**КПІ ім. Ігоря Сікорського**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформатики та програмної інженерії**

# Звіт до комп'ютерного практикуму з курсу «Основи програмування»

Прийняв
асистент кафедри ІПІ
Ахаладзе А. Е.
«13» грудня 2024 р.

Виконав
студент групи ІП-43
Дутов І. А.

**Київ 2024**

# Комп'ютерний практикум №5

***Тема:*** *Покажчики і масиви. Робота з рядками.*

**Завдання:**
Написати програму для впорядкування масиву рядків, використовуючи покажчики.

## Текст програми

***../src/main.c***

```c
#include "main.h"

#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

#include "input/input.h"

#define MAX_INT_INPUT_CHAR_COUNT 4
#define MIN_STRING_LENGTH 1
#define MAX_STRING_LENGTH 1000

#define MIN_STRING_COUNT 1
#define MAX_STRING_COUNT 1000

void print_demo() {
  printf(
      "\Вас вітає програма Sort It Out, що відсортовує випадково згенеровані "
      "або введені користувачем рядки.\n");

  show_warning("Наші обмеження такі:");
  printf("%d ≤ кількість рядків, що вводяться ≤ %d\n", MIN_STRING_COUNT,
         MAX_STRING_COUNT);
  printf("%d ≤ максимальна довжина окремого рядка у символах ≤ %d\n",
         MIN_STRING_LENGTH, MAX_STRING_LENGTH);
  show_warning(
      "Можливо згенерувати тільки рядки з цифрами, спецсимволами, смайликами, "
      "англійськими буквами. Символи сортуються за таблицею UTF-8\n");
}

int main() {
  bool is_repeat = false;
  print_demo();

  do {
    int string_count = 0;
    int max_user_string_length = 0;
    while (read_int_and_validate(&string_count, "Кількість рядків для обробки",
                           "Кількість рядків", MAX_INT_INPUT_CHAR_COUNT,
                           true, MAX_STRING_COUNT, MIN_STRING_COUNT, true,
                           false) == FAILURE);

    char **string_list = (char **)malloc(string_count * sizeof(char *));
    if (string_list == NULL) {
      show_error_memory_allocation("списку рядків");
      return FAILURE;
    }

    while (read_int_and_validate(
```

```
            &max_user_string_length, "Максимальна довжина рядка",
            "Максимальна довжина рядка", MAX_INT_INPUT_CHAR_COUNT, true,
            MAX_STRING_LENGTH, MIN_STRING_LENGTH, true, false) == FAILURE);

    bool is_random_mode =
       ask_question("Чи хочете ви увімкнути випадковий режим?");

    if (is_random_mode) {
      initialize_generator();
      bool is_dynamic_length = ask_question(
         "Чи можуть згенеровані рядки мати менше знаків, аніж %d?",
         max_user_string_length);

      CharRange *chosen_ranges;
      int range_count;

      prompt_for_ranges(&chosen_ranges, &range_count);

      // Calculating before loop to save on performance
      int *cumulative_range_ends;
      calculate_cumulative_range_ends(&cumulative_range_ends, chosen_ranges,
                                      range_count);

      for (int i = 0; i < string_count; i++) {
        int string_length = (is_dynamic_length) ? get_random_integer_in_range(
                                                      1, max_user_string_length)
                                                : max_user_string_length;

        char *random_string;
        get_random_fixed_length_string_from_ranges(
           &random_string, string_length, chosen_ranges, range_count,
           cumulative_range_ends);
        string_list[i] = random_string;
      }

      print_string_list(string_list, "Рядок", string_count);
    } else {
      for (int i = 0; i < string_count; i++) {
        while (read_string_within_length(&string_list[i], "Рядок",
                                         max_user_string_length, true,
                                         i + 1) == FAILURE);
      }
    }

    bool is_ready = false;
    do {
      is_ready = ask_question("Чи готові ви побачити відсортовані рядки?");
      if (!is_ready) {
        printf("Гаразд, ми зачекаємо Вас!\n");
      }
    } while (!is_ready);

    // Sorting strings and outputting to user
    sort_inplace_string_list(string_list, 0, string_count - 1);
    print_string_list(string_list, "Відсортований рядок", string_count);

    // Clearing memory for next iteration
    for (int i = 0; i < string_count; i++) {
      free(string_list[i]);
    }
```

```
    free(string_list);

    is_repeat = ask_question("Бажаєте повторити програму?");
  } while (is_repeat);

  show_success("Дякуємо за використання нашої програми!");

  return SUCCESS;
}
```

*../src/main.h*

```
#ifndef MAIN_H
#define MAIN_H

#define DEFAULT_OPTION_COUNT 2 // ENGLISH_UPPER and ENGLISH_LOWER

#define SUCCESS 0
#define FAILURE 1

typedef struct {
  int start;
  int end;
  int byte_count;
} CharRange;

typedef enum {
  ENGLISH_UPPER,
  ENGLISH_LOWER,
  DECIMALS,
  SPECIAL_SYMBOLS,
  EMOTICONS,
  DEFAULT_RANGE_COUNT
} DefaultCharRangeNames;

extern const CharRange DEFAULT_CHAR_RANGES[DEFAULT_RANGE_COUNT];
extern const char *DEFAULT_CHAR_RANGE_NAMES[DEFAULT_RANGE_COUNT];
extern const int DEFAULT_OPTIONS[DEFAULT_OPTION_COUNT];

void initialize_generator();
void print_string_list(char **string_list, const char *prefix, int length);
void sort_inplace_string_list(char **string_list, int low, int high);
int get_random_integer_in_range(int start, int end);
int get_char_ranges_from_options(CharRange **ranges, int *choices, int length)
    ;
int get_random_fixed_length_string_from_ranges(char **string, int length,
                                    CharRange *ranges,
                                    int range_count,
                                    int *cumulative_range_ends);

int calculate_cumulative_range_ends(int **cumulative_range_ends,
                            CharRange *ranges, int range_count);
int prompt_for_ranges(CharRange **ranges, int *range_count);

#endif
```

*../src/string/sort.c*

```c
#include <string.h>

#include "../main.h"

void swap_strings(char **string1, char **string2) {
  char *temp = *string1;
  *string1 = *string2;
  *string2 = temp;
}

int partition_strings(char **string_list, int low, int high) {
  const char *pivot = string_list[high]; // Choose the last item as pivot
  int i = low - 1; // Pointer for the next element to swap with

  for (int j = low; j < high; j++) {
    if (strcmp(pivot, string_list[j]) > 0) {
      i++;
      swap_strings(&string_list[i],
                   &string_list[j]); // Swap strings lexicographically less
    }
  }
  swap_strings(&string_list[i + 1],
               &string_list[high]); // Place the pivot in correct position
  return i + 1;
}

// Implementation of quicksort for strings
void sort_inplace_string_list(char **string_list, int low, int high) {
  if (low < high) {
    int partition_index = partition_strings(string_list, low, high);

    sort_inplace_string_list(string_list, low, partition_index - 1);
    sort_inplace_string_list(string_list, partition_index + 1, high);
  }
}
```

***../src/string/random.c***

```c
#include <stdlib.h>
#include <time.h>

#include "../input/input.h"
#include "main.h"

#ifndef MAX_BYTE_COUNT_PER_CHAR
#define MAX_BYTE_COUNT_PER_CHAR 4
#endif

const CharRange DEFAULT_CHAR_RANGES[DEFAULT_RANGE_COUNT] = {
    [DECIMALS] = {48, 57, 1},
    [SPECIAL_SYMBOLS] = {33, 47, 1},
    [ENGLISH_UPPER] = {65, 90, 1},
    [ENGLISH_LOWER] = {97, 122, 1},
    [EMOTICONS] = {128512, 128591, 4}};

const char *DEFAULT_CHAR_RANGE_NAMES[DEFAULT_RANGE_COUNT] = {
    [ENGLISH_UPPER] = "Англійські великі літери (A-Z)",
    [ENGLISH_LOWER] = "Англійські малі літери (a-z)",
```

```
    [DECIMALS] = "Десяткові цифри (0-9)",
    [SPECIAL_SYMBOLS] = "Спеціальні символи (!\"#$...)",
    [EMOTICONS] = "Смайлики"};

const int DEFAULT_OPTIONS[DEFAULT_OPTION_COUNT] = {ENGLISH_UPPER,
                                        ENGLISH_LOWER};

void initialize_generator() { srand(time(NULL)); }

int get_random_integer_in_range(int start, int end) {
  return (rand() % (end - start + 1)) + start;
}

int get_char_ranges_from_options(CharRange **ranges, int *choices, int length)
    {
  *ranges = (CharRange *)malloc(length * sizeof(CharRange));
  if (*ranges == NULL) {
    show_error_memory_allocation("перетворення числових варіантів на проміжки");
    return FAILURE;
  }

  for (int i = 0; i < length; i++) {
    (*ranges)[i] = DEFAULT_CHAR_RANGES[choices[i]];
  }

  return SUCCESS;
}

int get_random_character_in_range(CharRange range, char *buffer) {
  int random_code_point = get_random_integer_in_range(range.start, range.end);
  // Convert the code point into a UTF-8 byte sequence
  if (random_code_point <= 0x7F) {
    buffer[0] = (char)random_code_point; // 1 byte for ASCII characters
    return 1;
  } else if (random_code_point <= 0x7FF) {
    // 2 bytes for characters in the range U+0080 to U+07FF
    buffer[0] = 0xC0 | (random_code_point >> 6);
    buffer[1] = 0x80 | (random_code_point & 0x3F);
    return 2;
  } else if (random_code_point <= 0xFFFF) {
    // 3 bytes for characters in the range U+0800 to U+FFFF
    buffer[0] = 0xE0 | (random_code_point >> 12);
    buffer[1] = 0x80 | ((random_code_point >> 6) & 0x3F);
    buffer[2] = 0x80 | (random_code_point & 0x3F);
    return 3;
  } else {
    // 4 bytes for characters in the range U+10000 to U+10FFFF
    buffer[0] = 0xF0 | (random_code_point >> 18);
    buffer[1] = 0x80 | ((random_code_point >> 12) & 0x3F);
    buffer[2] = 0x80 | ((random_code_point >> 6) & 0x3F);
    buffer[3] = 0x80 | (random_code_point & 0x3F);
    return 4;
  }
}

// Ensures proper probability for different ranges
int calculate_cumulative_range_ends(int **cumulative_range_ends,
                                    CharRange *ranges, int range_count) {
  *cumulative_range_ends = (int *)malloc(range_count * sizeof(int));
  if (*cumulative_range_ends == NULL) {
```

```
    show_error_memory_allocation("масиву кінцевих точок проміжків");
    return FAILURE;
  }

  int sum = 0;
  for (int i = 0; i < range_count; i++) {
    int range_size = ranges[i].end - ranges[i].start + 1;
    sum += range_size;
    (*cumulative_range_ends)[i] = sum;
  }

  return SUCCESS;
}

int get_random_fixed_length_string_from_ranges(char **string, int length,
                                               CharRange *ranges,
                                               int range_count,
                                               int *cumulative_range_ends) {
  int max_byte_count = 1;
  for (int i = 0; i < range_count; i++) {
    if (ranges[i].byte_count > max_byte_count) {
      max_byte_count = ranges[i].byte_count;
    }
  }

  *string = (char *)malloc((length * max_byte_count + 1) *
                           sizeof(char)); // 4 bytes per UTF-8 chars
                                          // +1 for null terminator
  if (*string == NULL) {
    show_error_memory_allocation("створення випадкового рядка");
    return FAILURE;
  }

  size_t generated_char_count = 0;
  size_t generated_byte_count = 0;

  char utf8_buffer[max_byte_count];

  while (generated_char_count < length) {
    int random_char_index =
        get_random_integer_in_range(1, cumulative_range_ends[range_count - 1]);

    int chosen_range_index = 0;
    while (random_char_index > cumulative_range_ends[chosen_range_index]) {
      chosen_range_index++;
    }

    int bytes_written =
        get_random_character_in_range(ranges[chosen_range_index], utf8_buffer);

    // Copy the UTF-8 character into the string
    for (int i = 0; i < bytes_written; i++) {
      (*string)[generated_byte_count++] = utf8_buffer[i];
    }
    generated_char_count++;
  }

  (*string)[generated_byte_count] = '\0';
  return SUCCESS;
}
```

*../src/options.c*

```c
#include <stdio.h>

#include "input/input.h"
#include "main.h"
#include "stdlib.h"

#define OPTION_NUMBER_SPACING 5

void print_options() {
  printf("\n");
  for (int i = 0; i < DEFAULT_RANGE_COUNT; i++) {
    printf(GREEN "ОПЦІЯ %-*d" RESET "%s\n", OPTION_NUMBER_SPACING, i + 1,
           DEFAULT_CHAR_RANGE_NAMES[i]);
  }
}

void print_selected_options(const int *selected_options, int option_count) {
  printf("[");
  for (int i = 0; i < option_count; i++) {
    printf("%d", selected_options[i] + 1);
    // If the last element, do not print the ,
    if (i != option_count - 1) {
      printf(", ");
    }
  }
  printf("]: ");
}

int add_option(int **arr, int *count, int option) {
  (*count)++;
  int *temp = realloc(*arr, (*count) * sizeof(int));
  if (!temp) {
    show_error_memory_allocation("перевиділення пам'яті для списку опцій");
    return FAILURE;
  }
  temp[*count - 1] = option;
  *arr = temp;
  return SUCCESS;
}

int remove_option(int **arr, int *count, int index) {
  if (index < 0 || index >= *count) {
    show_error("Неприйнятний індекс для видалення опції.");
    return FAILURE;
  }
  for (int i = index; i < *count - 1; i++) {
    (*arr)[i] = (*arr)[i + 1];
  }
  (*count)--;

  if (*count == 0) {
    free(*arr);
    *arr = NULL;
  } else {
    int *temp = realloc(*arr, (*count) * sizeof(int));
    if (!temp) {
      show_error_memory_allocation(
          "перевиділення пам'яті після видалення опції");
```

```
    return FAILURE;
   }
   *arr = temp;
  }

  return SUCCESS;
}

int process_option_input(char *c, int option_upper_bound) {
  *c = getchar();
  if (*c >= '1' && *c <= option_upper_bound) {
   return *c - '0' - 1; // Return the option as index
  }
  return -1;
}

int prompt_for_ranges(CharRange **ranges, int *range_count) {
  print_options();

  show_warning(
     "Оберіть однудекілька- опцій за цифрами. Повторний ввід цифри призведе "
     "до скасування "
     "її вибору.\n Введіть 0, коли будете вдоволені своїм набором. За "
     "замовчуванням обирається набір англійської мови.");

  const char option_upper_bound = (char)((int)'0' + DEFAULT_RANGE_COUNT);
  int option_count = 0;
  int *selected_options = NULL;
  char c;
  bool is_ended = false;

  while (!is_ended) {
   int option = process_option_input(&c, option_upper_bound);

   if (option >= 0) { // If valid input (1 to option_upper_bound)
    bool is_repeat_found = false;
    for (int i = 0; i < option_count; i++) {
     if (selected_options[i] == option) {
      is_repeat_found = true;
      if (remove_option(&selected_options, &option_count, i) == FAILURE) {
       return FAILURE;
      }
      i--;
      break;
     }
    }
    if (!is_repeat_found) {
     if (add_option(&selected_options, &option_count, option) == FAILURE) {
      return FAILURE;
     }
    }
   } else if (c == '0') {
    is_ended = true;
    clear_stdin();
   } else if (c == ' ' || c == '\t') {
    continue;
   } else if (c == '\n') {
    print_selected_options(selected_options, option_count);
   } else {
    show_warning(
```

```
        "Усі цифри і символи, що не входять в опції, пропускаються.");
    }
  }

  if (option_count == 0) {
    selected_options = (int *)DEFAULT_OPTIONS;
    option_count = DEFAULT_OPTION_COUNT;
  }

  if (get_char_ranges_from_options(ranges, selected_options, option_count) ==
      FAILURE) {
    return FAILURE;
  }

  *range_count = option_count;
  return SUCCESS;
}
```

***../src/input/input.h***

```
#ifndef INPUT_INTERNALS_H
#define INPUT_INTERNALS_H

#include <stdbool.h>

#define RED "\033[31m"
#define GREEN "\033[32m"
#define YELLOW "\033[33m"
#define RESET "\033[0m"

#ifndef SUCCESS
#define SUCCESS 0
#endif

#ifndef FAILURE
#define FAILURE 1
#endif

#ifndef PIPE
#define PIPE 1
#endif

typedef enum {
  GREATER_EQUAL,
  GREATER,
  LESS,
  LESS_EQUAL,
  WITHIN_RANGE
} RangeCheckResult;

int show_error(const char *format, ...) __attribute__((format(printf, 1, 2)));
int show_warning(const char *format, ...) __attribute__((format(printf, 1, 2)))
    ;
int show_success(const char *format, ...) __attribute__((format(printf, 1, 2)))
    ;
bool ask_question(const char *format, ...)
    __attribute__((format(printf, 1, 2)));

int show_error_overlength(const char *name, int max_char_count);
```

```c
int show_error_memory_allocation(const char *reason);
int show_error_not_number(const char *name);

int read_input(char *input, int max_char_count, const char *name);

// Utils
void replace_commas_with_dots(char *string);
int ask_confirmation();
void clear_stdin();
void remove_newline(char *string);

bool is_input_floating_point(char *str);
bool is_input_within_length(const char *input);
bool is_input_number_after_conversion(const char *endptr, const char *input);

// Reading valid input
int read_int_and_validate(int *value, const char *full_name,
                    const char *short_name, int max_char_count,
                    bool is_restricted, int max_value, int min_value,
                    bool is_max_included, bool is_min_included);

int read_string_within_length(char **value, const char *name,
                        int max_char_count, bool is_numbered, int number);
#endif
```

***../src/input/int.c***

```c
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

#include "input.h"

#define MAX_VALUE (long)INT_MAX
#define MIN_VALUE (long)INT_MIN

void prompt_user_input_int(const char *name, bool is_restricted, int min_value,
                     int max_value) {
  if (is_restricted) {
    printf("%s від (%d до %d): ", name, min_value, max_value);
  } else {
    printf("%s: ", name);
  }
}

int show_range_error_int(const char *name, RangeCheckResult result,
                   int min_value, int max_value) {
  switch (result) {
    case LESS:
      show_error("%s має бути більшийрівний- %d.\n", name, min_value);
      break;
    case LESS_EQUAL:
      show_error("%s має бути більший за %d.\n", name, min_value);
      break;
    case GREATER:
      show_error("%s має бути меншийрівний- %d.\n", name, max_value);
      break;
    case GREATER_EQUAL:
```

```
    show_error("%s має бути менший за %d.\n", name, max_value);
    break;
  default:
    break;
 }
 return PIPE;
}

// Range Checking Functions
RangeCheckResult validate_range_int(int value, int min_value, int max_value,
                            bool is_min_included,
                            bool is_max_included) {
 if (is_min_included && value < min_value) return LESS;
 if (!is_min_included && value ⩽ min_value) return LESS_EQUAL;
 if (is_max_included && value > max_value) return GREATER;
 if (!is_max_included && value ⩾ max_value) return GREATER_EQUAL;
 return WITHIN_RANGE;
}

RangeCheckResult validate_overflow_int(long int value) {
 if (value > MAX_VALUE) return GREATER;
 if (value < MIN_VALUE) return LESS;
 return WITHIN_RANGE;
}

int read_int_and_validate(int *value, const char *full_name,
                    const char *short_name, int max_char_count,
                    bool is_restricted, int max_value, int min_value,
                    bool is_max_included, bool is_min_included) {
 prompt_user_input_int(full_name, is_restricted, min_value, max_value);

 char input[max_char_count + 2];

 if (read_input(input, max_char_count, full_name) == FAILURE) {
  return FAILURE;
 }

 if (!is_input_within_length(input)) {
  show_error_overlength(full_name, max_char_count);
  clear_stdin();
  return FAILURE;
 }

 char *endptr;
 long int temp_value = strtol(input, &endptr, 10);

 if (!is_input_number_after_conversion(endptr, input)) {
  show_error_not_number(full_name);
  return FAILURE;
 }

 RangeCheckResult global_check = validate_overflow_int(temp_value);
 if (global_check != WITHIN_RANGE) {
  show_range_error_int(short_name, global_check, min_value, max_value);
  return FAILURE;
 }

 *value = (int)temp_value;

 if (is_restricted) {
```

```
    RangeCheckResult range_check = validate_range_int(
        *value, min_value, max_value, is_min_included, is_max_included);
    if (range_check != WITHIN_RANGE) {
      show_range_error_int(full_name, range_check, min_value, max_value);
      return FAILURE;
    }
  }

  return SUCCESS;
}
```

*../src/input/utils.c*

```
#include <stdarg.h>
#include <stdio.h>
#include <string.h>

#include "input.h"

int show_error(const char *format, ...) {
  va_list args;
  va_start(args, format);

  printf("\n" RED "ПОМИЛКА!␣");
  vprintf(format, args);
  printf(RESET "\n");

  va_end(args);
  return PIPE;
}

int show_warning(const char *format, ...) {
  va_list args;
  va_start(args, format);

  printf("\n" YELLOW "УВАГА!␣");
  vprintf(format, args);
  printf(RESET "\n");

  va_end(args);
  return PIPE;
}

int show_success(const char *format, ...) {
  va_list args;
  va_start(args, format);

  printf("\n" GREEN "ПЕРЕМОГА!␣");
  vprintf(format, args);
  printf(RESET "\n");

  va_end(args);
  return PIPE;
}

int ask_confirmation() {
  char choice;

  printf(
```

```
    "Введіть + , якщо погоджуєтесь. Інакше введіть будьяку- "
    "іншу клавішу: ");

  choice = getchar();
  clear_stdin();

  if (choice == '+') {
    return SUCCESS;
  }

  return FAILURE;
}

bool ask_question(const char *format, ...) {
  va_list args;
  va_start(args, format);
  printf("\n" YELLOW);
  vprintf(format, args);
  printf(RESET "\n");
  va_end(args);

  return (ask_confirmation() == SUCCESS) ? true : false;
}

int show_error_overlength(const char *name, int max_char_count) {
  show_error("Довжина %s в символах має бути меншою за %d.\n", name,
             max_char_count);
  return PIPE;
}

int show_error_memory_allocation(const char *reason) {
  show_error("Не вдалося виділити достатньо памяті' для %s.", reason);
  return PIPE;
}

int show_error_not_number(const char *name) {
  show_error("%s має бути числом і не містити додаткових символів!", name);
  return PIPE;
}

int read_input(char *input, int max_char_count, const char *name) {
  if (!fgets(input, max_char_count + 2, stdin)) {
    show_error("Не вдалося прочитати ввід для %s.\n", name);
    return FAILURE;
  }
  return SUCCESS;
}

void clear_stdin() {
  int c;
  while ((c = getchar()) != '\n' && c != EOF);
}

void remove_newline(char *string) {
  size_t string_length = strlen(string);
  if (string_length > 0 && string[string_length - 1] == '\n') {
    string[string_length - 1] = '\0';
  }
}

```

```c
bool is_input_within_length(const char *input) {
  // Last character should be '\n' if input is within length
  return (input[strlen(input) - 1] == '\n' && strlen(input) - 1 != 0);
}

bool is_input_floating_point(char *str) {
  while (*str) {
    switch (*str) {
      case '.':
      case ',':
      case 'e':
      case 'E':
        return true;
      default:
        break;
    }
    str++;
  }
  return false;
}

bool is_input_number_after_conversion(const char *endptr, const char *input) {
  return (endptr != input && *endptr == '\n');
}
```

***../meson.build***

```
project('op-lab-5', 'c')

c_args = ['-Isrc', '-g' ]
include_dirs = include_directories('src')

main_sources = [
  'src/main.c',
  'src/options.c',
  'src/input/int.c',
  'src/input/string.c',
  'src/input/utils.c',
  'src/string/random.c',
  'src/string/sort.c'
]

executable('main.out', main_sources,
  c_args: c_args,
  include_directories: include_dirs,
  link_args: []
)
```

**Введені та одержані результати**

```
Кількість рядків для обробки (від 1 до 1000): 10
Максимальна довжина рядка (від 1 до 1000): 10

Чи хочете ви увімкнути випадковий режим?
Введіть +, якщо погоджуєтесь. Інакше введіть будь-яку іншу клавішу: +

Чи можуть згенеровані рядки мати менше знаків, аніж 10?
Введіть +, якщо погоджуєтесь. Інакше введіть будь-яку іншу клавішу: -

ОПЦІЯ 1    Англійські великі літери (A-Z)
ОПЦІЯ 2    Англійські малі літери (a-z)
ОПЦІЯ 3    Десяткові цифри (0-9)
ОПЦІЯ 4    Спеціальні символи (!"#$...)
ОПЦІЯ 5    Смайлики

УВАГА! Оберіть одну-декілька опцій за цифрами. Повторний ввід цифри призведе до скасування її ви
 Введіть 0, коли будете вдоволені своїм набором. За замовчуванням обирається набір англійської м
1230

Рядок 1 : YVCaS46Bem
Рядок 2 : m02qwQGtrV
Рядок 3 : c4K2e7Qrpx
Рядок 4 : jJ7CxEzkRC
Рядок 5 : FNwkOzYTnY
Рядок 6 : uoohBb8B7i
Рядок 7 : Yl7ZB9DR8x
Рядок 8 : jtO4NY2hSG
Рядок 9 : ItndNnaIgC
Рядок 10: hE4FXfqtqE

Чи готові ви побачити відсортовані рядки?
Введіть +, якщо погоджуєтесь. Інакше введіть будь-яку іншу клавішу: +

Відсортований рядок 1 : FNwkOzYTnY
Відсортований рядок 2 : ItndNnaIgC
Відсортований рядок 3 : YVCaS46Bem
Відсортований рядок 4 : Yl7ZB9DR8x
Відсортований рядок 5 : c4K2e7Qrpx
Відсортований рядок 6 : hE4FXfqtqE
Відсортований рядок 7 : jJ7CxEzkRC
Відсортований рядок 8 : jtO4NY2hSG
Відсортований рядок 9 : m02qwQGtrV
Відсортований рядок 10: uoohBb8B7i

Бажаєте повторити програму?
Введіть +, якщо погоджуєтесь. Інакше введіть будь-яку іншу клавішу: 
```

```
Кількість рядків для обробки (від 1 до 1000): 10
Максимальна довжина рядка (від 1 до 1000): 20

Чи хочете ви увімкнути випадковий режим?
Введіть +, якщо погоджуєтесь. Інакше введіть будь-яку іншу клавішу: +

Чи можуть згенеровані рядки мати менше знаків, аніж 20?
Введіть +, якщо погоджуєтесь. Інакше введіть будь-яку іншу клавішу: +

ОПЦІЯ 1    Англійські великі літери (A-Z)
ОПЦІЯ 2    Англійські малі літери (a-z)
ОПЦІЯ 3    Десяткові цифри (0-9)
ОПЦІЯ 4    Спеціальні символи (!"#$...)
ОПЦІЯ 5    Смайлики 🌍😃😂

УВАГА! Оберіть одну-декілька опцій за цифрами. Повторний ввід цифри призведе до скасування її вибору.
 Введіть 0, коли будете вдоволені своїм набором. За замовчуванням обирається набір англійської мови.
234
[2, 3, 4]: 0

Рядок 1 : *,(gj#as1'!).".d)v
Рядок 2 : 8gq"e11s(6q.m10!8q+
Рядок 3 : *x2bff9'9a-
Рядок 4 : 2"l7-q2n2t&
Рядок 5 : kphkiw/w#hm42wl
Рядок 6 : x)%uix1,sycbt
Рядок 7 : pedn+&y*p72.2#5v/a
Рядок 8 : 5ju
Рядок 9 : iq'0(
Рядок 10: wh.p

Чи готові ви побачити відсортовані рядки?
Введіть +, якщо погоджуєтесь. Інакше введіть будь-яку іншу клавішу: +

Відсортований рядок 1 : *,(gj#as1'!).".d)v
Відсортований рядок 2 : *x2bff9'9a-
Відсортований рядок 3 : 2"l7-q2n2t&
Відсортований рядок 4 : 5ju
Відсортований рядок 5 : 8gq"e11s(6q.m10!8q+
Відсортований рядок 6 : iq'0(
Відсортований рядок 7 : kphkiw/w#hm42wl
Відсортований рядок 8 : pedn+&y*p72.2#5v/a
Відсортований рядок 9 : wh.p
Відсортований рядок 10: x)%uix1,sycbt

Бажаєте повторити програму?
Введіть +, якщо погоджуєтесь. Інакше введіть будь-яку іншу клавішу: -

ПЕРЕМОГА! Дякуємо за використання нашої програми!
```

```
УВАГА! Можливо згенерувати тільки рядки з цифрами, спецсимволами, смайликами, англ
ами. Символи сортуються за таблицею UTF-8

Кількість рядків для обробки (від 1 до 1000): 12
Максимальна довжина рядка (від 1 до 1000): 100

Чи хочете ви увімкнути випадковий режим?
Введіть +, якщо погоджуєтесь. Інакше введіть будь-яку іншу клавішу: -
Рядок 1: I
Рядок 2: LoV3
Рядок 3: Pr0gr4mming
Рядок 4: B3cause
Рядок 5: 1t
Рядок 6: 5ives
Рядок 7: Th3
Рядок 8: Fr33dom
Рядок 9: T0
Рядок 10: Cr3ate Things
Рядок 11: 3very
Рядок 12: DaY!

Чи готові ви побачити відсортовані рядки?
Введіть +, якщо погоджуєтесь. Інакше введіть будь-яку іншу клавішу: +

Відсортований рядок 1 : 1t
Відсортований рядок 2 : 3very
Відсортований рядок 3 : 5ives
Відсортований рядок 4 : B3cause
Відсортований рядок 5 : Cr3ate Things
Відсортований рядок 6 : DaY!
Відсортований рядок 7 : Fr33dom
Відсортований рядок 8 : I
Відсортований рядок 9 : LoV3
Відсортований рядок 10: Pr0gr4mming
Відсортований рядок 11: T0
Відсортований рядок 12: Th3

Бажаєте повторити програму?
Введіть +, якщо погоджуєтесь. Інакше введіть будь-яку іншу клавішу:
```

```
Кількість рядків для обробки (від 1 до 1000): 8
Максимальна довжина рядка (від 1 до 1000): 20

Чи хочете ви увімкнути випадковий режим?
Введіть +, якщо погоджуєтесь. Інакше введіть будь-яку іншу клавішу: +

Чи можуть згенеровані рядки мати менше знаків, аніж 20?
Введіть +, якщо погоджуєтесь. Інакше введіть будь-яку іншу клавішу: +

ОПЦІЯ 1    Англійські великі літери (A-Z)
ОПЦІЯ 2    Англійські малі літери (a-z)
ОПЦІЯ 3    Десяткові цифри (0-9)
ОПЦІЯ 4    Спеціальні символи (!"#$...)
ОПЦІЯ 5    Смайлики

УВАГА! Оберіть одну-декілька опцій за цифрами. Повторний ввід цифри призведе до скасув
 Введіть 0, коли будете вдоволені своїм набором. За замовчуванням обирається набір анг
12345
[1, 2, 3, 4, 5]: 0

Рядок 1 : m🥶v#😹😐😔😎😊😩🧙‍♂️Ae🐱q2😎bD🥶
Рядок 2 : PK😦😣s
Рядок 3 : A😂wl74Ba😝-e🧿🙋‍♂️A!😤😮C
Рядок 4 : +😐srA😵Y😵😵"2
Рядок 5 : 😃t
Рядок 6 : v😞
Рядок 7 : k😁vk🙏U&
Рядок 8 : 😾👧oF😨7l😵e😛4A7🤪X

Чи готові ви побачити відсортовані рядки?
Введіть +, якщо погоджуєтесь. Інакше введіть будь-яку іншу клавішу: +

Відсортований рядок 1 : +😐srA😵Y😵😵"2
Відсортований рядок 2 : A😂wl74Ba😝-e🧿🙋‍♂️A!😤😮C
Відсортований рядок 3 : PK😦😣s
Відсортований рядок 4 : k😁vk🙏U&
Відсортований рядок 5 : m🥶v#😹😐😔😎😊😩🧙‍♂️Ae🐱q2😎bD🥶
Відсортований рядок 6 : v😞
Відсортований рядок 7 : 😃t
Відсортований рядок 8 : 😾👧oF😨7l😵e😛4A7🤪X
```

## Теоретичні розрахунки

Цей комп'ютерний практикум не потребує жодних розрахунків, адже пов'язаний виключно з рядками.

***Висновки:*** Програма працює коректно. Програма вирішує поставлене завдання.