные функции по их глобальным именам запрещено.

3. Вычисление с точностью ε_2 величины определенного интеграла от функции $f(x)$ на отрезке $[a, b]$ должно быть реализовано в отдельной функции `integral(f, a, b, eps2)` на языке Си.

4. Си-функции `root` и `integral` должны быть предварительно протестированы. Основная программа должна предоставлять возможности тестирования, активируемые опцией командной строки (см. ниже).

5. Программа должна быть снабжена поясняющими комментариями в объёме, достаточном для её понимания. Все глобальные и статические переменные должны быть документированы в комментариях.

3.2. Сборка и тестирование программы

1. Сборка программы должна осуществляться при помощи утилиты `make`. Соответствующий файл должен явно или неявно описывать зависимости между всеми целями сборки.

2. Должны быть определены цели `all` и `clean`, первая из которых полностью собирает программу, а вторая — удаляет все промежуточные файлы (в частности, объектные модули).

3. Должна быть определена цель `test`, которая запускает программу в режиме тестирования функций `root` и `integral` с подобранными вручную значениями параметров. Для каждой из них необходимо запустить как минимум три теста. В тестах должны фигурировать как минимум три **новые** математические функции, которые можно реализовать на языке Си или ассемблера. Использовать целевые функции (из варианта) для тестирования **нельзя**.

4. Сдаваемый проект в GitHub Classroom должен включать в себя `Makefile`.

5. *(для усложнённого варианта II.5)* Выбор конкретного метода решения уравнений должен управляться определёнными символами на этапе препроцессирования и передаваться через ключ `-D`.

3.3. Поддерживаемые опции командной строки

Программа должна поддерживать следующие ключи командной строки в длинной и короткой форме:

1. Опции `--help` и `-h`, которые выводят на печать все допустимые ключи командной строки.

2. Опции `--root` и `-r`, которые печатают абсциссы точек пересечения кривых.

3. Опции `--iterations` и `-i`, которые печатают число итераций, потребовавшихся на приближенное решение уравнений при поиске точек пересечения.

4. Опции `--test-root` и `-R`, которые позволяют протестировать функцию `root`. Фактические параметры вызова `root` задаются единственным параметром этой опции в виде `F1:F2:A:B:E:R`, где `F1`, `F2` — номера используемых функций, `A`, `B`, `E` — значения параметров `a`, `b`, `eps1` функции `root`, `R` — правильный ответ (вычисленный аналитически). Программа должна вызывать функцию `root` с указанными параметрами, сравнивать результат с правильным ответом и выводить на экран полученный результат, абсолютную и относительную ошибку.

```
$ ./integral --test-root 1:2:3.0:5.0:0.0001:4.0
4.00001 0.00001 0.0000025
```

5. Опции `--test-integral` и `-I`, которые позволяют протестировать функцию `integral`. Фактические параметры вызова `integral` задаются единственным параметром этой опции в виде `F:A:B:E:R`, где `F` — номера используемой функций, `A`, `B`, `E` — значения параметров `a`, `b`, `eps2` функции `integral`, `R` — правильный ответ (вычисленный аналитически). Программа должна вызывать функцию `integral` с указанными параметрами, сравнивать результат с правильным ответом и выводить на экран полученный результат, абсолютную и относительную ошибку.

```
$ ./integral --test-integral 1:3.0:5.0:0.0001:8.0
8.00001 0.00001 0.000005
```

6. Программа, запущенная без входных параметров, должна выводить ответ на поставленную задачу (площадь фигуры).

4. Методические указания

4.1. Численные методы

Поиск корня

1. Для поиска точек пересечения на некотором отрезке $[a, b]$ двух функций $f(x)$ и $g(x)$ рассмотрим функцию $F(x) = f(x) - g(x)$. Если $F(x)$ имеет разные знаки на концах отрезка, а её производная не меняет знак, то $F(x)$ имеет ровно один корень на этом отрезке (являющийся абсциссой точки пересечения $f(x)$ и $g(x)$). Если дополнительно известно, что и вторая производная не меняет знак, то корень можно найти с помощью метода хорд или метода касательных. Границы отрезка $[a, b]$, удовлетворяющие этим условиям, требуется найти аналитически (вручную).

2. В *методе деления отрезка* пополам определяется средняя точка с отрезка $[a, b]$ и из двух отрезков $[a, c]$ и $[c, b]$ выбирается тот, на концах которого функция $F(x)$ имеет разные знаки. К выбранному отрезку применяется

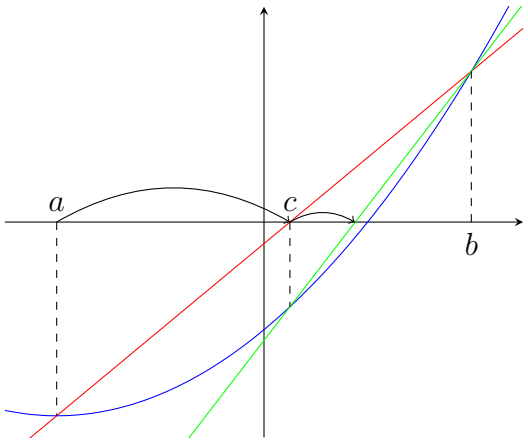

Рис. 1: Метод хорд

та же процедура. Процесс деления отрезков прекращается, когда длина очередного отрезка станет меньше требуемой точности ε; за корень уравнения можно принять любую точку этого отрезка.

3. В *методе хорд* концы $(a, F(a))$ и $(b, F(b))$ кривой $y = F(x)$ соединяются прямой линией и определяется точка пересечения этой линии с осью абсцисс:

$$c = \frac{aF(b) - bF(a)}{F(b) - F(a)}.$$

Далее выбирается отрезок $[c, b]$ в случае, если $F'(x)F''(x) > 0$ (приближение к искомому корню слева), или отрезок $[a, c]$ в противном случае (приближение справа), и к нему применяется та же процедура.

Если приближение «идет» слева, то на очередном шаге надо сравнить величины $F(c)$ и $F(c+\varepsilon)$: если они одного знака, то процесс продолжается, иначе на отрезке $[c, c + \varepsilon]$ имеется корень, и потому процесс завершается. При приближении справа надо проверять знаки $F(c - \varepsilon)$ и $F(c)$.

4. В *методе касательных* проводится касательная к кривой $y = F(x)$ в точке $(b, F(b))$, если $F'(x)F''(x) > 0$, или в точке $(a, F(a))$ в противном случае, и определяется точка с пересечения этой касательной с осью абсцисс:

$$c = d - \frac{F(d)}{F'(d)},$$

где $d = b$, если $F'(x)F''(x) > 0$ (приближение справа), и $d = a$ в противном случае (приближение слева). Далее проводится касательная к кривой в точке $(c, F(c))$ и процедура повторяется. Критерий завершения процесса приближения к корню аналогичен критерию завершения метода хорд.

5. В *комбинированном методе* одновременно применяется метод хорд и метод касательных, в связи с чем приближение к корню происходит с двух сторон. Критерий завершения процесса — длина очередного отрезка меньше ε.

6. При использовании метода хорд, метода касательных или комбинированного метода выбор отрезка на очередной итерации зависит от того, имеют первая и вторая производная $F(x)$ одинаковые знаки $(F'(x)F''(x) > 0)$,

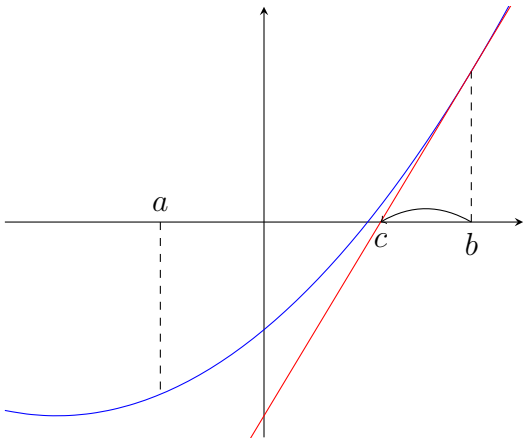

Рис. 2: Метод касательных

либо разные ($F'(x)F''(x) < 0$). Функция `root` должна самостоятельно распознавать, какой из двух случаев, имеет место при текущем обращении к ней. Это можно сделать проверкой следующих двух условий:

- функция возрастает или убывает;
- график функции расположен выше хорды, соединяющей концы графика, или ниже.

Поскольку производные $F'(x)$ и $F''(x)$ на отрезке $[a, b]$ не меняют знак, для проверки первого условия достаточно сравнить $F(a)$ с нулём (при $F(a) < 0$ функция возрастает). Для проверки же второго условия надо сравнить в какой-то внутренней точке отрезка значения функции и хорды; например, если взять среднюю точку $(a + b)/2$ отрезка, то соотношение

$$F\left(\frac{a+b}{2}\right) > \frac{F(a) + F(b)}{2}$$

означает, что график функции расположен выше хорды. Если функция возрастает и ее график расположен ниже хорды, или если функция убывает и ее график расположен выше хорды, то первая и вторая производные имеют одинаковый знак, иначе — разный.

Вычисление интеграла

1. Квадратурные формулы для приближенного вычисления интеграла I от функции $F(x)$ на отрезке $[a, b]$ имеют следующий вид (n — число разбиений отрезка $[a, b]$):

   - Формула прямоугольников

     $$I \approx I_n = h(F_0 + F_1 + \ldots + F_{n-1}), \quad F_i = F(a + (i + 0.5)h), \quad h = \frac{b-a}{n}.$$

   - Формула трапеций

     $$I \approx I_n = h(0.5F_0 + F_1 + \ldots + F_{n-1} + 0.5F_n), \quad F_i = F(a + ih), \quad h = \frac{b-a}{n}.$$

- Формула Симпсона (n — чётное)

$$I \approx I_n = \frac{h}{3}(F_0 + 4F_1 + 2F_2 + 4F_3 + \ldots + 4F_{n-3} + 2F_{n-2} + 4F_{n-1} + F_n),$$

$$F_i = F(a + ih), \quad h = \frac{b-a}{n}.$$

2. Для обеспечения требуемой точности ε при приближенном вычислении интеграла I по квадратурной формуле нужно подобрать соответствующее число n разбиений отрезка интегрирования. Известны формулы, выражающие n через ε, но в эти формулы входят производные подынтегральной функции, что неудобно на практике. Поэтому для достижения требуемой точности обычно используется следующий метод: берется некоторое начальное число разбиений n_0 (например, 10 или 20) и последовательно вычисляются значения I_n при n, равном $2n_0$, $4n_0$, $8n_0$ и т.д. Известно правило Рунге:

$$|I - I_n| \approx p|I_n - I_{2n}|$$

(для формул прямоугольников и трапеций $p = 1/3$, для формулы Симпсона $p = 1/15$). Согласно этому правилу, когда на очередном шаге величина $p|I_n - I_{2n}|$ окажется меньше ε, в качестве приближенного значения для I можно взять I_{2n}.

3. При реализации функции `integral` следует учитывать, что в формулах трапеций и Симпсона в сумму I_{2n} входят значения F_i, вычисленные ранее для суммы I_n, поэтому их не следует перевычислять заново. Необходимо также избежать хранения значений с предыдущей итерации в массиве.

4.2. Реализация

1. В функциях `root` и `integral` используются параметры-функции (`f`, `g` и др.). Рекомендуется завести имя для типа функции от одного вещественного аргумента и использовать указатель на такой тип в параметрах функций `root` и `integral`.

```
typedef double afunc(double);
...
double root(afunc *f, afunc *g, double a, double b, double eps1)
```

2. Необходимо помнить о том, что сборка ассемблерного кода на платформе Windows отличается от сборки для UNIX-окружения. В командной строке `nasm` целевой формат должен быть указан как `-f win32` вместо `-f elf32`, а имена функций должны иметь ведущее подчёркивание: функция, которая вызывается из Си-кода по имени `foo`, должна на выходе ассемблера иметь имя `_foo`[1]. В GitHub Classroom должен быть загружен вариант программы, предназначенный для сборки и запуска в UNIX-окружении.

3. Для разбора опций командной строки можно воспользоваться функцией `getopt_long`.

4. Аналитическую работу с кривыми упрощают онлайн-сервисы, например, `https://www.wolframalpha.com/`.

[1]Новые версии NASM позволяют автоматически проставить ведущее подчёркивание с помощью опции командной строки `--prefix _` либо с помощью директивы `%pragma win32 prefix _` в ассемблерном файле.

5. Сдача задания

Код программы загружается в GitHub Classroom (ссылку на задание предоставит преподаватель) и затем сдаётся лично преподавателю в формате индивидуальной беседы.

Список литературы

1. Трифонов Н. П., Пильщиков В. Н. Задания практикума на ЭВМ (1 курс). Методическая разработка (составители). — М.: ВМК МГУ, 2001.
2. Ильин В. А., Садовничий В. А., Сендов Бл. Х. Математический анализ. Т.1 — М.: Наука, 1985.

Приложение А. Вариант I.11 (повышенной сложности)

Плоская фигура задается во время сборки программы в виде текстового описания. Фигура ограничивается графиками трех функций, отрезок, на котором эти графики пересекаются, заранее определен и указан в этом же текстовом файле. Используется следующий формат текстового файла.

Первая строка содержит два вещественных числа, разделенные пробелами. Числа задают границы отрезка, на котором следует искать точки пересечения кривых. Следующие три строки описывают функции, используя для этого польскую обратную запись. Каждая строка содержит разделенные пробелами термы: переменная величина `x`, вещественное число, операция.

Переменная задается символом `x`. Вещественные числа задаются в формате, поддерживаемом функцией `scanf`. Также должны поддерживаться константные значения `e` и `pi`.

Поддерживаемые операции:

1. бинарные: `+, -, *, /`;
2. унарные: `sin, cos, tan, ctg`.

Листинг 1: Пример входного файла

```
0.0 4.0
2 x 4 / tan -
x
0.2 pi *
```

Имя текстового файла не фиксируется и должно задаваться при сборке в виде переменной окружения:

```
$ SPEC_FILE=in.txt make
```

Для обработки текстового файла потребуется разработать и реализовать вспомогательную Си-программу, которая по заданному файлу с описанием строит ассемблерный листинг с функциями, вычисляющими заданные выражения. Для вычисления необходимо использовать команды сопроцессора x87. Для приведенного выше файла будет получен код, приведенный на Листинге 2.

Польская запись считывается и по ней строится промежуточное представление: дерево, описывающее выражение над переменной x и вещественными числами. При необходимости вычисляются производные, которые также представляются в виде деревьев. Следует учитывать, что слишком высокие деревья не могут быть вычислены на стеке регистров x87 без предварительного преобразования, уменьшающего их высоту.

После того, как был получен ассемблерный листинг, начинается сборка основной программы, вычисляющей площадь фигуры. Ассемблерный листинг транслируется nasm, полученный объектный модуль компонуется с модулем, полученным от компилятора.

В `Makefile` должна быть описана полная процедура сборки, с учетом всех зависимостей: предварительного построения вспомогательной программы и генерации ассемблерного листинга.

При решении усложненного задания к показателям по контестам дополнительно прибавляются: две задачи в случае, если вычисление

производных не требуется; три задачи, если производится вычисление производных.

Листинг 2: Пример построенного кода

```
section .data
    const1 dq 2.0
    const2 dq 4.0
    const3 dq 0.2

section .text
f1:
    push ebp
    mov ebp, esp
    fld qword [const1]
    fld qword [ebp + 8]
    fld qword [const2]
    fdivp
    fptan
    fxch
    fstp st1
    fsubp
    pop ebp
    ret

f2:
    push ebp
    mov ebp, esp
    fld qword [ebp + 8]
    pop ebp
    ret

f3:
    push ebp
    mov ebp, esp
    fld qword [const3]
    fldpi
    fmulp
    pop ebp
    ret
```