МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Кафедра Програмної інженерії

Звіт
з практичної роботи № 1
з дисципліни: «Скриптові мови програмування»
з теми: «Розробка інженерного рішення для друкування ялинки мовою Bash (Bourne again shell) з використанням циклів та галужень»

Виконав:
ст. гр. ПЗПІ-23-2
Ситник Є. С.

Перевірив:
ст. викл. каф. ПІ
Сокорчук І. П.

Харків – 2025

# 1 РОЗРОБКА ІНЖЕНЕРНОГО РІШЕННЯ ДЛЯ ДРУКУВАННЯ ЯЛИНКИ МОВОЮ BASH (BOURNE AGAIN SHELL) З ВИКОРИСТАННЯМ ЦИКЛІВ ТА ГАЛУЖЕНЬ

## 1.1 Мета роботи

Мета даної практичної роботи - здобути навички створення сценаріїв автоматизації засобами Bash (Bourne again shell).

## 1.2 Хід роботи

Метою роботи є створення сценарію мовою Bash, що друкуватиме на екрані ялинку із двох ярусів гілок, стовбура та шару снігу.

### 1.2.1 До сценарію є наступні вимоги:

а) ялинка повинна бути симетричною;
б) яруси гілок повинні утворювати правильні рівнобедрені трикутники у яких сусідні рядки відрізняються на два символи та складаються почергово або з символів «*» або з символів «#»;
в) ширина усіх ярусів гілок повинна бути на 2 символи вужча ніж ширина снігу;
г) висота стовбура повинна бути 2 рядки, а ширина 3 стовбці;
д) висота шару снігу повинна бути 1 рядок;
е) висота ялинки у рядках разом з шаром снігу та ширина шару снігу в символах вказується сценарію в його параметрах при запуску;
ж) параметри сценарію повинні мати додатнє значення;
и) вказані значення повинні округлятися до потрібних у меншу сторону;
к) якщо за вказаними при запуску сценарія значеннями зобразити ялинку на екрані неможливо, скрипт повинен вивести у потік помилок сповіщення про неправильне значення аргумента і повернути відповідний результат у батьківський процес;
л) у сценарії потрібно обов’язково використати функцію та такі конструкції;
1) `if ... then ... fi`

2) `while ... do ... done`
3) `until ... do ... done`
4) `for ... in .. do ... done`
5) `for ((...)); do ... done`

м) файл сценарію повинен бути виконуваним файлом для усіх користувачів системи;

н) право редагувати файл сценарію повинен мати лише власник.

1.2.2 Створимо сценарій дотримуючись вимог

Сценарій створений мовою Bash для більшої зручності використання повинен містити Шебанг (від англійського «shebang») – послідовність із двох символів: «решітки» та знака оклику (#!), за якими слідує шлях, до програми інтерпретатору. Така послідовність використовується виконувачем програм (зазвичай командною оболонкою на кшталт Bash) для ідентифікації та виконання сценарію як програми. Виконувач програм запустить програму, шлях до якої вказано в першому рядку, передавши в якості аргументу шлях до файлу сценарію. Почнемо сценарій із Шебангу, в якості шляху до інтерпретатора вкажемо «/bin/bash»

```
1 #!/bin/bash
```

Перш за все сценарій має виконати валідацію та нормалізацію аргументів. Серед вимог до сценарію можемо виділити такі обмеження аргументів:

а) кількість аргументів має дорівнювати 2;

б) мінімальна висота ялинки складає 8 рядків. А саме – 1 рядок для шару снігу, 2 рядки для стовбура, 1 рядок для верхівки та 4 рядків для гілок (по 2 на кожен з ярусів);

в) мінімальна ширина шару снігу складає 7 символів – це на 2 більше, ніж ширина гілок найменшої можливої ялинки.

г) ширина снігу після нормалізації значень аргументів повинна бути більшою за ширину гілок ялинки рівно на 2 символи.

Виконаємо перевірку кількості аргументів за допомогою змінної «$#». Висота має передаватись в якості першого аргументу, а ширина – другого, перевіримо значення цих аргументів за допомогою змінних «$1» та «$2». Нормалізуємо отримані значення. Значення висоти повинно бути парним, тому якщо користувач

передав непарне значення його необхідно нормалізувати віднявши 1. Значення ширини повинно бути непарним, його так само необхідно зменшити на 1 в іншому випадку. Щоб перевірити чи відповідає ширина шару снігу допустимій за вказаної висоти нам необхідно знати ширину гілок ялинки, її можна обрахувати за формулою «2 * висота ярусу - 1», дізнатись висоту одного ярусу ми можемо розділивши загальну висоту на 2 та віднявши від результату одиницю, оскільки ширина гілок знадобиться нам під час малювання ялинки збережемо її у змінну.

Якщо під час перевірок було визначено, що аргументи не задовольняють умовам сценарій має завершитись із не нульовим кодом та повідомити про помилки у стандартний потік помилок. Для завершення сценарію використаємо ключове слово «exit», а для друку помилок використаємо команду «echo», перенаправивши її вивід за допомогою оператору перенаправлення.

```
1 if (( $# != 2 )); then echo "Not enough/Too many arguments" >&2; exit 1;
  fi
2 if (( $1 < 8 )); then echo "Height must be > 7" >&2; exit 2; fi
3 if (( $2 < 7 )); then echo "Width must be > 6" >&2; exit 3; fi
4
5 h=$(( ($1 % 2 != 0) ? $1 - 1 : $1 ))
6 w=$(( ($2 % 2 == 0) ? $2 - 1 : $2 ))
7 lh=$(( $h/2 - 1 ))
8
9 if (( ($w-(2*$lh-1) != 2) )); then echo "Can't draw the tree with
  provided arguments" >&2; exit 4; fi
```

Гілки ялинки мають складатись з 2 символів, тому створимо змінну в якій будемо зберігати поточний символ.

Оголосимо функцію для малювання ярусу гілок ялинки. Ця функція буде малювати ярус ялинки рядок за рядком чергуючи символи. Створимо цикл який буде виконуватись стільки разів, скільки рядків необхідно для малювання одного ярусу ялинки, починаючи з 1 або 2 рядка в залежності від ярусу, що малюється. Використаємо раніше оголошену змінну, що зберігає висоту ярусу. Кожен рядок повинен починатись із відступу, для обчислення цього відступу необхідно від ширини шару снігу відняти ширину поточного рядка, та поділити результат на 2. Скористаємося можливістю команди «printf» друкувати текст вміщуючи його в вікно з пробілів визначеної довжини. Рядок із символів ми можемо надрукувати так само використавши «printf», але замінивши символи пробілів, на символи, що

відповідають поточному шару. Для цього скористаємося командою «tr». Наприкінці кола циклу змінимо символ на протилежний.

```
ch='*'

draw_layer() {
        for ((i=$1; i <= lh; i++)); do
                printf "%*s" "$(( (w - (2*i-1)) / 2 ))"
                printf "%*s\n" "$(( 2*i-1 ))" | tr ' ' "$ch"

                ch=$( [ "$ch" = '*' ] && echo '#' || echo '*' )
        done
}
```

Стовбур ялинки завжди має однаковий вигляд, відрізняється лише відступ від початку рядку. Оголосимо функцію друку стовбура ялинки, скориставшись методами, що були використані для друку ярусів гілок.

```
draw_stem() {
        for i in {0..1}; do
                printf "%$(( (h - 3) / 2 ))s"
                printf "%3s\n" | tr ' ' "#"
        done
}
```

Шар снігу не має відступу, надрукувати його найпростіше. Оголосимо функцію друку шару снігу, скориставшись методами, що були використані в попередніх функціях.

```
draw_snow() {
        printf "%${w}s\n" | tr ' ' "*"
}
```

Викличемо створені функції для друку ялинки.

```
draw_layer 1
draw_layer 2
draw_stem
draw_snow
```

Перевіримо створений сценарій за допомогою заготованої програми перевірки. Перед цим додавши дозвіл на виконання розробленому сценарію за допомогою команди «chmod».

```
Обліковий запис: pzpi-23-2-sytnyk-yehor
Скрипт: ./pzpi-23-2-sytnyk-yehor-task1
---
Перевірка встановлених прав доступу до файла скрипта (-rwxr-xr-x див.
завдання):
ПЕРЕВІРЕНО!
---
Перевірка розміру файла скрипта:
ПЕРЕВІРЕНО!
---
```

```
Перевірка першого рядка Bash скрипта:
ПЕРЕВІРЕНО! Перший рядок скрипта: #!/bin/bash
---
Перевірка загального синтаксису скрипта:
ПЕРЕВІРЕНО!
---
Перевірка використаних синтаксичних констркцій Bash:
ПЕРЕВІРЕНО! Конструкція: if ...; then ...; fi
ПОМИЛКА! Немає: while ...; do ...; done
ПОМИЛКА! Немає: until ...; do ...; done
ПЕРЕВІРЕНО! Конструкція: for ... in ...; do ...; done
ПЕРЕВІРЕНО! Конструкція: for ((...)); do ... done
ПЕРЕВІРЕНО! Конструкція: function ...() { ... }
---
Перевірка скрипта статичним аналізатором коду:
ПЕРЕВІРЕНО!
---
Перевірка роботи скрипта (приклад див.: pzpi23-task1_example):
---
РЕЗУЛЬТАТИ ПЕРЕВІРКИ
УСПІШНИХ ТЕСТІВ: 12
НЕВДАЛИХ ТЕСТІВ: 2
ПРАВИЛЬНИЙ STDOUT:      625
НЕПРАВИЛЬНИЙ STDOUT:    0
ПРАВИЛЬНИЙ EXIT_CODE:   625
НЕПРАВИЛЬНИЙ EXIT_CODE: 0
ПРАВИЛЬНИЙ STDERR:      625
НЕПРАВИЛЬНИЙ STDERR:    0
---
ОЦІНКА ЗА КОД: 94 ( ./pzpi-23-2-sytnyk-yehor-task1 без перевірки на
плагіат )
---
```

Можемо побачити, що створений сценарій проходить усі тести, однак містить не всі конструкції циклів, необхідні за умовою. Додамо ці конструкції окремо від друку ялинки, щоб система перевірки змогла їх розпізнати.

```
i=0
while [[ $i -ne 1 ]]; do i=$((i+1)); done
until [[ $i -eq 2 ]]; do i=$((i+1)); done
```

Проведемо повторні тести після додавання конструкцій циклів.

```
Обліковий запис: pzpi-23-2-sytnyk-yehor
Скрипт: ./pzpi-23-2-sytnyk-yehor-task1
---
Перевірка встановлених прав доступу до файла скрипта (-rwxr-xr-x див.
завдання):
ПЕРЕВІРЕНО!
---
Перевірка розміру файла скрипта:
ПЕРЕВІРЕНО!
---
Перевірка першого рядка Bash скрипта:
ПЕРЕВІРЕНО! Перший рядок скрипта: #!/bin/bash
---
```

```
Перевірка загального синтаксису скрипта:
ПЕРЕВІРЕНО!
---
Перевірка використаних синтаксичних констркцій Bash:
ПЕРЕВІРЕНО! Конструкція: if ...; then ...; fi
ПЕРЕВІРЕНО! Конструкція: while ...; do ...; done
ПЕРЕВІРЕНО! Конструкція: until ...; do ...; done
ПЕРЕВІРЕНО! Конструкція: for ... in ...; do ...; done
ПЕРЕВІРЕНО! Конструкція: for ((...)); do ... done
ПЕРЕВІРЕНО! Конструкція: function ...() { ... }
---
Перевірка скрипта статичним аналізатором коду:
ПЕРЕВІРЕНО!
---
Перевірка роботи скрипта (приклад див.: pzpi23-task1_example):
---
РЕЗУЛЬТАТИ ПЕРЕВІРКИ
УСПІШНИХ ТЕСТІВ: 14
НЕВДАЛИХ ТЕСТІВ: 0
ПРАВИЛЬНИЙ STDOUT:      625
НЕПРАВИЛЬНИЙ STDOUT:    0
ПРАВИЛЬНИЙ EXIT_CODE:   625
НЕПРАВИЛЬНИЙ EXIT_CODE: 0
ПРАВИЛЬНИЙ STDERR:      625
НЕПРАВИЛЬНИЙ STDERR:    0
---
ОЦІНКА ЗА КОД: 100 ( ./pzpi-23-2-sytnyk-yehor-task1 без перевірки на
плагіат )
---
```

## 1.3 Висновки

Під час виконання даної практичної роботи я навчився створювати сценарії автоматизації засобами Bash (Bourne again shell).

# ДОДАТОК А

Повний текст розробленого сценарію

```
#!/bin/bash

if (( $# != 2 )); then echo "Not enough/Too many arguments" >&2; exit 1; fi
if (( $1 < 8 )); then echo "Height must be > 7" >&2; exit 2; fi
if (( $2 < 7 )); then echo "Width must be > 6" >&2; exit 3; fi

h=$(( ($1 % 2 != 0) ? $1 - 1 : $1 ))
w=$(( ($2 % 2 == 0) ? $2 - 1 : $2 ))
lh=$(( $h/2 - 1 ))

if (( ($w-(2*$lh-1) != 2) )); then echo "Can't draw the tree with provided arguments" >&2; exit 4; fi


ch='*'

draw_layer() {
        for ((i=$1; i <= lh; i++)); do
                printf "%*s" "$(( (w - (2*i-1)) / 2 ))"
                printf "%*s\n" "$(( 2*i-1 ))" | tr ' ' "$ch"

                ch=$( [ "$ch" = '*' ] && echo '#' || echo '*' )
        done
}

draw_stem() {
        for i in {0..1}; do
                printf "%$(( (h - 3) / 2 ))s"
                printf "%3s\n" | tr ' ' "#"
        done
}

draw_snow() {
        printf "%${w}s\n" | tr ' ' "*"
}

draw_layer 1
draw_layer 2
draw_stem
draw_snow

i=0
while [[ $i -ne 1 ]]; do i=$((i+1)); done
until [[ $i -eq 2 ]]; do i=$((i+1)); done

```