

Lab 05

**Nested Loops and Control Flow using ‘break’ and ‘continue’**

**Muhammad Mujtaba**

**CMD ID: 540040**

[mmujtaba.bese25seecs@seecs.edu.pk](mailto:mmujtaba.bese25seecs@seecs.edu.pk)

**Class:** BESE 16B

**Batch:** 2k25

# Task 1 [CLO 1]:

## CODE:

#include <iostream>

int main()

{

int min\_x = 0;

int max\_x = 5;

int min\_y = 0;

int max\_y = 5;

for (int i = min\_x; i <= max\_x; i++)

{

for (int j = min\_y; j <= max\_y; j++)

{

std::cout << "(" << i << ", " << j << ") ";

}

std::cout << "\n";

}

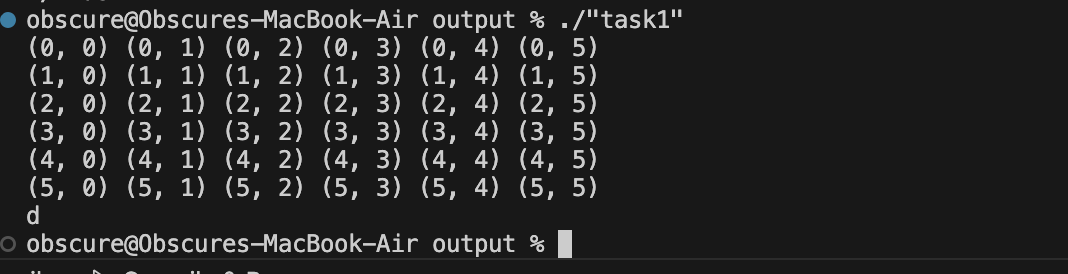
std::cin.ignore();

std::cin.get();

return 0;

}

## OUTPUT:



# Task 2 [CLO 1]:

## CODE:

#include <iostream>

#include <math.h>

#include <iomanip>

// utility for printing spaces

inline void print\_spaces(int count, bool debug = false)

{

// `count + 1` is used as std::setw sets the total width including the first charcter of next output field

std::cout << std::setw(count + 1) << std::setfill(debug ? '.' : ' ');

}

void numberSquarePattern(int size);

void pyramidPattern(int lines\_y);

void rightAlignedPyramidPattern(int lines\_y, char toPrint);

void rightTrianglePattern(int size);

int main()

{

std::cout << "\n----------------------------------\n\n";

numberSquarePattern(6);

std::cout << "\n----------------------------------\n\n";

pyramidPattern(5);

std::cout << "\n----------------------------------\n\n";

rightAlignedPyramidPattern(5, '3');

std::cout << "\n----------------------------------\n\n";

rightTrianglePattern(6);

std::cout << "\n----------------------------------\n\n";

// ignore previous input

std::cin.ignore();

std::cin.get();

return 0;

}

void numberSquarePattern(int size)

{

// size is the size of square

// at every place inside the region "size x size"

// it only prints the character if it is at outer edge

for (int i = 1; i <= size; i++)

{

for (int j = 1; j <= size; j++)

{

// if first or last line, print all line

if (i == 1 || i == size)

std::cout << j;

else

{

// if first or last item IN line, print "the number" otherwise print " "

if (j == 1 || j == size)

std::cout << j;

else

std::cout << " ";

}

}

std::cout << "\n";

}

}

void pyramidPattern(int lines\_y)

{

// lines\_y is how many lines to print in vertical direction

const int height = lines\_y \* 2; // height is two times "lines\_y" as we skip even number lines

// no cost as compiler will optimize this away

// first calculate how many spaces are required for the character

// then print the character just like a right facing triangle

// CAN BE THOUGHT AS

// first print right facing triangle then slant it

// but processes on each line seperately

int spaces\_count;

for (int i = 1; i <= height; i += 2) // `\* 2` and `+= 2` as we only need to print even numbers

{

spaces\_count = height - i;

print\_spaces(spaces\_count);

for (int j = 1; j <= height; j++)

{

if (j <= i)

std::cout << j << " ";

}

std::cout << "\n";

}

}

void rightAlignedPyramidPattern(int lines\_y, char toPrint)

{

// if even number of lines are being printed we will print the middle line twice

const int lines\_y\_half\_ceiling = std::ceil(lines\_y / 2.f); // pre calculate

// will be optimzed away by compiler

// same priciple as above

// calculate spaces then print character

int spaces\_count;

for (int i = 1; i <= lines\_y; i++)

{

// print spaces

spaces\_count = std::abs(lines\_y\_half\_ceiling - i) \* 2;

if (lines\_y % 2 == 0 && i > lines\_y\_half\_ceiling)

spaces\_count -= 2;

print\_spaces(spaces\_count);

// print characters

for (int j = 1; j <= lines\_y; j++)

{

// before middle point

if (i <= lines\_y\_half\_ceiling)

{

// print triangle facing left

if (j <= i)

std::cout << toPrint << " ";

}

else

{

// print triangle facing right

if (j >= i)

std::cout << toPrint << " ";

}

}

std::cout << "\n";

}

}

void rightTrianglePattern(int size)

{

int spaces\_count;

for (int i = 1; i <= size; i++)

{

spaces\_count = size - i;

print\_spaces(spaces\_count \* 2);

for (int j = 1; j <= size; j++)

{

// print triangle facing left

if (j <= i)

std::cout << j << " ";

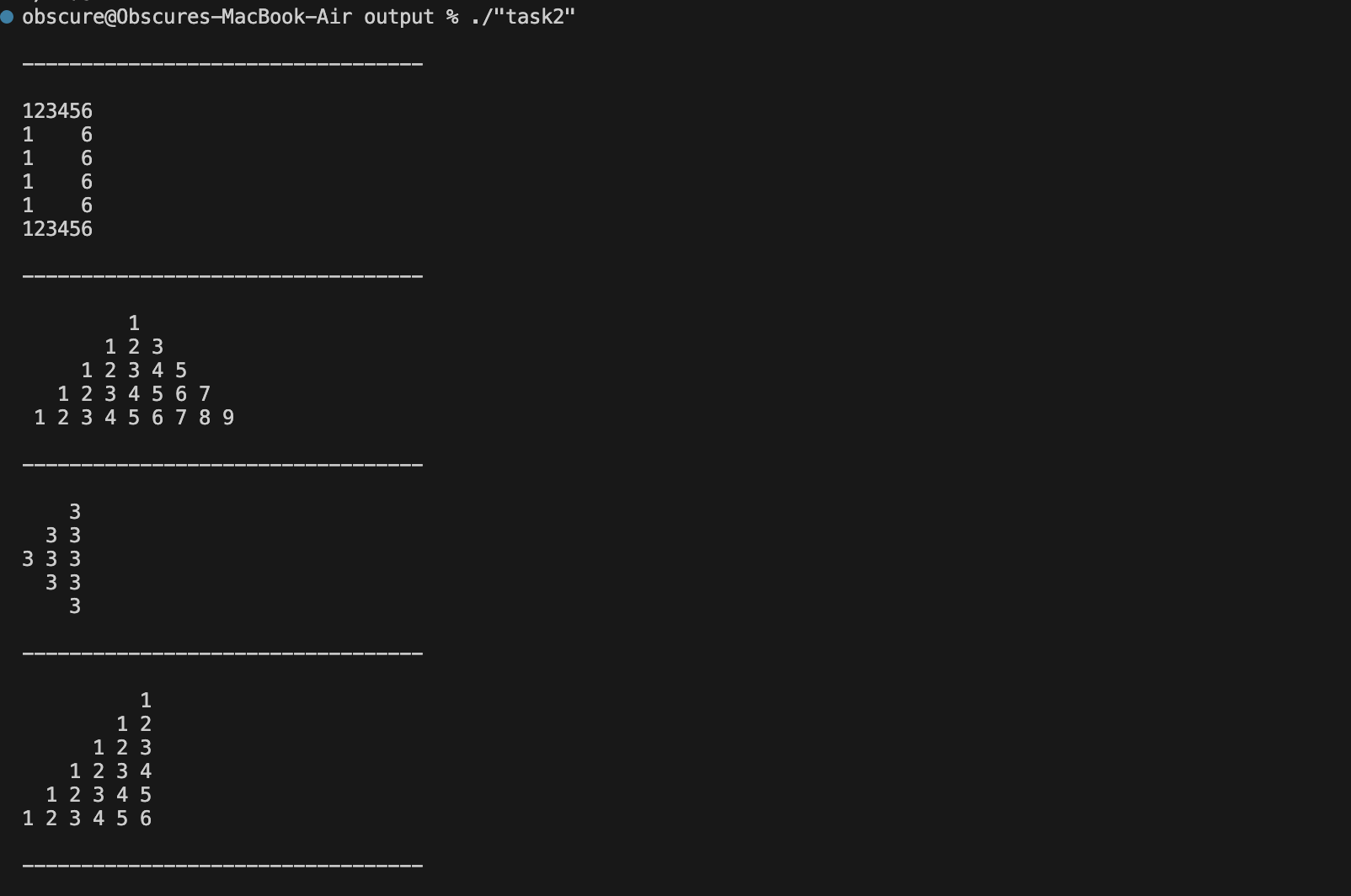
}

std::cout << "\n";

}

}

## OUTPUT:



// settings `DEFAULT` //

inline void print\_spaces(int count, bool debug = false)

numberSquarePattern(6);

pyramidPattern(5);

rightAlignedPyramidPattern(5, '3');

rightTrianglePattern(6);

A screenshot of a computer

AI-generated content may be incorrect.

// settings `ALTERED INPUT` //

inline void print\_spaces(int count, bool debug = true)

numberSquarePattern(8);

pyramidPattern(3);

rightAlignedPyramidPattern(7, 'O');

rightTrianglePattern(9);

# Task 3 [CLO 2]:

## CODE:

#include <iostream>

bool isPrime(int number, bool &primeError);

int main()

{

int count;

std::cout << "Print how many numbers you will enter: ";

std::cin >> count;

int number; // the number

bool primeFound = false; // is primeNumber Found

bool primeError = false; // is error detected

while (count > 0)

{

count--;

std::cout << "Enter a postiive non zero number: ";

std::cin >> number;

if (isPrime(number, primeError))

{

std::cout << "You entered a prime number.\n";

primeFound = true;

break;

}

if (primeError)

std::cout << "You entered an invalid number.\n";

}

if (count <= 0 && !primeFound)

std::cout << "You ran out of tries. No Prime Number in List.\n";

std::cin.ignore();

std::cin.get();

return 0;

}

bool isPrime(int number, bool &primeError)

{

// if number is negative or zero then set primeError to true

if (number <= 0)

{

primeError = true;

return false;

}

if (number == 1 || number == 2)

return true;

primeError = false;

// check numbers from 2 to (number / 2) + 1 for divisibility

for (int i = 2; i < int(number / 2.f) + 1; i++)

{

if (number % i == 0)

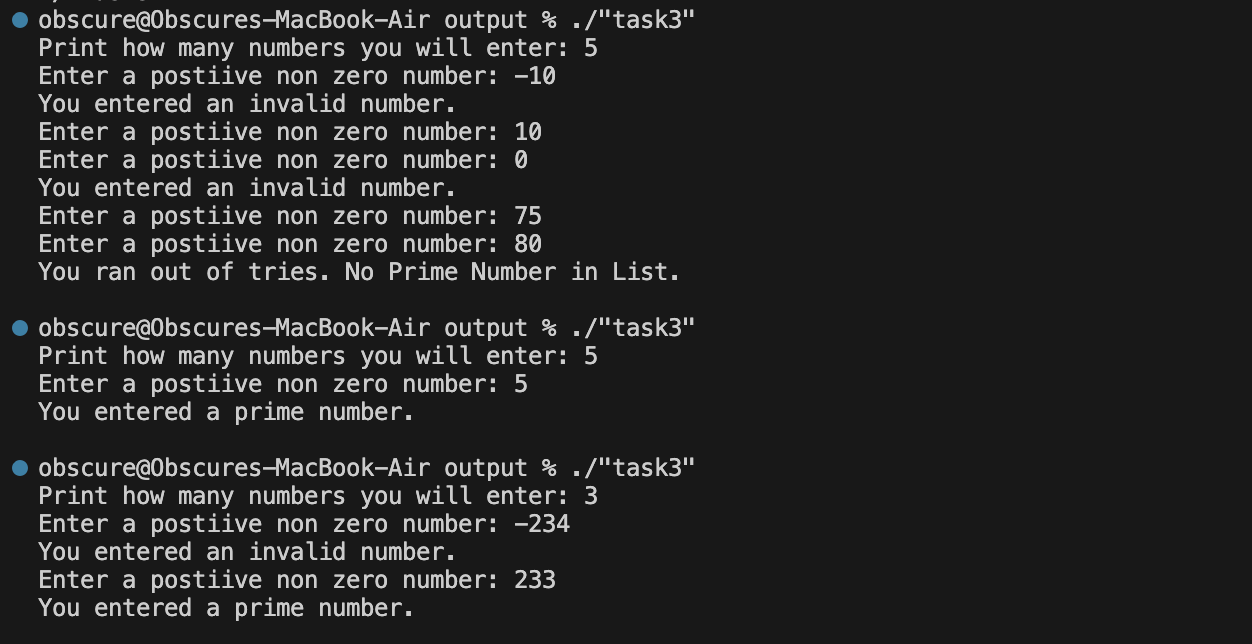
return false;

}

return true;

}

## OUTPUT:



# Task 4 [CLO 2]:

## CODE:

#include <iostream>

int main()

{

float sum = 0;

int count = 0;

int tmp\_input;

std::cout << "Enter -1 to exit. \n\n";

while (true)

{

std::cout << "Enter marks: ";

std::cin >> tmp\_input;

// if input -1 then exit

if (tmp\_input == -1)

{

std::cout << "\* Exiting... \n\n";

break;

}

// check if between 0 and 10

if (tmp\_input <= 0 || tmp\_input >= 100)

{

std::cout << "\* Invalid number skipped \n";

std::cout << "\* Enter a number between 0 and 100. \n\n";

continue; // take another numb from

}

sum += tmp\_input;

count++;

}

std::cout << "Average marks: " << sum / count << "\n";

std::cin.ignore();

std::cin.get();

return 0;

}

## OUTPUT:

