Swinburne University of Technology Sarawak

COS10009 Introduction to Programming

Modular Programming – Procedure / Function / Method (Lab 03)

Pass Task 3.1 Customized Text Message Generator

Task: Write a program that will generate a customized greetings text message according to the duration of membership of each club member.

To Do:

Functions allow you to 'chunk' together related lines of code. Functions can receive data as arguments. Pass Task 3 Resources folder contains a complete program (Greetings.c) that generate customized message according to the input from the user. However, all the statements in this program are written inside the main function, and many statements are repeated for many times throughout the program. Your task is to identify and extract the repeated code, define a function for the repeated code, set up any necessary parameters and call the function accordingly.

Pass Task 3.2 Hello User with Functions

Task: In this task we change our Reading and Writing (Hello User) program to use functions to prompt and read in values.

To Do:

Function not only can receive data as arguments, it can also allow you to return a value. The functions we are going to use in this task are contained in the file input_functions.h (see the resources for this task). We are going to modify our previous program to use these functions.

If you recall our Reading and Writing (Hello User) program did the following:

- ask the user to enter their name, age, and the current year,
- calculate the year the user was born, and
- output a greeting message.

1. Download the input_functions.h and input_functions.c (see the Resources for this task). This contains some useful functions you can use to read values from the user.
2. Copy input_functions.c and input_functions.h into the folder next to your hello_user.c file (from Pass Task 1.2).
3. Create a hello_user_with_functions program based on the program start code shown below. Compile with "gcc hello_user.c input_functions.c -o hello.exe"
4. Fill in 3.2 Question sheet on the data type your are using.

```
#include <stdio.h>
#include <time.h>
#include "input_functions.h"
#define INCHES 0.393701 // This is a global constant

void main()
{
   char name[256];
   char family_name[256];
   int year_born;

   /** Replace with a function from input_functions.h **/
   printf("Your name is %s !\n", name);
   ....
```

Pass Task 3.3 Guess Number Game

Task: Write a program that generates a random number between the interval (0, 100). It prompts the user to guess the random number generated. Procedures / functions/ methods are a great tool for capturing the instructions needed to perform a task, but sometimes you need to be able to capture the instructions needed to process a value. By using function, you can now create artefacts to encapsulate the steps needed to process a value.

To Do:

Functions allow you to build modular programs that take parameters and return a result.
To explore this topic, we will implement the following:

1. Generates a random number between the interval (0, 100).
2. Prompts the user to guess the random number by entering an integer (call read_integer() function from input_functions.h).
3. Call function check() to determine if the integer input by user is smaller, bigger or equal to the random number. This function should receive two integers as arguments (random number and input by user), and return a string indicating the results of the comparison. (You will need to write this function)
4. The string returned by function check() will be shown on screen to tell the user if the guess is equal, larger or less than the random number.
5. Repeat step 2 to step 5 until the user has successfully guessed the random number correctly
6. Call the read_boolean(prompt) function from input_functions.h to prompt user if he/she would like to continue with a new game? Go back to step 1 and repeat the whole program if the user enters “Yes”, otherwise end the program.

Credit Task 3.1 Code Reusability / Eliminating Repeated Code

Task: In this task, you are required to eliminate repeated code, apply code reusability and implement modularity in your program.

To Do: Credit Task 3 Resources folder contains an example program showing un-normalised code. Extract the repeated code, define a function for the repeated code, set up any necessary parameters and call the function as necessary.