```c
#include <stdio.h>
#include <math.h>

/// NOTE --> DO NOT EXECUTE CODE IN THIS FILE, CREATE A NEW PROJECT AND COPY PASTE THE GIVEN SECTION TO VIEW THE EXAMPLE

/// TOPIC 1: DECISION INSTRUCTIONS
/* By assigning values to TRUE and FALSE (1&0) we can instruct the CPU to selectively execute functions
based on the conditions reported by the user --> Decisions using logical expressions

LOGICAL EXPRESSIONS

== equals to
!= Not equal to
>= Greater than or equal to
<= Less than or equal to

!() infers the opposite of the boolean expression used

Decisions can be accessed by the CPU through the use of 3 basic instructions

1. if/else
--> The basic decision instruction with an instruction block to be executed if a logical expression is
    TRUE and an optional insctruciton block if the expression is FALSE

2. ELSE-if
--> The short-hand form multiply nested decision instructions
    In other words, if the condition relies on more than one single TRUE statement,
    then the use of the else-if instruction can be added to the if/else statement

3. Switch
--> OPTIONAL USE IN THIS COURSE
    an alternate form of the ELSE-IF statement
*/

/// THE IF DECISION INSTRUCTION

/* THE IF DECISION INSTRUCTION allows us to isolate C insturcitons to have the program execute these
instrucitons only if a logical
expression is TRUE (==1)

SYNTAX

{
int x;

if (logical expression)             NOTE THE LACK OF THE USE ';' FOR THE PURPOSE OF NOT ENDING THE INSTRUCTION
JUST YET
    {
        instruction 1;              A logical expression can be as simple as if (x==2) {instruction}
        instruction 2;              when CPU finds that integer x is equal to 2 aka TRUE, then instruction is
executed
        .
        .
        .
        instruction 3;
    }

}
*/

/// Make sure to use the proper indentation throughout the program's syntax to ensure clean practice

/// Refer to the flow chart below for a graphical depiction of the if decision instruction

/*
```

```
                Declaration of Variables
                            |
                            |
                            |
                            |
            if (logical expression according to variable)-------- TRUE --------> Instruction in bracket is
executed
                            |                                                                   |
                            |                                                                   |
                            |                                                                   |
                            |                                                                   |
                        FALSE                                                                   |
                            |                                                                   |
                            |                                                                   |
                            |                                                                   |
                        Next function ------------------------------------------------------------|


*/

/// COPY PASTE EXAMPLE FOR CLEAR PROGRAM


void main()
{
    double a=0.0, b=1.0;
    if(a< 10.0)
    {
        a=a+1.0;
        b=b+1.0;
    }
    printf("%lf = a\n%lf = b", a, b);
}

/// rather than using just a nested expression, we can use a logical operator

void main()
{
    double a=0.0, b=1.0;
    if(a< 10.0 && a > 1.0) /// in this case the expression is false since a is less than 1.0
    {
        a=a+1.0;
        b=b+1.0;
    }
    printf("%lf = a\n%lf = b", a, b);
}

/// IF/ELSE DECISION INSTRUCTION

/* The if/else instruction allows us to specify a set of instructions to be executed if the logical expression
is TRUE
as well as a set of instructions if the expression is FALSE


SYNTAX

{
int x;

if (logical expression)             NOTE THE LACK OF THE USE ';' FOR THE PURPOSE OF NOT ENDING THE INSTRUCTION
JUST YET
    {
        instruction 1;              A logical expression can be as simple as if (x==2) {instruction}
        instruction 2;              when CPU finds that integer x is equal to 2 aka TRUE, then instruction is
executed
        .
```

```
        .
        .
        instruction 3;
    }
else                                if FALSE aka x not equal to 2, then else instruction is executed
    {
        instruction 1;
        instruction 2;
        .
        .
        .
        instruction 3;
    }

}

                Declaration of Variables
                                |
                                |
                                |
                                |
            if (logical expression according to variable)-------- TRUE --------> Instruction in bracket is
executed
                                |                                                                   |
                                |                                                                   |
                                |                                                                   |
                                |                                                                   |
                        FALSE                                                                       |
                                |                                                                   |
                                |                                                                   |
                                |                                                                   |
                                else instruction---------------- next instruction ------------------|


 */

 /// COPY PASTE EXAMPLE

 {
     int grade = 95;

     if (grade >= 90){              /// This is executed in the case where grade is greater than or equal to 90
        printf("You get an A+");
     }
     else{                          /// If the above statement is false, then this will be executed
        printf("Less than 90\n");
        printf("Less than A+\n");
     }
 }

 /// what would happen if grade was made to be 85?

 /// The following example depicts the use of all 3 of the decision instructions (if, else-if and else)

 {

     int choice = 0;
     if(choice == 1){
        printf("First choice selected");
     }
     else if(choice == 2){
        printf("Second choice selected");
     }
     else{
        printf("Wrong selection");
     }
```

```
}

/// find the error in the following code. How would you correct it?

{
    int grade = 90;

    if (grade >=85){
       printf("You have an A\n");
    }
    else if(grade >=90){
       printf("you have an A+\n");
    }
}

/// ANSWER --> If grad eis equal to 90, logical conflict between if and else if statement
/// switch position of the logical expressions and it will be correct, or change the boolean operators


/// TOPIC 2: LOOP INSTRUCTIONS

/*
Most programs require repeated instructions in order to execute multiple functions without constant input
a loop can be one of the two following options

1. Definite repetition loop --> if we know in advance the number of repetitions

2. Indefinite repetition loop --> Unknown number of repetitions -> based on user input

The number of repetitions:

1. in a definite repetition loop is controlled by a COUNTER

2. in an indefinite loop, the repetition is controlled by a SENTINEL

*/

/// COUNTER DEFINITE LOOP
{
 int count;
for( count=1; count<=100; count++)
    printf("%d",count);
} /// FOR THIS WE KNOW THAT WE WILL GO FROM 1 TO 100 CONTINUOUSLY

/// SENTINEL INDEFINITE LOOP
{
 int reverseNumber=0;
int number=12345;
    while(number>0)
    {
        reverseNumber= (reverseNumber*10)+ number%10;
        number/=10;
    }
    printf("Reverse Number is: %d\n", reverseNumber);
}
/// The function will continue to execute a number of times until the statement is no longer true in the while
logical expression

///LOOP INSTRUCTIONS IN C

/* There exists three repetition instructions

1. "while" instruction
2. "do/while" instruction
3. "for" instruction
```

```
*/

/// while REPETITION INSTRUCTION

/*

while (logical expression)          note the resemblance to the if/else syntax
{
    instruction 1;
    instruction 2;
    .
    .
    .
    instruction 3;
}

if the logical expression remains true, then the loop will continue to execute
repetition loops do not always have to be executed, sometimes they can be completely ignored if they are
set up to extract improper user input.

SYNTAX

                Declaration of Variables
                            |
                            |-----------------------Repeat-----------------------------------------|
                            |                                                                      |
                            |                                                                      |
            while (logical expression according to variable)-------- TRUE --------> Instruction in bracket is
executed
                            |
                            |
                            |
                            |
                        FALSE
                            |
                            |
                            |
                            Next function


Note the SIMILARITY to the if/else statement logic, the only difference is that the CPU will run the logic
expression
after every TRUE logical conclusion based on the instructions in the brackets, once the expression is FALSE,
then
we continue onto the next function

*/

/// example of a definite repetition loop using while instruction

{
    int ctr;
    ctr=1;
    while (ctr<=10)
    {
        printf("Iteration number: %d\n", ctr);
        ctr=ctr++ /// same as using the expression ctr=ctr+1 "incrementation"
    }
    printf("ctr upon exit of the loop: %d\n",ctr);
}
/// can you identify the COUNTER in this loop? why is it a COUNTER?

/// example of an indefinite repetition loop using the while instruction

{
    int sentinel;
```

```c
    sentinel=1;
    while (sentinel !=0)
    {
        printf("Enter 0 to exit the loop: \n");
        scanf("%d", &sentinel);
    }
    printf("we are out\n");
}

/// What's the difference between this loop and the one above?

/// do/while REPETITION INSTRUCTION

/* the do/while repetition instruction mimics that of the while instruction, with the sole difference
being that when presented with the do/while instruction, the CPU will execute the instructions at least once
before
checking the logic

SYNTAX

do
{
    instruction 1;
    .
    .
    .
    .
    instruction n;
}
while (logical expression);

NOTE that the logical expression exists at the end of the instruction block, this is due to the fact that CPU
will
execute the instructions at least once before checking the logic and moving on.

FLOW CHART

                    Declaration of Variables ----------------------|
                                |                                  |
                                |                                  |
                               do                                  |
                                |                                  |
                                |                                  |
                                |                                  |
                        instructions                               |
                                |                                  |
                                |                                  |
                                |                                  |
                        while (logic expr)----------TRUE--------|
                                |
                                |
                               FALSE
                                |
                                |
                        Next instruction

To avoid an infinite loop, we must modify at least one variable COUNTER or SENTINEL within the instruction block
such that the logical expression in the while declaration is eventually FALSE

*/

/// Example of a definite repetition loop using do/while instruction

{
    int ctr;
    ctr=1;
```

```c
    do
    {
        printf("Iteration %d\n", ctr);
        ctr=ctr++ /// remember what this means
    }
    while (ctr<=10);
}

/// the program will execute the instructions once before going through the logic check in the while argument bracket

{
    int sentinel;
    sentinel = 1; /// is this line required?
    do
    {
        printf("enter 0 to exit the loop\n");
        scanf("%d", &sentinel);
    }
    while (sentinel !=0);

}
/* The variable assignement above is not required since the do/while loop will run once before checking the logic.
therefore we will assign a variable to sentinel through the instructions, it only matters when sentinel == 0
*/

/// for REPETITION INSTRUCTION

/* The for looping instruction is a DEFINITE repition instruction that makes use of the COUNTER

SYNTAX for a single C instruction repeat

{
    for(expression 1; expression 2; expression 3)
        instruction;
}

SYNTAX for a multiple C instruction repeat

{
    for(expression 1;expression 2; expression 3)
    {
        instruction 1;
        .
        .
        .
        .
        instruction n;
    }

}
The three expressions in the for loop have the following roles:

1. EXPRESSION 1 --> ARITHEMATIC expression to initialize the counter ex) x = 1
2. EXPRESSION 2 --> LOGICAL expression to test the counter against its final value ex) x<= 10
3. EXPRESSION 3 --> ARITHEMATIC expression that modifies the value of the counter ex) x=x++ ... incrementation of x

FLOWCHART


                            Declaration of Variables
                                        |
                                        |
                                Expression 1; (Counter initialization)
```

```
                                    |
                                    |-----------------------------------------|
                                    |                                         |
                                Expression 2;----- TRUE ------------------ Expression 3; (Counter Modifier)
                                (Logical test)
                                    |
                                    |
                                    |
                                   FALSE
                                    |
                                    |
                                    |
                                Next instruction

                                */
```

```
/// Operation of a for instruction program

/*
A Program execution flows first into expression 1 which is evaluated once to initialize the loop counter via
expression 2

B expression 2 is then evaluated --> logical test

C if expression 2 is FALSE, the the instruction block is skipped

D if expression 2 is TRUE, then the instruction block is executed

E once completed, expression 3 modifies the counter

F back to step B

*/

/// What is a NESTED LOOP?

/* many of the loops above can be combined with other loops, where the other loop lies within the instruction
block
of the primary loop.

this inception of loops is referred to as NESTED LOOPS --> the loop within the loop

*/

/// Example of a nested loop

{
    double x,y,z;
    for (x=-10.0; x<11.0; x=x+2.0)
    {
        for (y=-10.0; y<11.0; y=y+2.0)
        {
            z=x*x+y*y;
            printf("z=%f\n",z);
        }
    }
}

/// How many variables will be displayed on the console in this case?

/// answer (20 variables for y in each of the 20 variables of x) therefore 20*20 --> 400

/// It is a big TABOO to use the "goto" and "break" instructions in C

/* don't worry about what they're for, they just interrupt a loop without respecting the logical expressions
presented */
```