BSc (Hons) in Computer Science
BSc (Hons) in computer systems Eng.
Year 1

Lab 06 – Top-Down Design with Functions

SE1012 – Programming Methodology **Semester 2, 2023**

Use VMWare horizon Linux machines for the lab sessions:

1. Use to the following link to access VLab : vlab.sliit.lk
2. Use the following login credentials to log into the virtual machines.

Username: sliit
Password:12345

Top-down design, also known as stepwise design or functional decomposition, is a software design approach where you break down a complex problem or system into smaller, manageable sub-problems or modules. Each module can then be further divided into smaller modules until you have a set of easily solvable tasks. This approach helps in organizing and structuring your code in a clear and understandable manner.
In the context of programming in the C language, top-down design involves the following steps:

1. **Problem Understanding:** Clearly understand the problem you're trying to solve. Identify the main components and functionalities the program should have.
2. **High-Level Design:** Create an outline or pseudocode of the main program's structure. Define the major functions or modules that will be needed. These functions represent the high-level tasks that the program needs to perform.
3. **Module Decomposition:** Break down each major function/module from the high-level design into smaller sub-functions/modules. Each module should have a specific, well-defined purpose. This continues until you have small enough modules that can be easily implemented.
4. **Function Prototypes:** Write function prototypes for each of the modules you've decomposed. Function prototypes declare the function's name, return type, and parameters without implementing the actual function logic. This allows you to define the interaction between different modules before writing their actual code.
5. **Implementation:** Begin implementing the lowest-level modules first. These modules should be self-contained and perform a specific task. Implement one module at a time, testing each one as you go to ensure that it works correctly.
6. **Integration:** As you complete the implementation of each module, integrate it into the higher-level modules that use it. This step-by-step integration ensures that the whole program comes together smoothly.

Example

```
#include <stdio.h>

// Function prototypes
float calculateAverage(float a, float b, float c);

int main() {
    float num1, num2, num3;
    printf("Enter three numbers: ");
    scanf("%f %f %f", &num1, &num2, &num3);

    float average = calculateAverage(num1, num2, num3);
    printf("Average: %.2f\n", average);

    return 0;
}

// Function to calculate the average of three numbers
float calculateAverage(float a, float b, float c) {
    return (a + b + c) / 3.0;
}
```

Task 1: Menu-Driven Calculator for Factorials, Combinations, and Permutations

Design a menu-driven calculator program in C that can perform calculations for factorials, combinations (nCr), and permutations (nPr). Implement separate functions for each operation, ensure continuous operation until the user decides to exit, and incorporate input validation using break statements. Follow the specifications below to create your program:

1. Create functions for each operation:
   - **calculateFactorial**: This function should take an integer as an argument and return the factorial of that number.
   - **calculateCombination**: This function should take two integers, n and r, as arguments and return the combination (nCr) of n and r.
   - **calculatePermutation**: This function should take two integers, n and r, as arguments and return the permutation (nPr) of n and r.

2. Display a menu to the user with the following options: (Use a switch-case structure to perform the selected operation based on the user's choice.)

   Menu:
   1. Calculate Factorial
   2. Calculate Combination (nCr)
   3. Calculate Permutation (nPr)
   4. Exit

3. Create a while loop that continues to display the menu and perform calculations until the user chooses the "Exit" option.

4. Validate inputs using break statements:
   - For each operation, use input validation to ensure that the entered values are valid (e.g., non-negative integers for factorials, and n >= r >= 0 for combinations and permutations). If invalid input is provided, display an appropriate error message and prompt the user to enter the values again using a loop and a break statement.

```
#include <stdio.h>

// Function prototypes
int calculateFactorial(int n);
int calculateCombination(int n, int r);
int calculatePermutation(int n, int r);

int main() {
    int choice;
    while (1) {
        printf("\nMenu:\n");
        printf("1. Calculate Factorial\n");
        printf("2. Calculate Combination (nCr)\n");
        printf("3. Calculate Permutation (nPr)\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: {
                int n;
                printf("Enter a number: ");
                scanf("%d", &n);
                printf("Factorial  of  %d  is  %d\n",  n,
calculateFactorial(n));
                break;
            }
            case 2: {
                int n, r;
                while (1) {
                    printf("Enter values for n and r (n >=
r >= 0): ");
                    scanf("%d %d", &n, &r);
                    if (n >= r && r >= 0) {
                        printf("nCr       is       %d\n",
calculateCombination(n, r));
                        break;
                    } else {
                        printf("Invalid input. Please try
again.\n");
                    }
                }
                break;
            }
            case 3: {
                int n, r;
                while (1) {
                    printf("Enter values for n and r (n >=
r >= 0): ");
                    scanf("%d %d", &n, &r);
                    if (n >= r && r >= 0) {
                        printf("nPr       is       %d\n",
calculatePermutation(n, r));
                        break;
                    } else {
                        printf("Invalid input. Please try
again.\n");
```

```
                    }
                }
                break;
            }
            case 4:
                printf("Exiting the program.\n");
                return 0;
            default:
                printf("Invalid choice. Please select a
valid option.\n");
        }
    }

    return 0;
}

// Function to calculate factorial
int calculateFactorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}

// Function to calculate combination (nCr)
int calculateCombination(int n, int r) {
    return calculateFactorial(n) / (calculateFactorial(r)
* calculateFactorial(n - r));
}

// Function to calculate permutation (nPr)
int calculatePermutation(int n, int r) {
    return calculateFactorial(n) / calculateFactorial(n -
r);
}
```

Task 2

Write two functions, one that displays a triangle and one that displays a rectangle. Use these functions to write a complete C program from the following outline:

```
int main(void) {

/* Draw triangle. */
/* Draw rectangle. */
/* Display 2 blank lines. */ /* Draw triangle. */
/* Draw rectangle. */

}
```

Use the below implemented functions in a program that draws a rocket ship (triangle over rectangles over intersecting lines), a male stick figure (circle over rectangle over intersect- ing lines), and a female stick figure (circle over triangle over intersecting lines) standing on the head of a male stick figure. Write function skip_5_lines and call it to place five blank lines between drawings.

Draws a circle

```
draw_circle(void)
{
      printf("    *    \n");
      printf(" *     * \n");
      printf("   * *   \n");
}
```

Draws intersecting lines

```
void
draw_intersect(void)
{
      printf("  / \\  \n"); /* Use 2 \'s to print 1 */
      printf(" /   \\ \n");
      printf("/     \\\n");
}
```

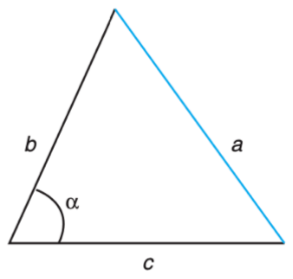
Draws a base line

```
void draw_base(void) {

printf("-------\n");

}
```

```
Draws a triangle

void draw_triangle(void) {

draw_intersect();

draw_base();

}
```

Task 3

Write a code for the following task: If we know the lengths of two sides (*b* and *c*) of a triangle and the angle between them in degrees (), we can compute the length of the third side (*a*) using the following formula.



$$a^2 = b^2 + c^2 - 2bc\cos$$

Hint: To use the math library cosine function (cos), we must express its argument angle in radians instead of degrees. To convert an angle from degrees to radians, we multiply the angle by /180. If we assume PI represents the constant , the C assignment statement that follows computes the unknown side length:

```
a = sqrt(pow(b,2) + pow(c,2)- 2 * b * c * cos(alpha * PI /
180.0));
```

Submission

Submit your answers for reach task into GitHub Classroom.

https://classroom.github.com/assignment-invitations/9bff8925c9642fa4ca488ec7c8d12926

Create folders for each task and submit your answers.