Міністерство освіти і науки України  
НТУУ «КПІ ім. І. Сікорського»

Кафедра автоматизації проектування енергетичних процесів і систем

Лабораторна робота №1

«СТРУКТУРИ»

Виконав: студент II курсу ТЕФ, групи TI-92

Сопронюк Ю. А.

Перевірив: Касьянов А. С.

Дата виконання: 20.10.2020

Київ-2020

**Варіант №11**

ЗАВДАННЯ

* Визначте структуру, в якій містилися б тільки числові значення функції f(x) (п.2), параметри a, b, значення кроку dxта граничних значень аргументу функції x1***,*** x2 (п.1)**.**
* Розробіть програму, що має працювати при різних значеннях параметрів функції, граничних значень x1***,*** x2 і кроку аргументу функції.
* Створіть функцію ReadPar(), призначення якої зчитування з клавіатури за допомогою функції scanf()значень параметрів функції та параметрів dx, x1***,*** x2.
* Знайдіть значення функції f(x)для всіх точок діапазону [x1,x2], зокрема для випадків, коли dx після можливих розрахунків може перевищувати точне значення на 0,1%.
* Створіть функцію Tab(), що виводить на екран розраховані значення функції f(x) та значення аргументу x з такою кількістю значущих цифр, як наведено у таблиці (п.3). При створенні таблиці слід застосувати символи псевдографіки.
* Для розрахованих значень функцій f(x) обчисліть параметри S1, S2, наведені в п.6. Створіть масив дійсних чисел rnd[i] такого ж розміру, що й масив значень функції f(x). Значення масиву дійсних чисел повинні бути випадковими в діапазоні [Smin, Smax], де Sminє меншим, а Smaxє більшим з поміж S1 таS2. Крок випадкових значень має складати δ\*Sсер, Sсер=(Smin+Smax)/2 **(**δ визначено вп.5).
* Виведіть на екран значення S1 таS2 та створеного масиву випадкових чисел в табличному режимі.
* Визначте лінійну функцію перетворення розрахованих значень функції f(x)в цілі числа діапазону [0, A] (п.5). Створіть функціюPrint(), яка б виводила перетворені значення функції на екран у вигляді вертикальної залежності функції  за допомогою символу, код якого описано в п. 4.

1. Значення параметрів:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| 2 | 4 | -1.2 | 3.8 | 0.25 |

1. Визначення функції:
2. Вигляд таблиці:

|  |  |
| --- | --- |
|  |  |
| 0.25 | 1.23 |
|  |  |

1. Код символу 0xB2 {▓}.
2. Параметри: A=70, =0.15.
3. S1– середнє значення серед парних за номером точок масиву функції , S2– середнє для квадратів значень функції серед розрахованих точок.
4. Що таке мови типу Асемблера?
5. Що таке імперативна парадигма програмування?

**ВИКОНАНАННЯ РОБОТИ**

**Код програми**

***main.c***

#include "functions.h"

#include <stdlib.h>

#include <time.h>

int main(void)

{

  struct funcData values;

  double s1, s2;

  double \*rnd;

  int \*linear;

  srand(time(0));

  ReadPar(&values);

  CalcFunc(&values);

  Tab(&values);

  s1 = FindS1(&values);

  s2 = FindS2(&values);

  rnd = GenerateRndArr(&values, s1, s2);

  TabRestData(values.size, rnd, s1, s2);

  linear = GenerateLinearArr(&values);

  Print(&values, linear);

  free(rnd);

  free(linear);

  return 0;

}

***functions.h***

#ifndef FUNCTIONS\_H\_INCLUDED

#define FUNCTIONS\_H\_INCLUDED

#define ARR\_BOUND 100

/\*

oxB2 не виводився (використання %lc та setlocale не допомогло, потік виводу автоматично закривався, або

виводився не той символ, тому взяв інший символ зі знаенням від 0 до 127 як було вказано в методичних вказівках

\*/

#define CHARACTER 0x7C

struct funcData

{

  int a;

  int b;

  float dx;

  double x1;

  double x2;

  int size;

  double funcValues[ARR\_BOUND];

};

void ReadPar(struct funcData \*data);

void CalcFunc(struct funcData \*data);

void Tab(struct funcData \*data);

double \*GenerateRndArr(struct funcData \*data, double s1, double s2);

double FindS1(struct funcData \*data);

double FindS2(struct funcData \*data);

void TabRestData(int arrSize, double \*arr, double s1, double s2);

int \*GenerateLinearArr(struct funcData \*data);

void Print(struct funcData \*data, int \*arr);

#endif

***functions.c***

#include "functions.h"

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

void ReadPar(struct funcData \*data)

{

  printf("Enter a: \n");

  scanf("%d", &data->a);

  printf("Enter b: \n");

  scanf("%d", &data->b);

  printf("Enter x1: \n");

  scanf("%lf", &data->x1);

  printf("Enter x2: \n");

  scanf("%lf", &data->x2);

  printf("Enter dx: \n");

  scanf("%f", &data->dx);

}

void CalcFunc(struct funcData \*data)

{

  double x = data->x1;

  int counter = 0;

  while (x <= data->x2)

  {

    if (x <= 2)

    {

      data->funcValues[counter++] = data->a \* pow(x, 2) - data->b \* x;

    }

    else

    {

      data->funcValues[counter++] = (x - data->a) / (x - data->b);

    }

    x += data->dx;

  }

  data->size = counter;

}

void Tab(struct funcData \*data)

{

  double x = data->x1;

  //заголовки x та f(x)

  printf("\n%-10lc%-8s%-8lc%-10s%-7lc\n", CHARACTER, "X", CHARACTER, "f(x)", CHARACTER);

  //вивід самої таблиці

  for (int i = 0; i < data->size; i++)

  {

    for (int i = 0; i < 37; i++)

    {

      printf("%lc", CHARACTER);

    }

    printf("\n");

    printf("%-7lc%-11.2f%-7lc%-11.2f%-7lc\n", CHARACTER, x, CHARACTER, data->funcValues[i], CHARACTER);

    x += data->dx;

  }

  printf("\n");

}

double FindS1(struct funcData \*data)

{

  double s1 = 0;

  int s1Counter = 0;

  //знаходження парних за номером (не за індексом масиву) точок масиву функції

  for (int i = 1; i <= data->size; i++)

  {

    if (i % 2 == 0)

    {

      s1 += data->funcValues[i - 1];

      s1Counter++;

    }

  }

  return (s1 / s1Counter); //знаходження s1

}

double FindS2(struct funcData \*data)

{

  double s2 = 0;

  //сума квадратів значень функції f(x)

  for (int i = 0; i < data->size; i++)

  {

    s2 += pow(data->funcValues[i], 2);

  }

  return (s2 / data->size); //знаходження s2

}

double \*GenerateRndArr(struct funcData \*data, double s1, double s2)

{

  double beta = 0.15;

  double \*rnd = (double \*)malloc(data->size \* sizeof(double));

  if (rnd == NULL) exit(EXIT\_FAILURE);

  double sMin = s1;

  double sMax = s2;

  double sAvg = (sMax + sMin) / 2;

  double step = beta \* sAvg;

  // міняємо місцем мінімальне і максимальне за необхідності (при min > max)

  if (sMin > sMax)

  {

    sMin = sMax + sMin;

    sMax = sMin - sMax;

    sMin = sMin - sMax;

  }

  rnd[0] = ((sMax - sMin) \* ((double)rand() / RAND\_MAX)) + sMin;

  for (int i = 1; i < data->size; i++)

  {

    double minRand = rnd[i - 1] - step;

    double maxRand = rnd[i - 1] + step;

    // генеруємо випадкове значення на проміжку

    double generatedValue = (rand() % 2 == 0) ? minRand : maxRand;

    if (generatedValue > sMax || generatedValue < sMin)

    {

      // якщо значення виходить за межі з якогось боку, візьмемо протилежне

      rnd[i] = (generatedValue == minRand) ? maxRand : minRand;

    }

    else

    {

      rnd[i] = generatedValue;

    }

  }

  return rnd;

}

void TabRestData(int arrSize, double \*arr, double s1, double s2)

{

  printf("S1: %3.5f\n", s1);

  printf("S2: %3.5f\n\n", s2);

  //заголовки N та rnd[i]

  printf("%-7lc%-7s%-7lc%-10s%-7lc\n", CHARACTER, "N", CHARACTER, "rnd[i]", CHARACTER);

  //вивід самої таблиці

  for (int i = 0; i < arrSize; i++)

  {

    for (int i = 0; i < 32; i++)

    {

      printf("%lc", CHARACTER);

    }

    printf("\n");

    printf("%-7lc%-7d%-7lc%-10.2f%-7lc\n", CHARACTER, i, CHARACTER, arr[i], CHARACTER);

  }

  printf("\n");

}

int \*GenerateLinearArr(struct funcData \*data)

{

  int \*linear = (int \*)malloc(data->size \* sizeof(int));

  if (linear == NULL) exit(EXIT\_FAILURE);

  double max = data->funcValues[0];

  double min = data->funcValues[0];

  double a = 0;

  double b = 70; //за умовою

  //знаходження max та min значень f(x)

  for (int i = 1; i < data->size; i++)

  {

    if (data->funcValues[i] > max)

    {

      max = data->funcValues[i];

    }

    else if (min > data->funcValues[i])

    {

      min = data->funcValues[i];

    }

  }

  for (int i = 0; i < data->size; i++)

  {

    // використаємо нормалізацію мінімум-максимум з довільним набором значень

    linear[i] = a + (data->funcValues[i] - min) \* (b - a) / (max - min);

  }

  return linear;

}

void Print(struct funcData \*data, int \*arr) {

  printf("Graph of a linear function: \n");

  for (int i = data->size; i > 0; i--)

  {

    for (int j = 0; j < arr[i - 1] - 1; j++)

    {

      printf(" ");

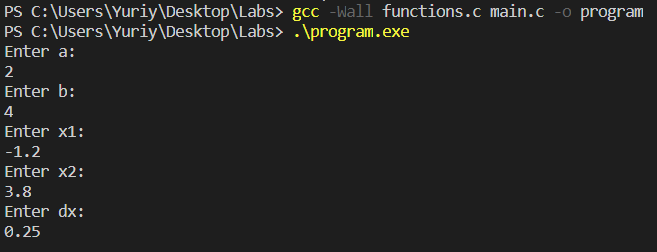
    }

    printf("%lc\n", CHARACTER);

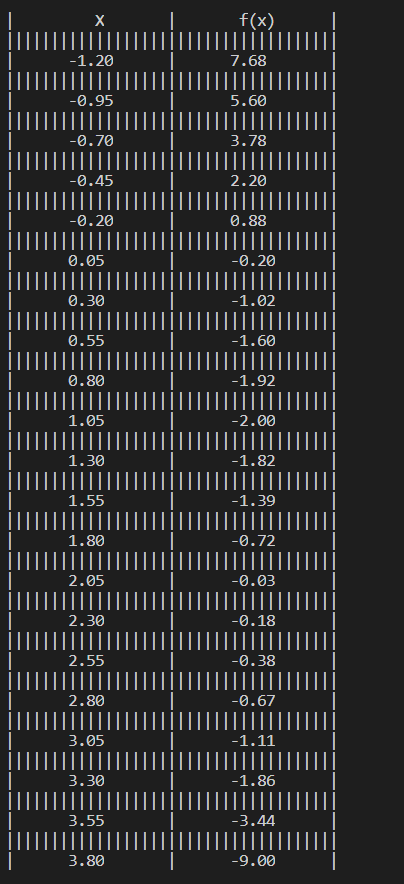
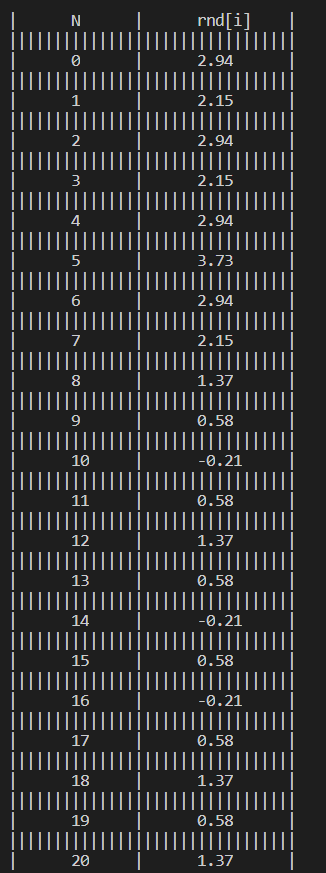
  }

}

**Результат роботи програми**



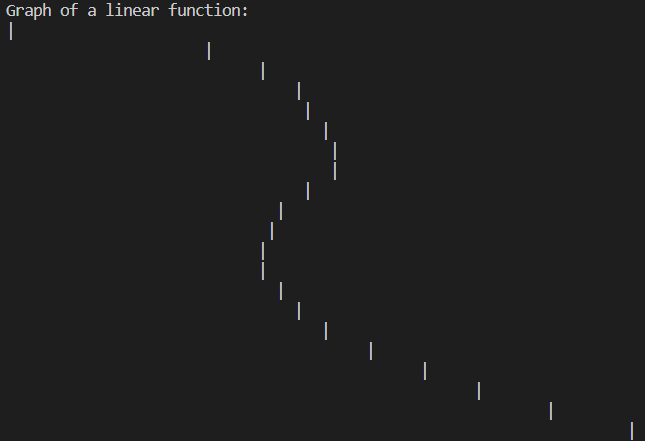
*Ввід даних користувачем*

*Таблиця значень x та f(x) Таблиця випадкових зачень*



*Значення S1 та S2*

**

*Графік ф-ї побудований з використанням символу на основі значень f(x)*

**Контрольні запитання**

7) *Що таке мови типу Асемблера?*

**Асемблер (*Assembly*)** – мова програмування, поняття якого відображає архітектуру електронно-обчислювальної машини. Мова асемблера – символьна форма запису машинного коду, використання якого спрощує написання машинних програм. ***Для однієї і тієї ж ЕОМ можуть бути розроблені різні мови асемблера***. На відміну від мов високого рівня абстракції, в яких багато проблем реалізації алгоритмів приховані від розробників, мова асемблера тісно пов'язана з системою команд мікропроцесора. Для ідеального мікропроцесора, у якого система команд точно відповідає мові програмування, асемблер виробляє по одному машинному коду на кожен оператор мови. На практиці для реальних мікропроцесорів може знадобитися кілька машинних команд для реалізації одного оператора мови.

*Найбільший плюс* мови асемблера – «наближеність» до процесора, який є основою використовуваного програмістом комп'ютера, а *головний мінус* – занадто дрібне ділення типових операцій, яке більшістю користувачів сприймається важко.

8) *Що таке імперативна парадигма програмування?*

**Імперативна парадигма програмування** – це підхід до побудови програм, що використовує алгоритмічну декомпозицію задачі, при якій програма представляється як послідовність дій (команд), які повинен виконати комп'ютер покроковим чином. Оскільки імперативна програма дуже схожа на накази комп'ютеру, то імперативне програмування іноді називають наказовим.

Імперативна парадигма програмування є відображенням архітектури традиційних ЕОМ, яка була запропонована фон Нейманом. Теоретичною основою імперативного програмування служить алгоритмічна система під назвою *«машина Т’юринга»* — абстрактний обчислювальний пристрій, що виконує послідовність команд програми, яка, таким чином, переходить з одного стану в інший. Відповідно програма в імперативному програмуванні розглядається як послідовність дій, які змінюють стан комп’ютера. Коротко це можна представити наступною нотацією:

*програма = послідовність дій.*

**ВИСНОВОК**

На даній лабораторній роботі була створена програма мовою програмування C, метою якої було використання структури для зберігання великої кількості різних значень, що зробило програму більш модульною, що в свою чергу дозволило легко міняти код, так як він став більш компактним (наприклад передача у більшість функцій лише структури, а також маніпуляція полями структури замість локальних змінних у функціях). Були розроблені функції зчитування вхідних даних, визначення значень f(x) в залежності від переданих даних користувачем та робота над отриманими значеннями, оформлення вихідних даних у вигляді таблиць, створення масиву випадкових чисел за спеціальним алгоритмом, а також побудова графіку по значенням функції використовуючи нормалізацію мінімум-максимум.

В ході виконання роботи були повторені особливості роботи зі структурами, вказівниками, масивами, модулями, особливостями роботи функцій printf (переважно для створення таблиць) та scanf у мові програмування C. Також було засвоєно новий алгоритм заповнення масиву випадковими числами та формула масштабування функції для довільних значень.