Отчёт ИДЗ-3 (ВАРИАНТ 27)

Ставцев М.А.

БПИ-217

4 балла

27. Разработать программу интегрирования функции $y = a + b * x^{-2}$ (задаётся двумя числами a,b) в заданном диапазоне (задаётся так же) методом Симпсона (точность вычислений $= 0.0001$).

Теория и определения:

По определения метода Симпсона, определённый интеграл с пределами интегрирования от $\boldsymbol{a}$ до $\boldsymbol{b}$ находится по следующей формуле:

$$\widetilde{I} = \sum_{i=1}^{N+1} \frac{x_i - x_{i+1}}{6}\left(f(x_i) + 4f\left(\frac{a+b}{2}\right) + f(x_{i+1})\right)$$

- *Процесс вычисления n – количества участков интегрирования*
  По условиям задачи точность вычисления должна равняться 0.0001
  При этом точность вычисления интеграла методом Симпсона оценивается следующей формулой:

$\left|\frac{f^{iv}(c)(b-a)^5}{180n^4}\right|$, где $|f^{iv}(c)|$ - максимальное значение модуля четвёртой производной функции $y = f(x) = a + b * x^{-2}$

Выпишем данные производные:

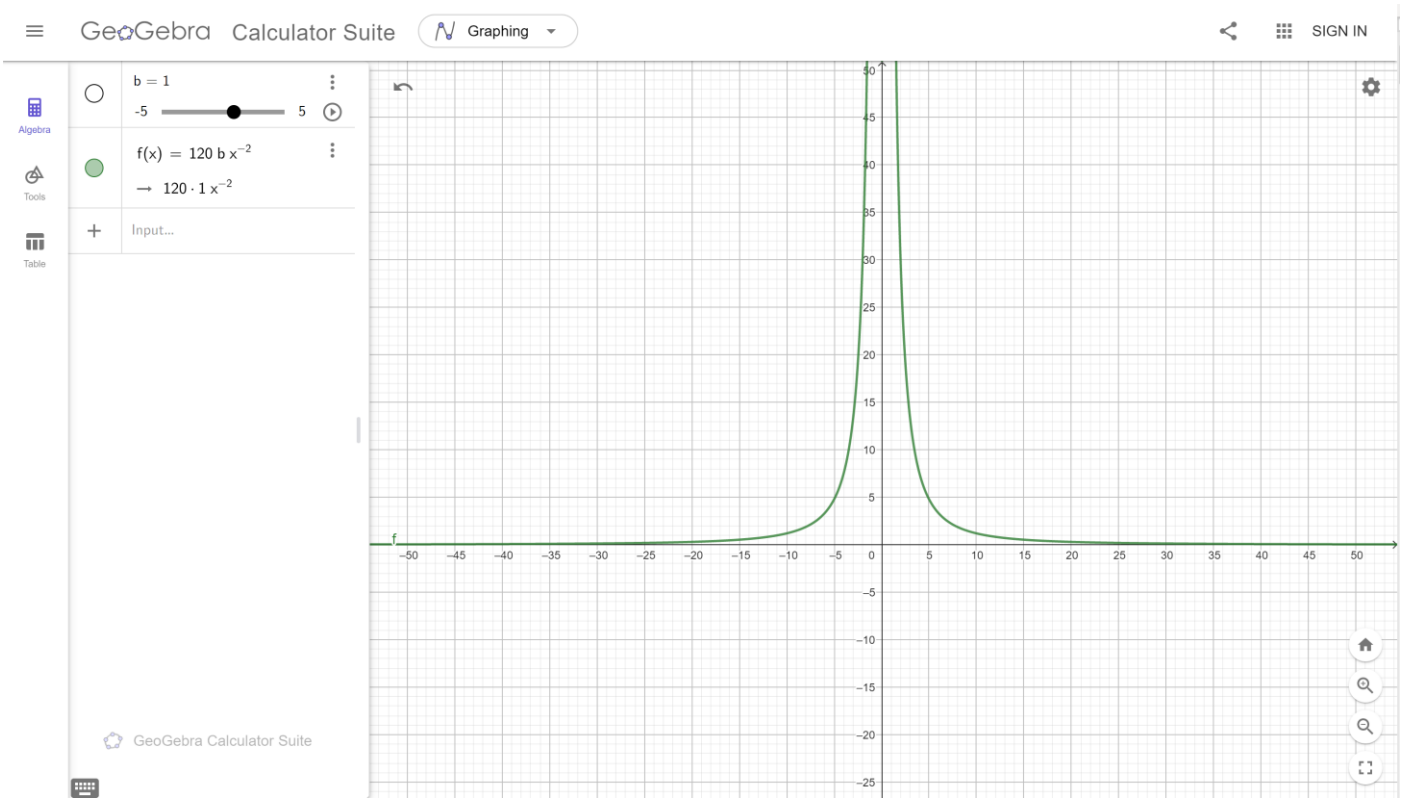
$f'(x) = -2bx^{-3}$

$f''(x) = 6bx^{-4}$

$f'''(x) = -24bx^{-5}$

$f^{IV}(x) = 120bx^{-6}$

Построим график функции $f^{IV}(x) = 120bx^{-6}$ в GeoGebra:



Можем видеть, что она стремится к бесконечности при $x \to 0$

В зависимости от параметра b её ветви направлены вверх или вниз.

На основе данного графика мы можем сделать вывод, что максимальное значение четвертая производная исходной функции будет принимать либо в точке $f^{IV}(a)$, либо в точке $f^{IV}(b)$

$$\text{то есть } \max = \max(|f^{IV}(a)|, |f^{IV}(b)|)$$

Теперь из ограничения на точность вычисления найдём n:

$$\left|\frac{\max*(b-a)^5}{180n^4}\right| < 0.0001$$

Возведём обе части в квадрат для снятия модуля и найдём ограничение на n

$n > \sqrt[8]{\frac{max^2*(b-a)^{10}}{0.000324}}$ - именно по этой формуле в функции calc_n вычисляется количество участков интегрирования, при округлении полученного значения вверх до целого числа функцией $ceil()$;

- Решением на C являются следующие файлы:
  Важно: запуск программы всегда необходимо осуществлять с 2 параметрами командной строки: seed и file name, в независимости от планируемого типа ввода данных. Если не будет 2 параметров, то приложение закончит свою работу с кодом ошибки 1. Можно, но не обязательно, добавить ещё один аргумент командной строки – количество повторений основных вычислений, что увеличит время выполнения работы команды. Если данный параметр не будет предоставлен, то все вычисления выполняться 1 раз.
  При компиляции ассемблерного кода необходимо указать флаг *-lm,* так как использовалась библиотека math.h

  - *main.c* – содержит основную функции *main*, *timespecDiff* для подсчёта затраченного времени и функцию f – которая возвращает значению функции $f(x) = a + b * x^{-2}$ в зависимости от введённых значений a и b

    Ввод и вывод данных:

    Ввиду специфики задачи программа может поддерживать 3 типа ввода данных – консоль, файл, рандом
    - Консоль – пользователь ввод два предела интегрирования значения a и b
    - Файл – значения a и b считываются из файла, который пользователь указал в качестве аргументов входной строки

    Ограничения и особенности ввода данных:

    1. Предел интегрирования a должен быть меньше b. Это считается некорректными входными данными, программа выводит сообщение об ошибке и завершается с кодом 1.
    2. Если один из пределов равен нулю, то это считается корректными входными данными, но интеграл от такой функции расходится. Программа выводит

соответствующее сообщение в консоль и завершает работу с кодом 0.

3. Если предел интегрирования $a < 0$ и $b > 0$, то это считается корректными входными данными, но интеграл от такой функции расходится. Программа выводит соответствующее сообщение в консоль и завершает работу с кодом 0.

- Рандом – значения a и b генерируются по следующему правилу:

1. Сначала определяется положительными или отрицательными будут оба предела интегрирования. Это происходит за счёт взятия рандомного значения $rand()$ по модулю 2 : 0 – положительные, 1 – отрицательные.
2. Сначала случайно генерируется предел b. Далее генерируется предел a, как значение $rand()$ взятое по модулю b, гарантирует, что $a < b$; и плюс ограничение 0.0001 – что предотвращает возможность того, что a станет равно 0.

$$a = rand() \ \% b + 0.0001$$

Вывод значения интеграла:

- Консоль – при условии консольного ввода или рандома
- Консоль + файл output.txt – если ввод был через файл

- *getData.c* – содержит функцию *getData*, которая считывает два предела интегрирования в зависимости от переданных в неё параметров (консольный ввод, ввод из файла, случайная генерация);
- *getRandomRange.c* – содержит функцию *getRandomRange*, которая генерирует два предела интегрирования, согласно правилу описанному выше. Результат генерации выводится в файл random_gen.txt

- *simpsonIntegral.c* – вычисления интеграла метод Симпсона в зависимости от параметров a, b, n

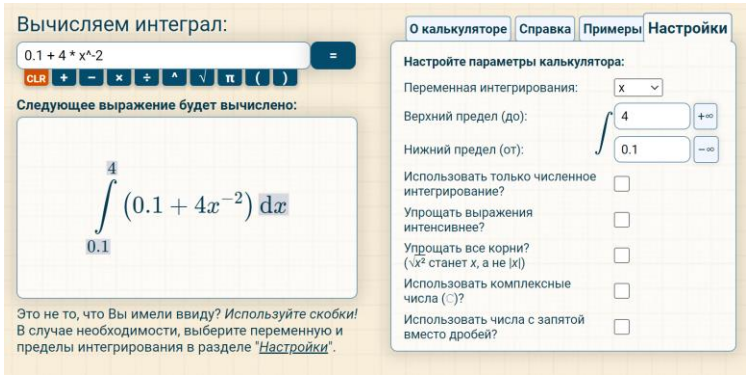
  Интервал от a до b разбивается на n участков длины $width = \frac{b-a}{n}$

  Находятся значения x_1 и x_2 – границы участков интегрирования, и далее происходят вычисления по формуле, представленной выше.
- *calc_n.c* – вычисление n согласно алгоритму описанному выше
- *header.h* – подключения необходимых библиотек

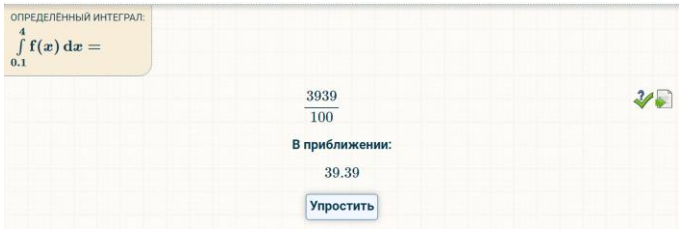
- Тестовое покрытие программы на C

  Для проверки результатов тестирования на корректность будем пользоваться сайтом integral-calculator.ru, который подсчитывает значения определённых интегралов.

  - TEST_1 $a = 0.1; b = 4$

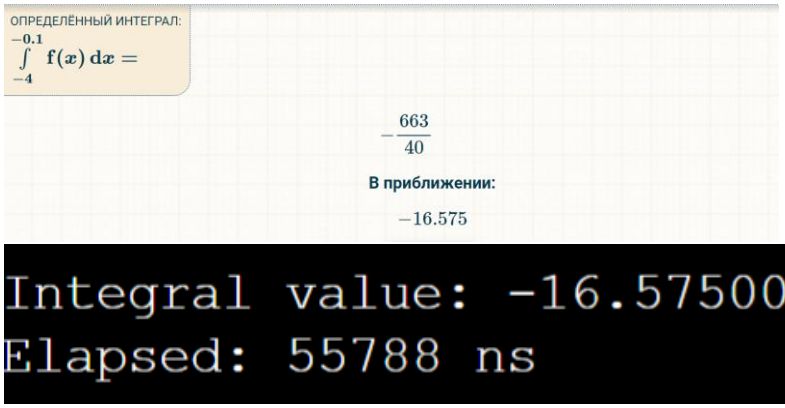
    

    Далее будут показаны только результаты подсчёта интеграла, без интерфейса ввода интеграла на сайте.

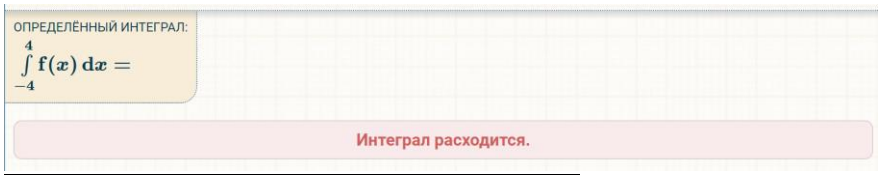

    ```
    Integral value: 39.390000
    Elapsed: 98092 ns
    ```

- TEST_2 $a = -4; b = -0.1$



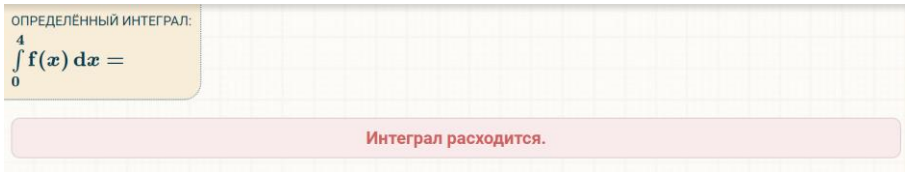

```
Integral value: -16.575000
Elapsed: 55788 ns
```

- TEST_3 $a = -4; b = 4$



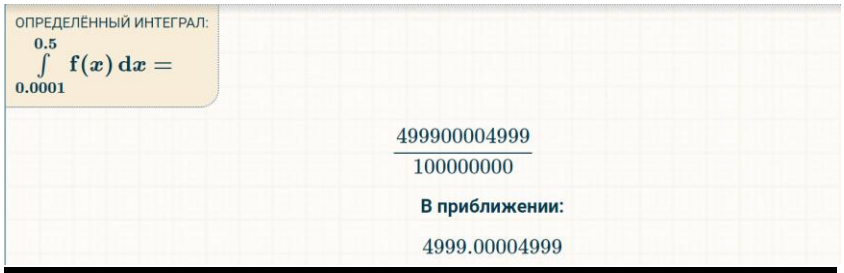

```
Input A and B:   -4 4
The integral diverges!
```

- TEST_4 $a = 0; b = 4$




```
Input A and B:   0 4
The integral diverges!
```

- TEST_5 $a = 0.0001; b = 0.5$

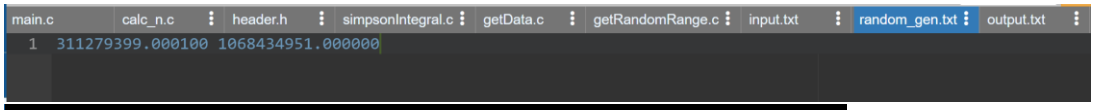


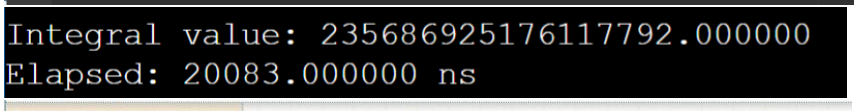

```
Integral value: 4999.000050
Elapsed: 11.220720 ms
```

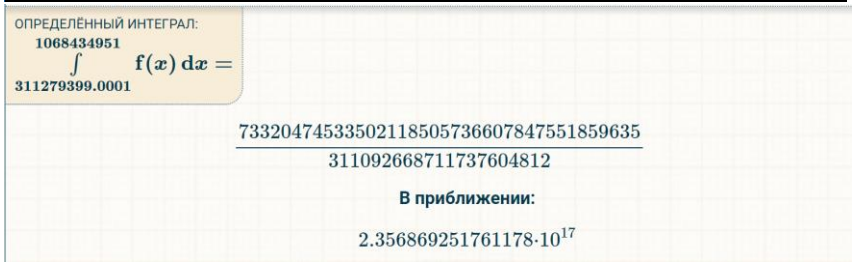
- TEST_6 Random Input

  Сгенерированные параметры (хранятся в файле random_gen.txt). Генерация происходит на основе $seed = 187456$, 1 параметра командной строки.

  $$a = 311279399.000100;\ b = 1068434951$$

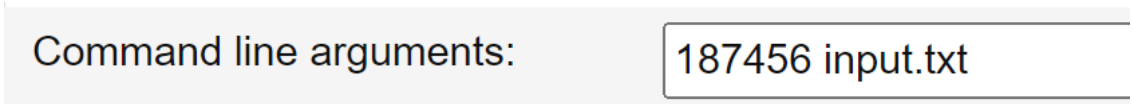


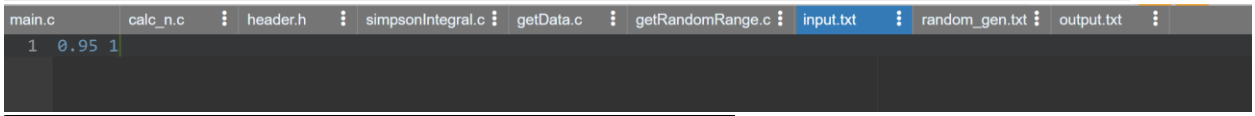


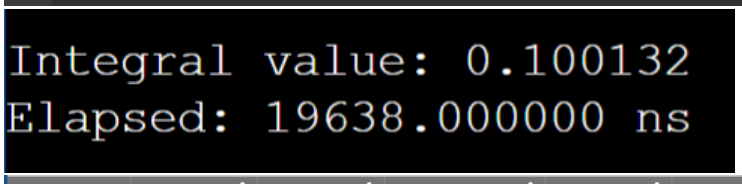


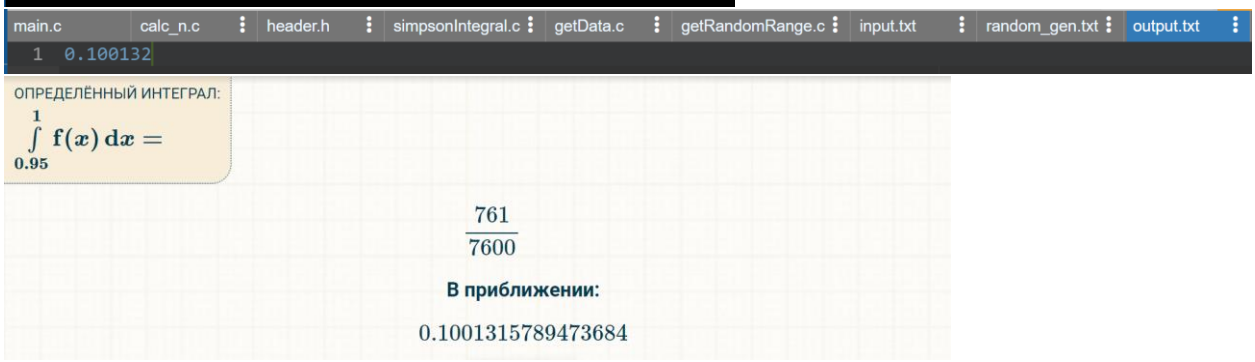
- TEST_7 File Input

  Пределы интегрирования внесены в файл input.txt – файл, являющийся 2 аргументом командной строки. $a = 0.95\ b = 1$









- Получение ассемблерных файлов программы, а также получения исполняемых .exe файлов показано далее на скриншотах:

```
mastavtsev@mastavtsev-VirtualBox:~$ gcc -masm=intel \
-fno-asynchronous-unwind-tables \
-fno-jump-tables \
-fno-stack-protector \
-fno-exceptions \
./main.c \
-S -o ./main.s
mastavtsev@mastavtsev-VirtualBox:~$ gcc -masm=intel \
-fno-asynchronous-unwind-tables \
-fno-jump-tables \
-fno-stack-protector \
-fno-exceptions \
./simpsonIntegral.c \
-S -o ./simpsonIntegral.s
```

```
mastavtsev@mastavtsev-VirtualBox:~$ gcc -masm=intel \
-fno-asynchronous-unwind-tables \
-fno-jump-tables \
-fno-stack-protector \
-fno-exceptions \
./calc_n.c \
-S -o ./calc_n.s
mastavtsev@mastavtsev-VirtualBox:~$ gcc -masm=intel \
-fno-asynchronous-unwind-tables \
-fno-jump-tables \
-fno-stack-protector \
-fno-exceptions \
./getRandomRange.c \
-S -o ./getRandomRange.s
```

```
mastavtsev@mastavtsev-VirtualBox:~$ gcc -masm=intel \
-fno-asynchronous-unwind-tables \
-fno-jump-tables \
-fno-stack-protector \
-fno-exceptions \
./getData.c \
-S -o ./getData.s
```

Дизассемблирование исходных файлов с ключами компиляции

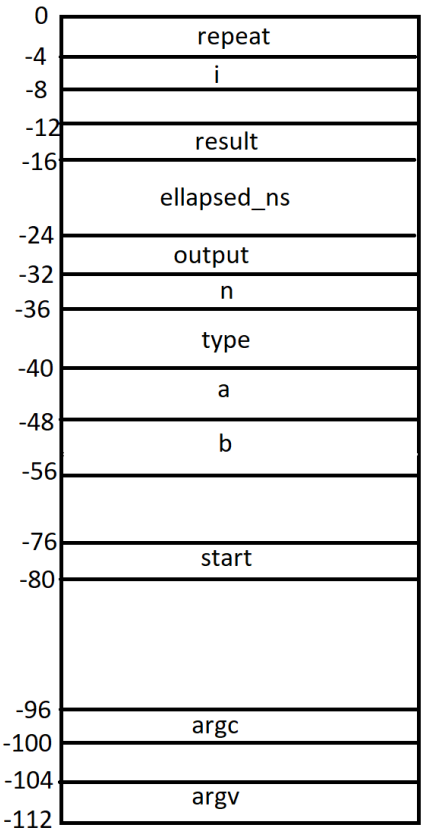
```
mastavtsev@mastavtsev-VirtualBox:~$ gcc ./main.c -c -o main.o
mastavtsev@mastavtsev-VirtualBox:~$ gcc ./getData.c -c -o getData.o
mastavtsev@mastavtsev-VirtualBox:~$ gcc ./calc_n.c -c -o calc_n.o
mastavtsev@mastavtsev-VirtualBox:~$ gcc ./simpsonIntegral.c -c -o simpsonIntegra
l.o
mastavtsev@mastavtsev-VirtualBox:~$ gcc ./getRandomRange.c -c -o getRandomRange.
o
```

Получение объектных файлов из С - версии программы и получение исполняемого .exe файла

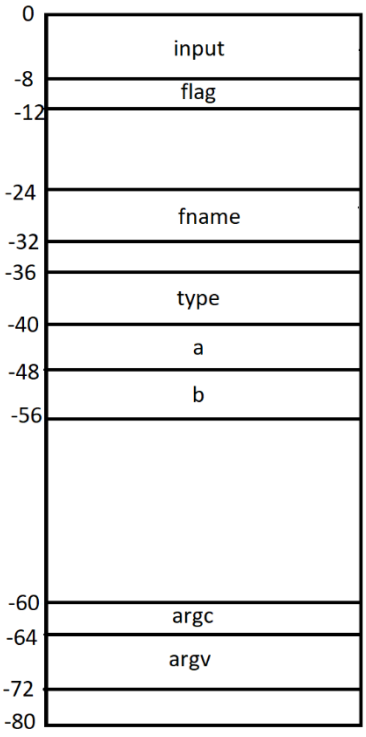
```
mastavtsev@mastavtsev-VirtualBox:~$ gcc ./main.s -c -o main.o
mastavtsev@mastavtsev-VirtualBox:~$ gcc ./getData.s -c -o getData.o
mastavtsev@mastavtsev-VirtualBox:~$ gcc ./getRandomRange.s -c -o getRandomRange.
o
mastavtsev@mastavtsev-VirtualBox:~$ gcc ./calc_n.s -c -o calc_n.o
mastavtsev@mastavtsev-VirtualBox:~$ gcc ./simpsonIntegral.s -c -o simpsonIntegra
l.o
```

Получение объектных файлов из Assembly - версии программы и получение исполняемого .exe файла

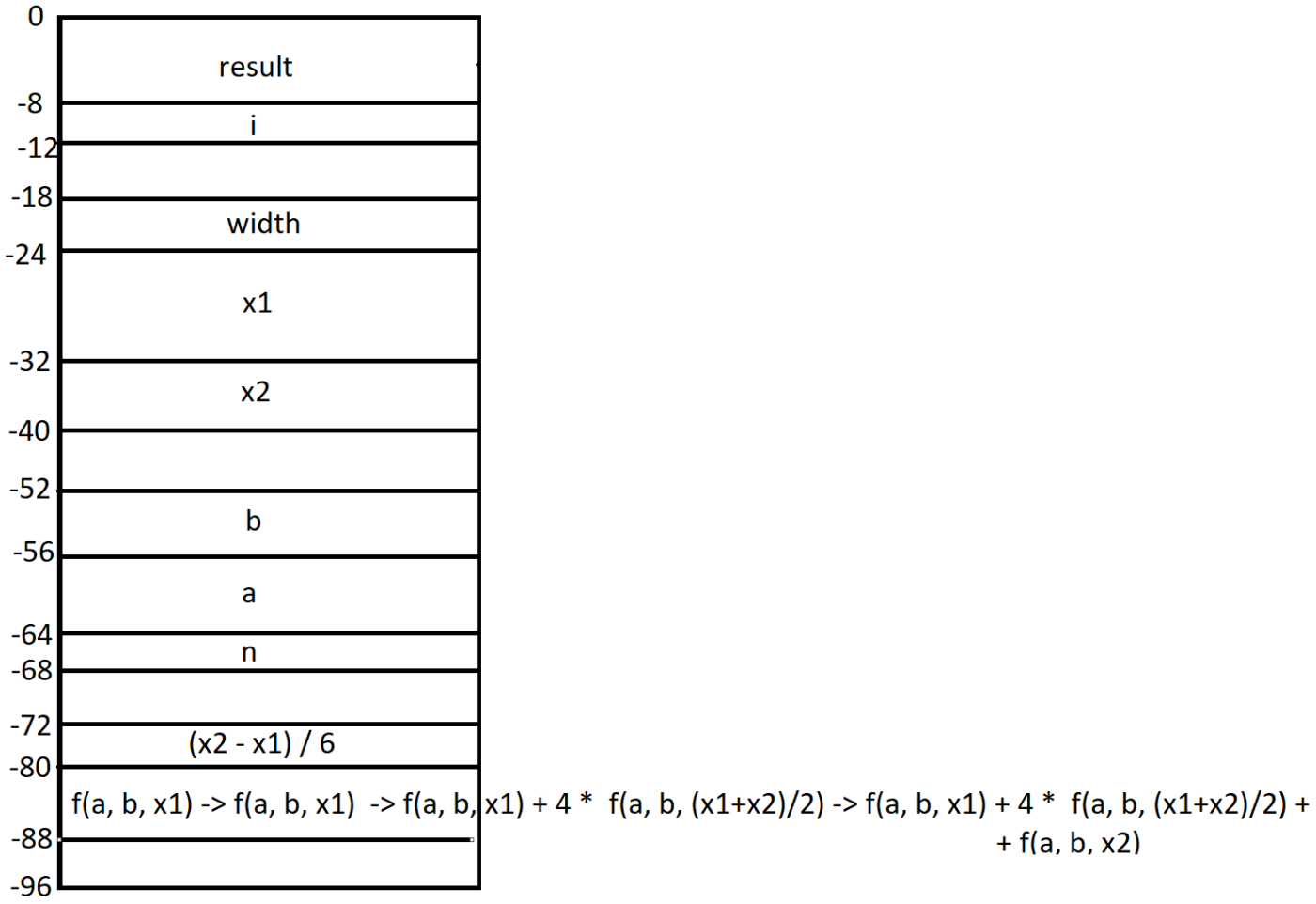
- 5 файлов на ассемблере сопровождены комментариями к кодам программ. В результате работы над кодом были построены следующие схемы стеков функций:



Стек функции $main$

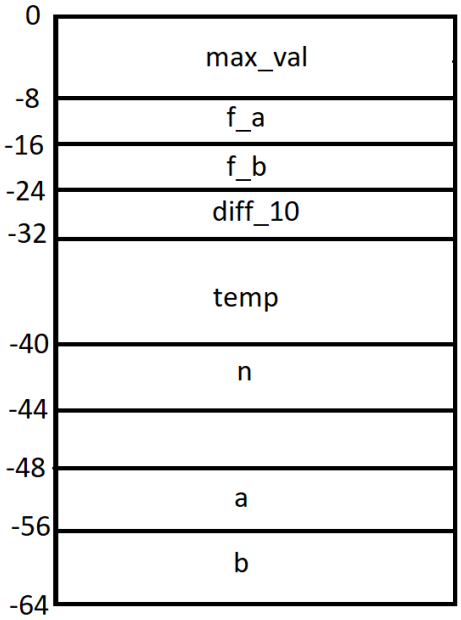


Стек функции $getData$

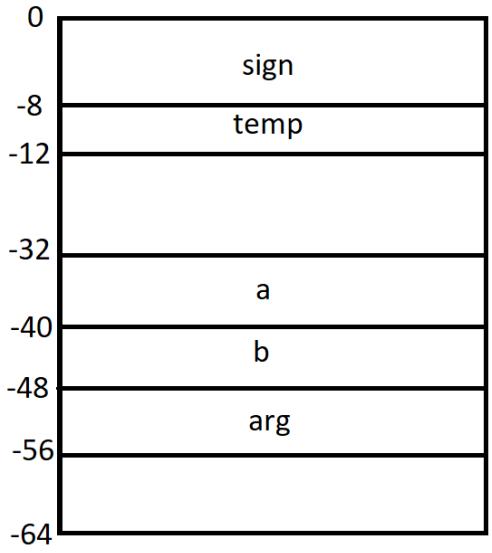


Стек функции $simpsonIntegral$

(стрелками показаны изменения значений хранимых в ячейке во время работы программы)



Стек функции $calc\_n$



Стек функции $getRandomRange$

- Сравнение тестовых прогонов будет приведено ниже для трёх программ: С программы, ассемблерной и изменённой ассемблерно.

5 баллов

- Все функции в программе принимают определённые параметры. Ниже приведены их сигнатуры:

```
double f(double a, double b, double x);
int calc_n(double a, double b);
double simpsonIntegral(double a, double b, int n);
void getData(int* type, double* a, double* b, int* argc, char **argv);
void getRandomRange(double* a, double* b, char* arg);
double f_d4(double x, double b);
double f(double a, double b, double x);
int main(int argc, char **argv);
```

- Внутри функций используются локальные переменны, анализ стеков для некоторых функций приведён выше.
- В ассемблерной программе присутствуют комментарии при передаче параметров из функции в функцию, а также комментарии о получении возвращаемых значений.
- В ассемблерная программа содержит комментарии о связи имён переменных и используемых регистрах.

6 баллов

- Замена использования стека в функциях на использование регистров.
  - main

    В функции main можно оптимизировать использование стека переменными a и b . Заменим использование переменной a на использование регистра $xmm3$; b на использование регистра $xmm4$

```
267 .L19: # тело цикла for (i = 0, i < repeat, i++)
268
269     movsd   xmm0, QWORD PTR -56[rbp]    # xmm0 = b
270     mov rax, QWORD PTR -48[rbp]         # rax = a
271     movapd  xmm1, xmm0                  # xmm1 = xmm0 == b
272     movq    xmm0, xmm3                   # xmm0 = xmm3 == a
273     call    calc_n@PLT                  # Вызов функции для подсчёта количества участков интегрирования
274
275     mov DWORD PTR -36[rbp], eax         # Результат работы calc_n - переменная n, кладётся на стек
276
```

    Оставим использование стека для переменной a при считывании переменной из консоли, а также при передаче a как параметра в функцию $simpsonIntegral$. Аналогично оставим без изменений взаимодействие со стеком для переменной b.

```
267 .L19: # тело цикла for (i = 0, i < repeat, i++)
268
269     movsd   xmm0, QWORD PTR -56[rbp]    # xmm0 = b
270     mov rax, QWORD PTR -48[rbp]         # rax = a
271     movapd  xmm1, xmm0                  # xmm1 = xmm0 == b
272     movq    xmm0, xmm3                   # xmm0 = xmm3 == a
273     call    calc_n@PLT                  # Вызов функции для подсчёта количества участков интегрирования
274
275     mov DWORD PTR -36[rbp], eax         # Результат работы calc_n - переменная n, кладётся на стек
276
```

    При сравнении затраченного времени с тестом 1 ($a = 0.1\ b = 4$) получаем уменьшение времени работы на 46000 ns, т.е. примерно в два раза.

```
Integral value: 39.390000
Elapsed: 51444.000000 ns
```

  - simpsonIntegral

    Заменим использование стека переменной $width$ на использование регистра $xmm5$.

    Заменим также использование стека переменной n на использование регистра $r15b$.

На аналогичном тесте 1 ($a = 0.1\ b = 4$) можем также видеть уменьшение работы программы за счёт данной оптимизации.

```
Integral value: 39.390000
Elapsed: 47043.000000 ns
```

- Сравнение размеров объектных и исполняемых .exe файлов программ. Объектными файлами являются: $main.o$, $getData.o$, $calc\_n.o$, $getRandomRange.o$, $simpsonIntegral.o$
  Важно: создание исполняемых .exe файлов из assembly кода необходимо производить с флагом -lm для работы библиотеки math.h

```
mastavtsev@mastavtsev-VirtualBox:~/ASM_OBJ$ gcc ./simpsonIntegral.o ./calc_n.o ./getRandomRange.o ./getData.o main.o -lm -o foo-asm.exe
```

Тип	Размер объектных файлов	Размер .exe файла
C code	13.4 Кб	17 Кб
ASM	11.8 Кб	16.9 Кб
AMS-MOD	11.9 Кб	16.9 Кб

C code – исходный код программы на C

ASM – код программы на ассемблере, до модификации

ASM-MOD – код программы на ассемблере, после модификации

<u>7 баллов</u>

- Описание возможности ввода входных данных приведено выше. Пользователь программы сам выбирает удобный ему тип ввода.
- Программа проверяет число входных аргументов, их должно быть не меньше двух, иначе будет выведено сообщение об ошибке и завершение работы программы с кодом 1. Допускается 3 параметр, число зацикливании выполнения основных вычислений. Однако, не советуется его использовать так как вычисления происходят за достаточно большое промежуток времени.
  Проверка на открытие файла происходит за счёт сравнения переменной для открытия входного (input) с NULL. (Регистра rax в assembly).
- Программа реализована за счёт 5 единиц компиляции, которые были перечислены выше.

- Приведены .txt файлы тестов TEST_1 – TEST_5 и TEST_7

8 баллов

- Генерация случайных значений границ интервала – a и b проходить в функции $getRandomRange$, генерация происходит на основе $seed$ введённого пользователем в командной строке.
- Тип ввода определяется пользователем из консоли. Пользователю выводятся в консоль 3 доступных опция для ввода данных, а он уже вводит номер, предпочитаемый опции в консоль.
  Тип ввода не определяется из командной строки для того, чтобы иметь какое-то подобие пользовательского интерфейса.
- В программе реализованы замеры времени за счёт библиотеки $time.h$ и функции $timespecDiff$
- Касательно зацикливания выполнения основных вычислении написано выше. Такая опция подразумевается, количество циклов необходимо указывать третьим параметром в командной строке. Однако, так делать не рекомендуется, так как время вычисления достаточное для его оценки и сравнения с другими версиями программы.
- Результаты тестов для различных версий программы приведены ниже.

9 баллов

- Сравнение трёх версий программы по памяти приведено выше.
- Сравнение по производительности на тестах TEST_1 – TEST_7
  Во всех текстах ввод из консоли.

Тест	C code	Assembly	Assembly MOD	Ответ
$a = 0.1; b = 4$	77186 нс	48119 нс	37653 нс	39.39
$a = -4; b = -0.1$	38182 нс	26625 нс	18 497 нс	-16.575
$a = -4; b = 4$	None	None	None	Diverges!
$a = 0; b = 4$	None	None	None	Diverges!
a=$0.0001; b = 0.5$	8.90339 мс	4.56464 мс	4.4671 мс	4999.00005
$a = 0.95; b = 1$	16936 нс	16817 нс	16245 нс	0.100132