// C201_PreTest.cpp : main project file.

#include "stdafx.h"
#include <iostream>
#include <string>

using namespace std;

// Forward Declaration
void Problem_1(void);
void Problem_2(void);
void Problem_3(void);
void Problem_4(void);
void Problem_5(void);
void Problem_6(void);
void Problem_7(void);
void Problem_8(void);
void Problem_9(void);
void Problem_10(void);
void Problem_11(void);
void Problem_12(void);
void Problem_13(void);
void Problem_14(void);
void Problem_15(void);
int Problem_16(void);
void Problem_17(void);
void Problem_18(void);
void Problem_19(void);
void Problem_20(void);
int main(void)
{
    cout << "C201 Pre Test" << endl;
    //Problem_1();
    //Problem_2();
    //Problem_3();
    //Problem_4();
    //Problem_5();
    //Problem_6();
    //Problem_7();
    //Problem_8();
    //Problem_9();
    //Problem_10();
    //Problem_11();
    //Problem_12();
    //Problem_13();
    Problem_14();
    Problem_15();
    cout << Problem_16() << endl;  // returns an integer
    Problem_17();
    Problem_18();
    Problem_19();
    Problem_20();

    getchar();  // you can also use system("pause");
    // but that is less efficient as you are
    // executing a call to run another program called pause in DOS
    return (0);
}
void Problem_1(void)
{
    cout << "\n--------------------------------------------------------" << endl;
    cout << "1) Identify the proper data type to store "1234" " << endl;
    cout << "a) char    b) int    c) double    d) none of the above" << endl;
    cout << "............." << endl;
    // int data1 = "1234";
    // char data2 = "1234";
    // double data3 = "1234";

    // Solution
    char data4[] = "1234";   // we can use array of characters
    string data5 = "1234";   // we can use the string class, but then we need to #include <string>

    cout << "Stored in an array of characters: " << data4 << endl;
    cout << "Stored in a string object: " << data5 << endl;
}

void Problem_2(void)
{
    cout << "\n--------------------------------------------------------" << endl;
    cout << "2) A computer can assign a value to a ______________ " << endl;
    cout << "a) variable    b) data type     c) both" << endl;
    cout << "............." << endl;

    // solution:
    int var1 = 1;
    cout << "var1: " << var1 << endl;

    // what about something like this?
    typedef int newInt; // create a new datatype using typedef
    newInt var2 = 2;    // here we are still assigning a value to a variable not to a data type!
    cout << "var2: " << var2 << endl;
    cout << "here we are still assigning a value to a variable not to a data type!" << endl;
}
void Problem_3(void)
{
    cout << "\n---------------------------------------------------------------------" << endl;
    cout << "3) What values will be stored in var1, var2 and var3" << endl;
    cout << "a) 12, 2, 2.0    b) 12, 2, 2.4    c) 12, 2.4, 2.4    d) None of the above" << endl;
    cout << "............." << endl;

    int var1 = 12, var2(0);
    double var3(0);

    var3 = var1 / 5;
    var2 = var3;       // Causes a warning because converting a double into int can be a loss of data

    cout << var1 << endl;
    cout << var2 << endl;
    cout << var3 << endl;

    cout << "Surprised?................." << endl;
    cout << "try dividing by 5.0 instead of 5" << endl;
    var3 = var1 / 5.0;
    var2 = var3;       // Causes a warning because converting a double into int can be a loss of data

    cout << var1 << endl;
    cout << var2 << endl;
    cout << var3 << endl;
}


void Problem_4(void) 
{
    cout << "\n--------------------------------------------------------" << endl;
    cout << "4)What is the output when inputs are 1 A " << endl;
    cout << "a) 2     b) 2 3     c) 2 3 Invalid " << endl;
    cout << " .........." << endl;

    int first;
    char second;
    cout << "Enter data for first and second: ";
    cin >> first >> second;

    switch (first) {
      case 1: 
        if (second == 'B') cout << "1 ";
        switch (second) {
          case 'A': cout << "2 "; // there is no Break here so it will also evaluated the next case!
          case 'B': cout << "3 "; break; // even though second = 'A', this case will also execute!
        }
        break;
      case 2: cout << "5 "; break;
      default: cout << "Invalid";
    }

    cout << "\n\nSurprized?............." << endl;
    cout << "try putting a break; in the line that evaluates case 'A' " << endl;

    getchar(); // consume the extra newline char when the user entered data using cin
}
void Problem_5(void)
{
    cout << "\n--------------------------------------------------------" << endl;
    cout << "5) During the execution of the above program segment, what happens when num = 9"
    cout << "a)0   b) 9     c) nothing is printed     d) infinite loop" << endl;
    int num;
    cout << "Enter num: ";
    cin >> num;
    do {
        cout << num % 10 << " ";
        num = num / 10;
    } while (num > 0);

    getchar(); // consume the extra newline char.
}

void Problem_6(void)
{
    cout << "\n--------------------------------------------------------" << endl;
    cout << "6) What is the output of the following code(take extra care)" << endl;

    int count = 0;
    while (count = 99)    // Note the use of single = sign (this is an
        count++;
    cout << "Count: " << count;

}

void Problem_7(void)
{
    cout << "\n--------------------------------------------------------" << endl;
    cout << "7) What is the content of the array A after the following C++ statements are executed
?" << endl;
    cout << "a)\[0 1 2 3 4\]     b)\[0 1 2 4 5\]      c)\[0 1.33 2.66 4 5.33\]       d)\[0 0 1 2 2\]        e) None" << endl;

    int A[5], i = 0;
    cout << "\nArray before Initialization: " << endl;
    for (i = 0; i < 5; i++) cout << A[i] << " ";
    for (i = 0; i < 5; i++)
        A[i] = i + (i / 3);
    cout << "\nArray after Initialization: " << endl;
    for (i = 0; i < 5; i++) cout << A[i] << " ";

}
void Problem_8(void)
{
    cout << "\n---------------------------------------------------------------------" << endl;
    cout << "8) Post - increment(n++) and Pre - increment(++n) operators will output same
results always. (true or false)" << endl;
    int n = 5;
    cout << "Before doing anything n is: " << n << endl;
    cout << "Printing n++ : " << n++ << endl;
    cout << "After printing n++ the new value of n is : " << n << endl;
    cout << "Printing ++n : " << ++n << endl;
    cout << "After printing ++n, the new value of n is : " << n << endl;
}
void Problem_9(void)
{
    cout << "\n---------------------------------------------------------------------" << endl;
    cout << "9) Constants and variables are functionally same in C++." << endl;
    int a = 5;
    const int b = 5;
    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    a = 6;
    // b = 6;  // This is an error since the LHS of the assignment can not be a constant.
}
void Problem_10(void)
{
    cout << "\n--------------------------------------------------------" << endl;
    cout << "10) Both precedence and associativity are same and are used in evaluating
expressions." << endl;
    // http://cs.stmarys.ca/~porter/csc/ref/cpp_operators.html

    // Adjacent Operators:
    // First we need to understand what is meant by the phrase adjacent operators.
    // It means two operators that are separated by a single operand, such as
    // the + and - in a + b - c or the * and ++ in *p++.

    // Precedence Rules:
    // The precedence rules of a language specify which operator is evaluated
    // first when two operators with different precedence are adjacent in an expression.

    // Associativity Rules:
    // The associativity rules of a language specify which operator is evaluated first
    // when two operators with the same precedence are adjacent in an expression.

    cout << "2 + 3 * 9 = " << 2 + 3 * 9 << endl;
    cout << "2 * 3 + 9 = " << 2 * 3 + 9 << endl;
    cout << "2 * (3 + 9) = " << 2 * (3 + 9) << endl;

    int n = 2;
    cout << "n was 2 before, what is the value of n after executing: n + (++n) ? " << n + (++n) << endl;

    int n = 2;
    cout << "n was 2 before, what is the value of n after executing: n + ++n ? " << n + ++n << endl;

    n = 2;
    cout << "n was 2 before, what is the value of n after executing: (n + 5 * 2) + (++n) ? " << (n + 5) * 2 + (++n) << endl;

    n = 2;
    if ((n == 3) || (n = 5)) // == goes first, then || then =
        cout << "n = " << n << endl;
    else
        cout << "n = " << n << endl;

    n = 2;
    if ((n = 3) || (n == 5))
        cout << "n = " << n << endl;

}
else
    cout << "n = " << n << endl;

n = 2;
if ((n == 3) && (n = 3))        // == goes first, then && then =
    cout << "n = " << n << endl;
else
    cout << "n = " << n << endl;

n = 2;
if ((n = 3) && (n == 3))
    cout << "n = " << n << endl;
else
    cout << "n = " << n << endl;

    // Operators that have the same precedence are bound to their arguments in the direction of
    // their associativity.
    // For example, the expression a = b = c is parsed as a = (b = c), and not as (a = b) = c because of
    // right-to-left associativity of assignment, but a + b - c is parsed (a + b) - c and not a + (b - c)
    // because of
    // left-to-right associativity of addition and subtraction.
void Problem_11(void)
{
    cout << "\n--------------------------------------------------------" << endl;
    cout << "11) An array can be used to store and manipulate a collection of data that is all of different data types." << endl;

    int myArray[5];
    myArray[0] = -1;
    myArray[1] = 'a';
    myArray[2] = 2.5;

    int i;
    for (i = 0; i < 5; i++)
        cout << "myArray[" << i << "] = " << myArray[i] << endl;

}

void Problem_12(void)
{
    cout << "\n--------------------------------------------------------" << endl;
    cout << "12) Preventing out-of-range index reference to arrays(or exceeding the index value), is entirely the responsibility of the programmer." << endl;
    int myArray[5];

    int i;
    for (i = 0; i < 10; i++)      // this can crash the program.
        myArray[i] =-1;

    for (i = 0; i < 10; i++)      // this can crash the program.
        cout << "myArray[" << i << "] = " << myArray[i] << endl;

}
void Problem_13(void)
{
    cout << "\n--------------------------------------------------------" << endl;
    cout << "13) During declaration, array size can be specified using a variable." << endl;

    int size = 5;

    //char myArray[size]; // most compiler don't allow this, since size is not a constant at compile time.
    cout << "No it can not" << endl;
}

//====================================================================================================
// ---------- Forward declaration
void function1(int x, int y);
void function2(int x, int &y);
void function3(int x, int *y);

void Problem_14(void)
{
    cout << "\n--------------------------------------------------------" << endl;
    cout << "14) Pass - by - value and Pass - by - reference can be identified based on function outputs." << endl;

    int x = 5, y=10;
    int *yy = &y;

    function1(x, y); // Pass by value
    cout << "x = " << x << endl;
    cout << "y = " << y << endl;

    x = 5; y = 10;
    function2(x, y); // Pass by reference
    cout << "x = " << x << endl;
    cout << "y = " << y << endl;

    x = 5; y = 10;
    function3(x, yy); // Pass by reference
    cout << "x = " << x << endl;
    cout << "y = " << y << endl;
}
void function1(int x, int y)
{
    cout << "\n\nBy val" << endl;
    cout << "x = " << x << endl;
    cout << "y = " << y << endl;  // Here y is a variable passed by value
    x = -1;
    y = -1;
}

void function2(int x, int &y)
{
    cout << "\n\nBy ref" << endl;
    cout << "x = " << x << endl;
    cout << "y = " << y << endl;  // Here y is a variable passed by reference
    x = -1;
    y = -1;
}

void function3(int x, int *y)
{
    cout << "\n\nBy ref" << endl;
    cout << "x = " << x << endl;
    cout << "y = " << y << endl;  // here y is an address
    x = -1;
    //y = -1;  //won't work!
    *y = -1;  // This will
}

void Problem_15(void)
{
    cout << "\n--------------------------------------------------------" << endl;
    cout << "15) Parameter lists can include both pass - by - value and pass - by - reference parameters." << endl;
    cout << "yup!, see last example" << endl;
}

int Problem_16(void)
{
    cout << "\n----------------------------------------" << endl;
    cout << "16) All functions must have a return statement." << endl;
    cout << "Non-void functions should return a value of corresponding to their type" << endl;
    // must have a return statement.
    return(0);
}

void Problem_17(void)
{
    cout << "\n----------------------------------------" << endl;
    cout << "17) Which of the following is a complete function definition ?" << endl;
    cout << "a) int funct(int x) { return (x = x + 1); }" << endl;
    cout << "b) int funct(double); " << endl;
    cout << "c) void funct(int) { cout << 'Hello'; } " << endl;
    cout << "d) int funct(int) {}; " << endl;
    cout << "e) None of the above" << endl;
    cout << "Solution............" << endl;
    cout << "a) is complete" << endl;
    cout << "b) is missing the parameter name. It is probably a forward declaration!" << endl;
    cout << "c) is also missing the parameter name";
    cout << "d) is also missing the parameter name, it can be used a forward declaration" << endl;
}

void Problem_18(void)
{
    cout << "\n--------------------------------------------------------" << endl;
    cout << "18) Which function is called from the main?" << endl;

    //
    //int add(int, int);
    //double add(int, int&);
    //int main() {
    //    int a = 1, b = 2;
    //    cout << add(a, b);
    //}

    cout << "a) int add(int, int);     b) double add(int, int&);" << endl;
    cout << "c) Compilation error    d) Run time error" << endl;

    cout << "this will be a compiler error since the parameters int and &int are both viewed as int by
    C++ compiler!" << endl;
}

double funct(int, const int);
//void funct(int, int);
double funct(double, int);
double funct(int, double);

void Problem_19(void)
{
    cout << "\n---------------------------------------------------------------------" << endl;

    cout << "19) Which of the function prototypes of funct are overloaded" << endl;
    cout << " double funct(int, const int); ==> 1" << endl;
    cout << " void funct(int, int);       ==> 2" << endl;
    cout << " double funct(double, int); ==> 3" << endl;
    cout << " double funct(int, double); ==> 4" << endl;

    cout << "a) 1 and 2    b) 1, 2 and 3    c) 1, 2 and 4    d) 1, 3 and 4    e) 1, 2, 3 and 4" << endl;

    cout << func(5, 10) << endl;
    cout << func(5.5, 10) << endl;
    cout << func(5, 5.5) << endl;

    cout << "d) 1, 3 and 4" << endl;  // f) 2, 3, 4 is also overloaded but that is not one of the choices.
}

double funct(int a, const int b)
{
    return(0.0);
}
double funct(double a, int b)
{
    return(0.0);
}
double funct(int a, double b)
{
    return(0.0);
}
// include <iostream>
// using namespace std;

// forward declaration
int funct1(int, int &);
void print(int);

void Problem_20(void)
{
    cout << "\n--------------------------------------------------------" << endl;
    cout << "20) What is the output of the following C++ program ?" << endl;
    cout << "a) 1212    b) 1213     c) 1312     d) 1313     e) 133" << endl;
    cout << "\nSolution.............." << endl;
    //int main(void) {
        int x = 3, y = 7;
        print(funct1(y, x));
        cout << x;   // the modified value of x gets printed!
    //}
}

int funct1(int a, int & b) {
    int x = 5;
    b = x + a;
    return b++; // returns b, then increments it which will change x in the calling program!
}
void print(int c) {
    cout << c;
}