COMP1917: Computing 1

2. Numbers In, Numbers Out

Reading: Moffat, Chapter 2.

The Art of Programming

- Think about the problem
- Write down a proposed solution
- Break each step into smaller steps
- Convert the basic steps into instructions in the program
- Use an editor to create a file containing the program
- Use the compiler to check the syntax of the program
- Test the program on a range of data

Work methodically – don't just hack !

Like an architect building a house, or a hungry person eating a pizza, you should plan in advance and do the job one piece and one bite at a time, not try to shove the whole thing into your mouth at once.

Remember:

> "A day of debugging can save an hour of planning"

Programming Task

Write a program to:

- read an integer from standard input (keyboard)
- compute the cube of that integer
- print that result to standard output (screen)

Structure of the Program

```
#include <stdio.h>

int main( void )
{
   // declare variable(s)

   // read value(s) from standard input (keyboard)

   // compute the required result

   // print result to standard output (screen)

   return 0;
}
```

Variables and Types

- Variables are used to store data
- In C each variable must have a type
- C has the following generic data types:

`char`	character	('A', 'e', '#', . . .)
`int`	integer	(2, 17, -5, . . .)
`float`	floating point number	(3.14159, . . .)

- there are other types, which are variations on these three.

Variable Declarations

- Each variable must have a declaration, which tells the compiler to reserve storage space
- Variable declarations are like the list of ingredients at the top of a cooking recipe
- A variable declaration must specify a data type and a name

```
float x;
char ch;
int num;
```

- Declarations can be placed after the opening bracket `{` in a function

Function and Variable Names – Identifiers

- Identifier names must be made up of letters and digits.
- The first character must be a letter.
- The underscore character '_' counts as a letter.
- uppercase and lowercase are different
- **Restrictions**: Keywords like:

  `if, while, do, int, char, float ...` *`etc`*

  cannot be used as identifiers

Input using `scanf()`

```
#include <stdio.h>

   ...

   scanf( "%d", &num );
```

- reads input according to a format and stores value(s) in arguments
- returns the number of input items successfully read, or `EOF` on failure or end of file
- at least one argument but possibly more
- each argument (unless an array) must be preceded by ampersand character `'&'`

scanf()

```
#include <stdio.h>
int scanf(const char *format, ...);
```

Format can contain:

- blanks, tabs (whitespace) – ignored
- ordinary characters – must match characters of input stream
- conversion specifications
  - %d – decimal integer
  - %c – character
  - %f – float
  - %lf – double (“long float”)

Assigning Values to Variables

- Variables can be assigned values using the = operator

```
n = 23;
x = 3.1415;
```

- Variables can also be initialised when they are declared

```
char c = 'A';
int x = 75;
```

- Variables can be modified during a program

```
cube = num * num * num;
x = x + 1;
```

  The expression to the right of = is evaluated and the result stored in the variable to the left of the =

- variables must be assigned before they are used; otherwise they will have "garbage" values.

Arithmetic Operators

The following are binary operators:

Name	Symbol	Example	Conditions
Add	$+$	$a+b$	none
Subtract	$-$	$a-b$	none
Multiply	$*$	$a*b$	none
Divide	/	a / b	ignore remainder for integer division
Modulus	%	$a \% b$	remainder of a / b

Abbreviated Assignment Operators

C allows certain “abbreviated” assignment operators when the same variable appears on the left and right side. For example,

```
x += 10;
```

is an abbreviation for

```
x = x + 10;
```

We can also use

```
-=    *=    /=    %=
```

Printing Variable Values with `printf()`

```
printf("The cube of %d is %d\n", num, cube );
```

- formatted output written to standard output
- very useful and flexible function
- can print multiple values in a single statement
- `printf()`, used judiciously, can be very helpful for debugging

Complete Program

```
#include <stdio.h>

int main( void )
{
   int num, cube;
   printf("Enter a number: " ); // prompt user for input
   scanf( "%d", &num );
   cube = num * num * num;
   printf("The cube of %d is %d\n", num, cube );
   return 0;
}
```

Slightly More Elaborate Programming Task

Einstein's theory of Relativity predicts that the mass of an object will increase as its velocity increases, according to this equation:

$$m = \frac{m_0}{\sqrt{1 - (\frac{v}{c})^2}}$$

where:

m	is the observed mass
m_0	is the rest mass
v	is the velocity
c	is the speed of light

Write a program which reads the rest mass and velocity from standard input, computes the observed mass and prints it to standard output.

(You do not need to understand the physics in order to complete the task!)

Variable Names

It is helpful to use descriptive names for your variables, which make the code eaiser to read and understand.

```
float mass, rest_mass;
float velocity, ratio;
```

These are called “variables” because their values can vary as the program executes.

We might like to treat the speed of light differently, because its value is constant and cannot change during the execution of the program.

There are two ways to handle constants in C.

Constant Declarations

We can declare a “constant variable”:

```
const float SPEED_OF_LIGHT = 299792458.0;
```

. . . or, using scientific notation . . .

```
const float SPEED_OF_LIGHT = 3e+8;
```

- variables declared with the `const` qualifier cannot be altered.
- attempting to alter a `const` variable will generate a compilation error.

Symbolic Constants

Alternatively, we can define a symbolic constant at the top of the file:

```
#define SPEED_OF_LIGHT 299792458.0
```

- we use symbolic constants to avoid burying "magic numbers" or values in the code.
- symbolic constants make the code easier to understand and maintain

```
#define name    replacement text
```

- the *compiler's pre-processor* will *parse* your code replacing all occurrences of *name* with replacement text.
- it will not make the replacement if *name* is inside quotes or part of another name.

Identifiers – Conventions

- many library functions begin their names with an underscore `'_'` . For this reason, you should avoid starting your own function or variable names with an underscore.
- some programmers like to capitalize the first letter of each word

  ```
  SetPrime();
  ```

  others like to use all lowercase, with an underscore between words

  ```
  int frequency_count;
  ```

- by convention, constants are defined in full uppercase.
- single letters from `'a'` to `'n'` often used as counters and indices.
- longer names are encouraged for external variables and for clarity.

`scanf()` and `printf()` with `floats`

```
printf("Enter rest mass: ");
scanf( "%f", &rest_mass );

printf("Enter velocity in m/s: ");
scanf( "%f", &velocity );

 ...

printf( "Observed mass = %1.6f\n", mass );
```

printf()

```
#include <stdio.h>
int printf( const char *format, ... );
```

`format` may contain ordinary text as well as conversion characters, which print the value of the next argument:

%d	decimal integer
%5d	decimal integer at least 5 chars wide
%f	floating point number
%5f	floating point number at least 5 chars wide
%.3f	floating point number 3 decimal places
%5.3f	floating point number at least 5 chars 3 decimal places

Mathematical Equations

If necessary, break a large equation into smaller pieces or split it across multiple lines.

$$m = \frac{m_0}{\sqrt{1 - (\frac{v}{c})^2}}$$

```
ratio = velocity / SPEED_OF_LIGHT;
mass = rest_mass / sqrt( 1.0 - ratio * ratio );
```

Math Library Functions

```
#include <math.h>

   sqrt( ... )
```

If your program uses mathematical functions like `sqrt()`, `sin()`, `cos()`, `log()`, `exp()`, etc. then you need to include `math.h` and also compile with the `-lm` option (which stands for "library math")

```
$ gcc -Wall -o einstein einstein.c -lm
```

Complete Program

```
#include <stdio.h>
#include <math.h>
// remember to compile with -lm

// speed of light in m/s
#define SPEED_OF_LIGHT 299792458.0

int main( void )
{
   float mass, rest_mass;
   float velocity;
   float ratio;
```

```
    printf("Enter rest mass: ");
    scanf( "%f", &rest_mass );

    printf("Enter velocity in m/s: ");
    scanf( "%f", &velocity );

    // compute observed mass using Einstein's equation
    ratio = velocity / SPEED_OF_LIGHT;
    mass = rest_mass / sqrt( 1.0 - ratio * ratio );

    printf( "Observed mass = %1.6f\n", mass );

    return 0;
}
```

Question

What if you type in a `velocity` that is greater than the `SPEED_OF_LIGHT`?

What will happen?

What should happen?

How can you make it happen?