

## **Лабораторна робота №2. Вступ до програмування. Основи debug процесу**

### **1 Вимоги**

#### **1.1 Розробник**

- 1) Гладков Костянтин Сергійович
- 2) студент групи КІТ – 320
- 3) 18.10.2020

#### **1.2 Загальне завдання**

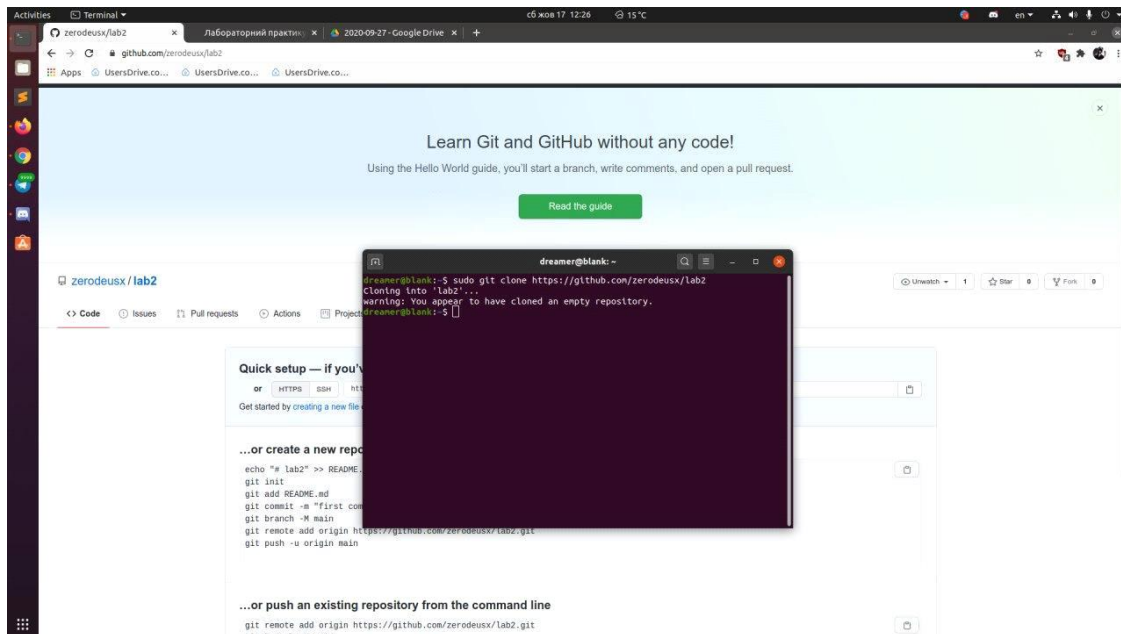
Зробити свій git репозиторій, скопіювати файл у мій репозиторій, змінити код програми у файлі, зробити debug процес з новим кодом, після чого відправити файл до github.

#### **1.3 Індивідуальне завдання**

Індивідуальне завдання відсутнє, завдання одне на всіх.

### **2 Завдання**

1. Зайшов на сайт github.com після чого створив репозиторій, назвав його lab2.
2. Використував команду git clone (git clone "URL").

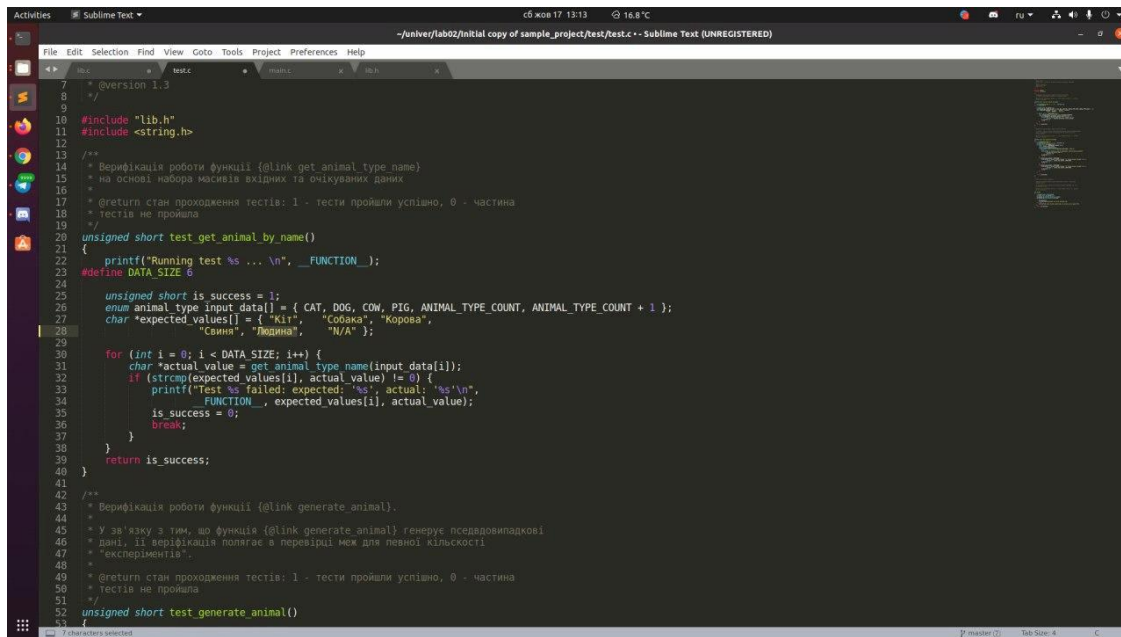


3. Знайшов папку з попередній роботі у файлового менеджера, скопіював всі файли, зайшов у папку lab2 (це мій репозиторій з гитхаба) та створив там ще одну папку під назвою lab02, туди я вставив файли, яку скопіював. Після цього я використував команду git add та git commit щоб зафіксувати зміни та назвав ці зміни "Initial Copy Of Sample Project".

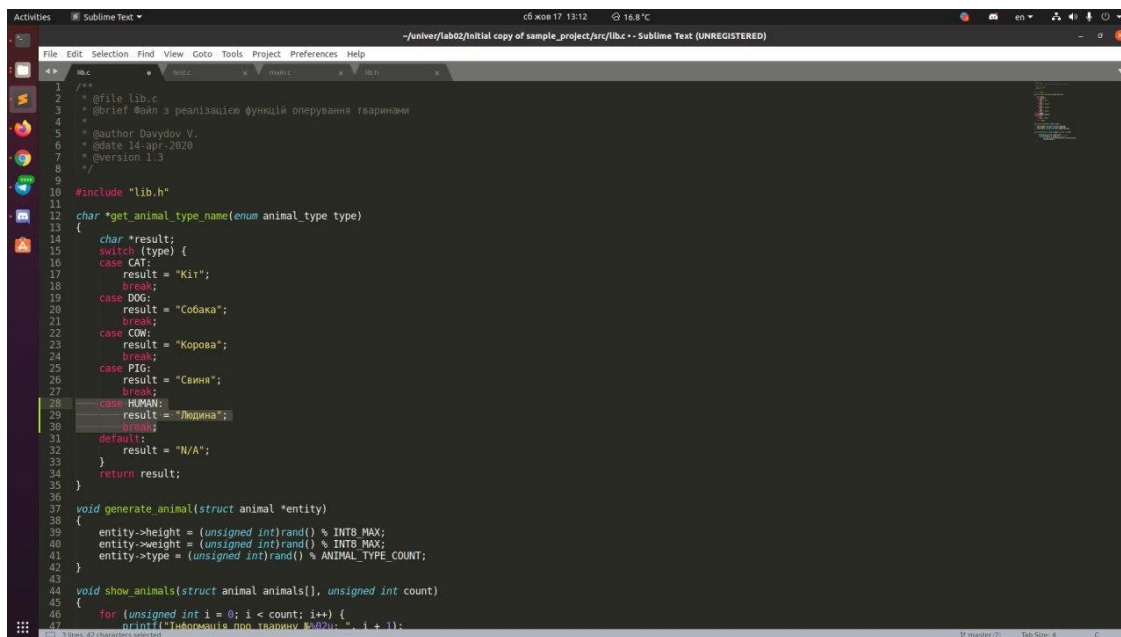
```
dreamer@blank:~/lab2/lab02$ git add .
dreamer@blank:~/lab2/lab02$ git commit
[master (root-commit) deb68dc] Initial Copy Of Sample Project
7 files changed, 366 insertions(+)
create mode 100644 lab02/Doxyfile
create mode 100644 lab02/Makefile
create mode 100644 lab02/README.md
create mode 100644 lab02/src/lib.c
create mode 100644 lab02/src/lib.h
create mode 100644 lab02/src/main.c
create mode 100644 lab02/test/test.c
```

4. Я обрав роботу на відмінно, тому додав новий тип "Людина", а зробив це так: за допомогою текстового редактору Sublime Text я відкрив lib.h (файл, який знаходився у папці з попереднього проекту)

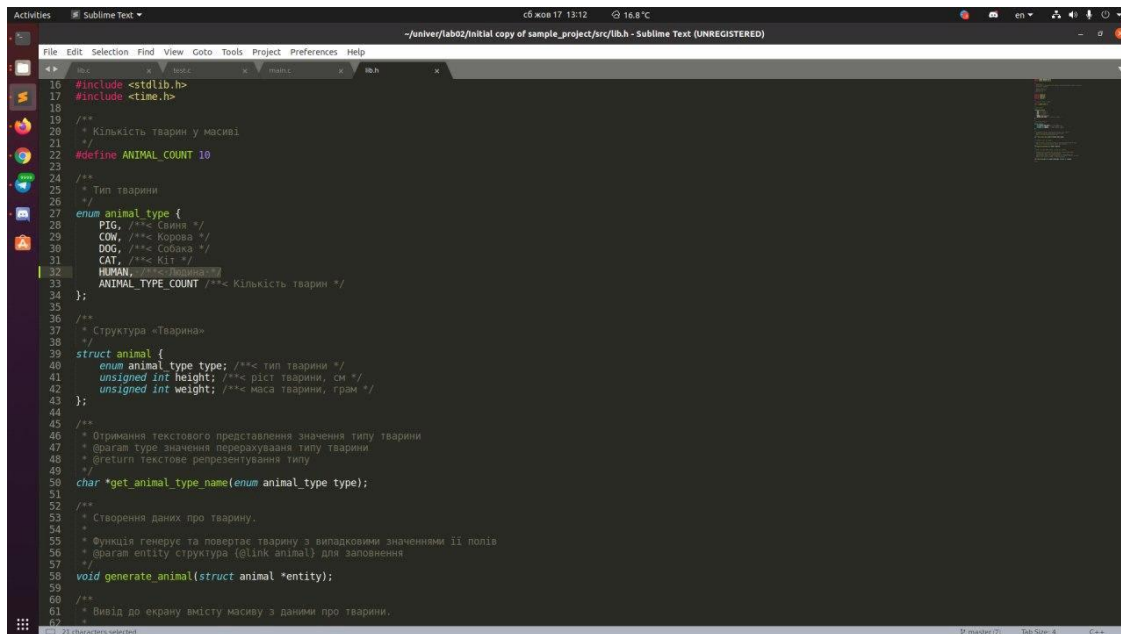
та додав там новий тип "Human". Після цього я відкрив файл test.c, де також додав у код "Людина". В останній раз я додав код у lib.c, де я зробив так, щоб якщо обирається "Людина", то програма завершувала цю частину та продовжувала з іншими. Проект я скомпілював за допомогою команди make clean prep compile.



```
1  * @version 1.3
2  */
3
4  #include "lib.h"
5  #include <string.h>
6
7  /**
8   * Верифікація роботи функції @link get_animal_type_name()
9   * на основі набору масивів вхідних та очікуваних даних
10  *
11  * @return стан проходження тестів: 1 - тести пройшли успішно, 0 - частина
12  * тестів не пройшла
13  */
14  unsigned short test_get_animal_by_name()
15  {
16      printf("Running test %s ... \n", __FUNCTION__);
17      #define DATA_SIZE 6
18
19      unsigned short is_success = 1;
20      enum animal_type input_data[] = { CAT, DOG, COW, PIG, ANIMAL_TYPE_COUNT, ANIMAL_TYPE_COUNT + 1 };
21      char *expected_values[] = { "Кіт", "Собака", "Корова",
22                                  "Свиня", "Людина", "N/A" };
23
24      for (int i = 0; i < DATA_SIZE; i++) {
25          char *actual_value = get_animal_type_name(input_data[i]);
26          if (strcmp(expected_values[i], actual_value) != 0) {
27              printf("Test %s failed: expected: '%s', actual: '%s'\n",
28                     __FUNCTION__, expected_values[i], actual_value);
29              is_success = 0;
30              break;
31          }
32      }
33      return is_success;
34  }
35
36  /**
37   * Верифікація роботи функції @link generate_animal().
38   *
39   * У зв'язку з тим, що функція @link generate_animal() генерує псевдовипадкові
40   * дані, її верифікація полягає в перевірці меж для певної кількості
41   * експериментів.
42   *
43   * @return стан проходження тестів: 1 - тести пройшли успішно, 0 - частина
44   * тестів не пройшла
45  */
46  unsigned short test_generate_animal()
47  {
48      // ...
49  }
```



```
1  * @file lib.c
2  * @brief Файл з реалізацією функцій управління тваринами
3  *
4  * @author Davydov V.
5  * @date 14-apr-2020
6  * @version 1.3
7  */
8
9  #include "lib.h"
10
11  char *get_animal_type_name(enum animal_type type)
12  {
13      char *result;
14      switch (type) {
15          case CAT:
16              result = "Кіт";
17              break;
18          case DOG:
19              result = "Собака";
20              break;
21          case COW:
22              result = "Корова";
23              break;
24          case PIG:
25              result = "Свиня";
26              break;
27          case HUMAN:
28              result = "Людина";
29              break;
30          default:
31              result = "N/A";
32      }
33      return result;
34  }
35
36  void generate_animal(struct animal *entity)
37  {
38      entity->height = (unsigned int)rand() % INT8_MAX;
39      entity->weight = (unsigned int)rand() % INT8_MAX;
40      entity->type = (unsigned int)rand() % ANIMAL_TYPE_COUNT;
41  }
42
43  void show_animals(struct animal animals[], unsigned int count)
44  {
45      for (unsigned int i = 0; i < count; i++) {
46          printf("Тварина по імені %s020: ", i + 1);
47          // ...
48      }
49  }
```



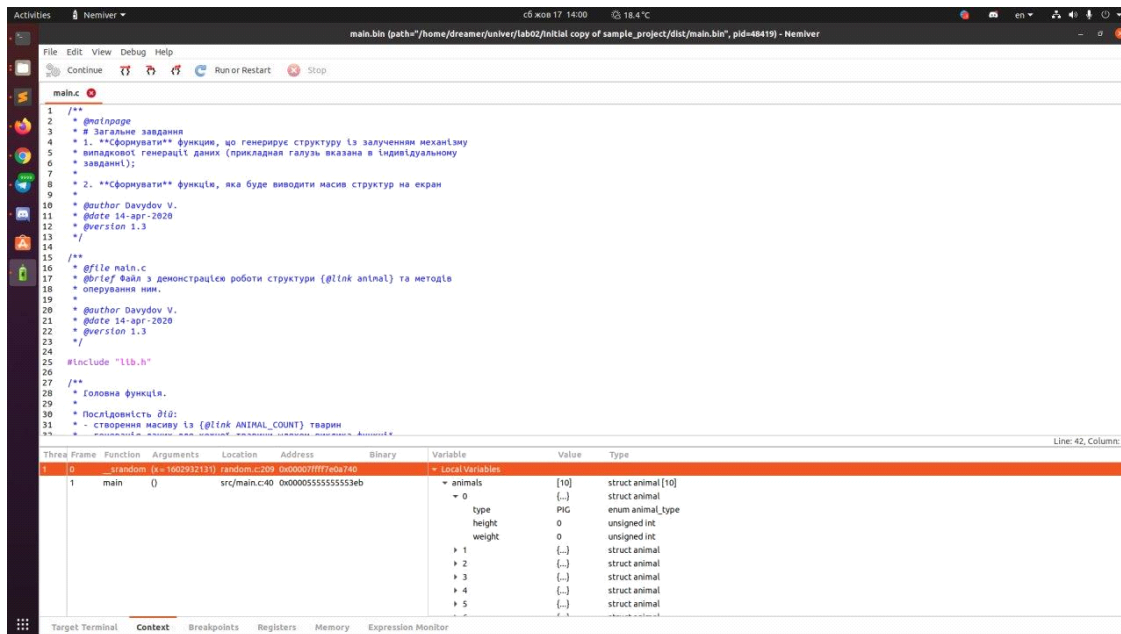
```
16 #include <stdlib.h>
17 #include <time.h>
18
19 /**
20  * Кількість тварин у масиві
21  */
22 #define ANIMAL_COUNT 10
23
24 /**
25  * Тип тварини
26  */
27 enum animal_type {
28     PIG, /**< Свиня */
29     COW, /**< Корова */
30     DOG, /**< Собака */
31     CAT, /**< Кіт */
32     HUMAN, /**< Людина */
33     ANIMAL_TYPE_COUNT /**< Кількість тварин */
34 };
35
36 /**
37  * Структура «Тварина»
38  */
39 struct animal {
40     enum animal_type type; /**< тип тварини */
41     unsigned int height; /**< ріст тварини, см */
42     unsigned int weight; /**< маса тварини, грам */
43 };
44
45 /**
46  * Отримання текстового представлення значення типу тварини
47  * @param type значення перерахування типу тварини
48  * @return текстове репрезентування типу
49  */
50 char *get_animal_type_name(enum animal_type type);
51
52 /**
53  * Створення даних про тварину.
54  */
55 /**
56  * Функція генерує та повертає тварину з випадковими значеннями її полів
57  * @param entity структура (@link animal) для заповнення
58  */
59 void generate_animal(struct animal *entity);
60
61 /**
62  * Вивід до екрану вмісту масиву з даними про тварини.
```

5. Я використовував nemiver. Там я відкрив файл main.bin та у контексті знайшов variables.

Step over – виконує наступне ствердження, але закінчує коли досягає наступну строку.

Step in - виконує наступне ствердження та намагається увійти у функцію.

Step out – завершую виконання поточної функції, закінчує, коли повертається до функції, яка викликала її.



6. Я використував команду "git diff", щоб дізнатися які зміни були після змін, які я зробив у кодї. Потім я написав команду "git add .", після цього зафіксував усе це як: "Added a type". Після цього я відправив усі ці файли до гітхабу за допомогою команди git push, де мене попросили залогинитися до гітхабу, що я і зробив.

```

dreamer@blank:~/lab2/lab02$ git diff
diff --git a/lab02/src/lib.c b/lab02/src/lib.c
index 56582cf..9a8cab4 100644
--- a/lab02/src/lib.c
+++ b/lab02/src/lib.c
@@ -25,6 +25,9 @@ char *get_animal_type_name(enum animal_type type)
     case PIG:
         result = "Свиня";
         break;
+    case HUMAN:

```

```
Activities Terminal 18:26 10°C
dreamer@blank: ~
--- a/lab02/src/lib.c
+++ b/lab02/src/lib.c
@@ -25,6 +25,8 @@ char *get_animal_type(enum animal_type type)
     case PIG:
         result = "Свиня";
         break;
+    case HUMAN:
+        result = "Людина";
+        break;
     default:
         result = "N/A";
     }
}
diff --git a/lab02/src/lib.h b/lab02/src/lib.h
index 2d3728e..b376455 100644
--- a/lab02/src/lib.h
+++ b/lab02/src/lib.h
@@ -25,6 +25,7 @@ enum animal_type {
     COU, /*< Коуса */
     DOG, /*< Собака */
     CAT, /*< Кіт */
+    HUMAN, /*< Людина */
};
dreamer@blank:~/lab2$ git commit -m "Added a type"
On branch master
Your branch is based on 'origin/master', but the upstream is gone.
(use 'git branch --unset-upstream' to fixup)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   src/lib.h
        modified:   test/test.c

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        test

no changes added to commit (use "git add" and/or "git commit -a")
dreamer@blank:~/lab2$ git add
dreamer@blank:~/lab2$ git commit -m "Added a type"
[master e20e4ac] Added a type
3 files changed, 6 insertions(+), 2 deletions(-)
dreamer@blank:~/lab2$ git push
Username for 'https://github.com': zerodeuxx
Password for 'https://zerodeuxx@github.com':
Enumerating objects: 20, done.
Counting objects: 100% (20/20), done.
Delta compression using up to 8 threads
Compressing objects: 100% (16/16), done.
Writing objects: 100% (20/20), 5.84 KiB | 997.00 KiB/s, done.
Total 20 (delta 4), reused 0 (delta 0)
remote: Resolving deltas: 100% (4/4), done.
To https://github.com/zerodeuxx/lab2
 * [new branch] master -> master
dreamer@blank:~/lab2$ cd ~
dreamer@blank: ~
```

## 7. Debug - відладчик, який знаходить помилки у коді.

Debug не використовує оптимізацію та також він дає повну інформацію про процес дебаггінгу (у PDB файлі). Release робить оптимізацію та не дає інформацію про процес дебаггінгу. Release швидший за Debug, але на мою думку краще використовувати Debug тому що Release може змінити код дуже сильно.

8. Першим ділом був створений git репозиторій за допомогою сайту github та назвов його lab2. Репозиторій був сконований до мого комп'ютеру за допомогою команди git clone. Усі файли з попередній роботі були сконовані у папку мого гіт репозиторію, файли були добавлені за допомогою команди git add. . Я зафіксував ці файли за допомогою команди git commit, назвав їх "Initial Copy Of Sample Project" та після цього почав роботи зміни у коді. Я добавив новий тип "Людина", скомпілював проект та за допомогою команди git diff я побавич усі ці зміни. Усі файли були добавлені за допомогою команди git add, потім я зафіксував зміни за допомогою команди git commit та назвав ці зміни як: "Added a type". Файли були отправлені до гітхабу за допомогою команди git push. Я користувався

дебаггером nemiver, там я додав виконуваний файл main.bin та у контексті побачив variables. Поекспериментував за "Step in", "Step over", "Step out". Режими роботи Debug та Release досліджував за допомогою того ж nemiver, також користувався інтернетом щоб мати кращу уяву про усе це.

## **Висновки**

Дізнався, що можна створювати свій репозиторій у гітхабі. Отримав досвід роботи з файлами у лінуksі, коли копіював файли. Розібрався у коді та зміг додати новий тип власноруч. Навчився як працювати з командами git та як відправляти файли до гітхабу. Отримав уявлення про роботу дебаггера та дізнався для чого він потрібен, та що існують різні методи для здійснення дебаггінгу.