Lesson 2: Structural Programming with C Language
The first C program

/* hello.c */
#include <stdio.h>
int main() {
    printf("Hello!\n");
    return 0;
}

This program exports the following text to screen:
Hello!
Investigate the example

The example program includes:
- A function definition: main()
- A commentary line
- A compiler directive (declaration of using library)
- An instruction to export text to screen (standard output)
- A value returned

This program:
- Asks computer to export a text to screen
- Returns value 0 to the parent program (the program that calls the example)
The main() function

- Used to start the execution of a C program, and is required.
- Declared in one of the following two ways:
  - `int main() { ... }`
  - `int main(int argc, char* argv[]) { ... }
- In C++, main() can be declared with “void” return type.
- When starting, some parameters are passed to the program; and when terminating, the program returns a code. Ex:
  - `C:\>copy /B a.dat b.dat`
Example 2: Calculate circle area

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main() {
    float R;
    printf("Radius = ");
    scanf("%f", &R);
    printf("The circle area is: %.3f\n", 3.14 * R*R);
    return 0;
}
```

- Result:
  - Radius = 1
  - The circle area is: 3.140
Display some text (export to screen)

- Syntax:
  - `printf("Format string", <values>);`

- Typing symbols:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Type</th>
<th>Symbol</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>%f, %e, %g, %lf</td>
<td>double, float</td>
<td>%x</td>
<td>int (hex)</td>
</tr>
<tr>
<td>%d</td>
<td>int</td>
<td>%o</td>
<td>int (oct)</td>
</tr>
<tr>
<td>%c</td>
<td>char</td>
<td>%u</td>
<td>unsigned int</td>
</tr>
<tr>
<td>%s</td>
<td>string of characters</td>
<td>%p</td>
<td>pointer</td>
</tr>
</tbody>
</table>

- Formatting:
  - `[%flags] [width] [.precision]type`
  - Ex: `%+15.5f`
Read user input (typed from keyboard)

- **Syntax:**
  - `scanf("Format string", <variable addresses>);`

- **Examples:**
  - `int age;`  
    `scanf("%d", &age);`
  
  - `float weight;`  
    `scanf("%f", &weight);`
  
  - `char name[20];`  
    `scanf("%s", name);`
Variables, types and values
Variables and types

- Variables can hold values, which can be changed during runtime
- Variables need to be declared before using, with a type
- Global scope or limited within a function
- In standard C, internal variables need to be declared at the beginning of functions, before any instructions
- Variable declaration: `<type> <list of variables>;
  ```
  int a, b, c;
  unsigned char u;
  ```
- Basic types:
  - char, int, short, long
  - float, double
Assignment

- To change the value of a variable with a new one

Syntax:
  - `<variable> = <constant, variable> or <expression>`

Ex:
  - `count = 100;`
  - `value = cos(x);`
  - `i = i + 2;`

Values of variables can be initialized at declaration (if not, the value is undefined):
  - `int count = 100;`
  - `char key = 'K';`
Constants

- Like variables but their values can not be changed at runtime
- Declared by adding `const` keyword
- Constants in C occupy memory like variables
- Ex:
  - `const double PI = 3.14159;`
  - `const char* name = "Nguyen Viet Tung";`
  - `PI = 3.14; /* error */`
- Another way to make a constant: using macro → not occupy memory (but un-typed)
  - `#define PI 3.14159`
### Basic data (aka primitive) types

<table>
<thead>
<tr>
<th>Type</th>
<th>Size (length in bytes)</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1</td>
<td>Character, integer</td>
</tr>
<tr>
<td>int</td>
<td>(depending: 2, 4, 8)</td>
<td>Integer</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
<td>Integer</td>
</tr>
<tr>
<td>long</td>
<td>4</td>
<td>Integer</td>
</tr>
<tr>
<td>long long</td>
<td>8</td>
<td>Integer</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
<td>Real (floating point)</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
<td>Real (floating point)</td>
</tr>
<tr>
<td>void</td>
<td>0</td>
<td>Unspecified type</td>
</tr>
</tbody>
</table>

- Characters in C are 8-bit integers
- Use `sizeof()` operator to calculate the sizes of variables or data types in bytes:
  - `sizeof(x)`
  - `sizeof(int)`
Type casting

- Conversion of value of an expression from one type to another

- Implicit casting:
  - float a = 30;
  - int b = 'a';

- Explicit casting:
  - int a = (int)5.6; /* take integer part */
  - float f = (float)1/3;

- Not any type can be casted into any other
  - char* s = 2.3; /* not compiled */
  - int x = "7"; /* compiled but incorrect */
Variable size and value range

- Signed and unsigned:
  - signed char (8 bits)  -128 ~ +127
  - signed short (16 bits)  -32768 ~ +32767
  - signed int (32 bits)  -2147483648 ~ +2147483648
  - signed long (32 bits)  -2147483648 ~ +2147483648
  - unsigned char (8 bits)  0 ~ +255
  - unsigned short (16 bits)  0 ~ +65535
  - unsigned int (32 bits)  0 ~ +4294967295
  - unsigned long (32 bits)  0 ~ +4294967295

- Attn:
  - Implicitly signed
  - Size of int variables are dependent on platform
Enumeration (enum)

- Used to define the possible values of a data type
- Syntax: `enum <type name> { <values> };`
- Ex:
  ```
  enum WildAnimal {Tiger, Leopard, Bear, Deer };
  enum WeekDay { Mon = 2, Tue, Wed, Thu, Fri, Sat, Sun = 1 };
  ```
- Usage:
  ```
  enum WildAnimal wa = Tiger;
  wa = Leopard;
  enum WeekDay n = Thu;
  ```
Structures (struct)

- Used to define data types composed of sub-variables (members, fields)
- Syntax: struct <data type> { <member variables> };
- Ex:
  ```c
  struct Student {
    char name[20];
    int birth_year;
    int school_year;
  };
  ```
- Usage:
  ```c
  struct Student st = {"Le Duc Tho", 1984, 56};
  st.birth_day = 1985;
  st.school_year = 54;
  ```
Defining new types from old ones (typedef)

- Used define new names for old types that are shorter or with different significations
- Syntax: typedef <original type> <new name>;
- Ex:
  - typedef double Height;
  - typedef unsigned char byte;
  - typedef enum WildAnimal WA;
  - typedef struct { … } Student;

- Usage
  - Height d = 165.5;
  - byte b = 30;
  - WA wa = Tiger;
Arrays

- Used to store multiple elements of a same type in memory at consecutive locations
- Are static pointers by nature
- Syntax: `<type name> <variable name> [ <no of elem.> ];`
- Ex:
  ```
  ```
  
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>23</td>
<td>50</td>
<td>18</td>
<td>40</td>
<td>25</td>
<td>33</td>
</tr>
</tbody>
</table>

- Access elements: index starting from 0
  - `age[3] = 20;`

- Two-dimensional arrays (and multi-dimensional):
  ```
  float matrix[10][20];
  matrix[5][15] = 1.23;
  ```
Some other types

- **Boolean:**
  - Not existing in C
  - Use `int`/`char` or `enum` for that goals
    ```c
    typedef int bool;
    const int false = 0, true = 1;
    typedef enum {false, true} bool;
    ```

- **String:**
  ```c
  char* ho_ten = "Nguyen Viet Tung";
  char dia_chi[50] = "So 1, Dai Co Viet, Ha Noi";
  ```

- **Union:** multiple member variables at the same memory address
  ```c
  union color {
    struct {unsigned char R,G,B,A;} s_color;
    unsigned int i_color;
  };
  ```
Composed types

- Used to combine multiple sub-variables of same or different types

```c
typedef struct {
    char name[20];
    unsigned int age;
    enum {Male, Female} sex;
    struct {
        char city[20];
        char street[20];
        int number;
    } address;
} Student;
```
Problems

Write programs to:

1. Use `sizeof()` operator to print the sizes of the basic and some composed data types to screen

2. Calculate trigonometric values of angle $\alpha$ entered by user

3. Enter data for Student structure (including name, year of birth, school year) and re-print to screen

4. Declare two variables $x$ (char) and $y$ (unsigned char), assign -1 to $x$, then cast its value and assign to $y$. Print out the value of $y$ and explain the result.

5. Declare a data type that describes a car with following information: model, weight, color, 4 wheels where each one has: type, radius, weight

6. From above problems, but input and then print out the information