Introduction to *C* programming

- **C is *not* an object orient programming language !**
  - **Students** doing the ***Computer Science*** in **Emory** learned the ***Java* programming language**
  - **Fact:**
    - **Java** is an **object-oriented** programming language
  - **Trade mark** of **Object-oriented** programming language:
    - **Object-oriented** programming languages provide a ***programming construct*** to **associate data (variables)** and **program code (methods)**.
    - The **program code *associated*** with the **data** has ***special* access permission** to the **data**

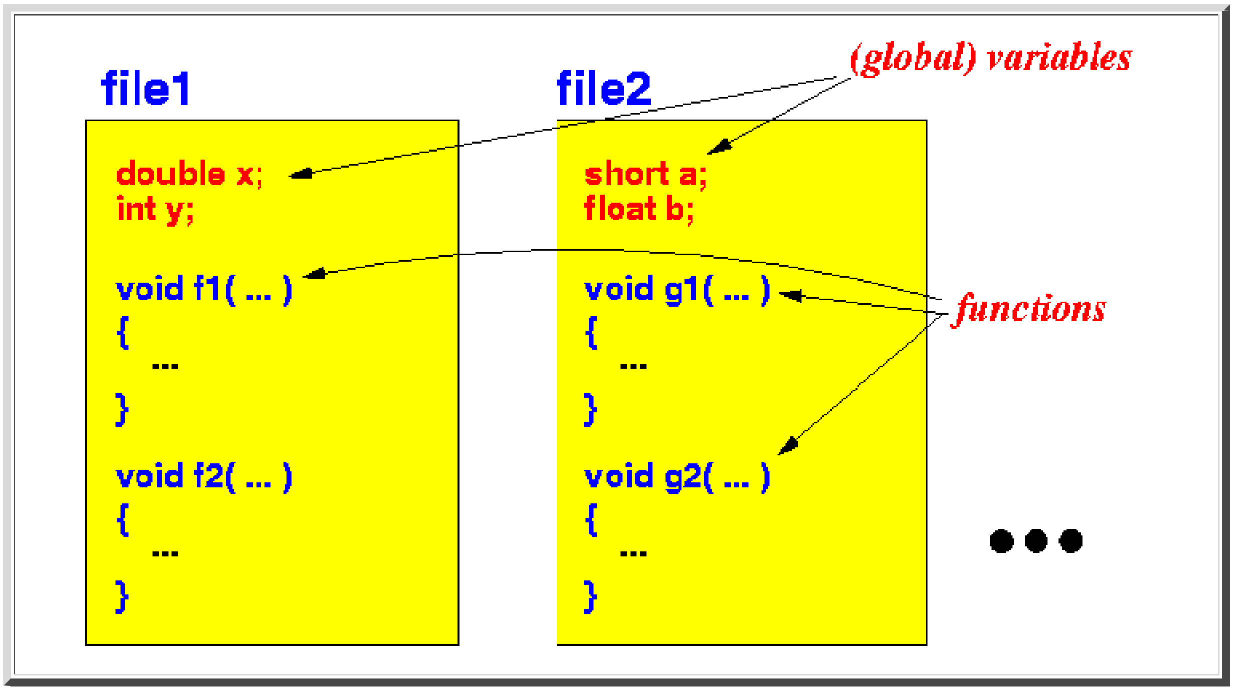
      Example:

      - In **Java**, ***only*** the **methods (code)** in the ***same* class** as the **variables (data)** can ***access*** the ***private* variables** defined inside that **class** !!!
  - **Fact:**
    - **C does *not*** provide **any mechanism** to **associate data (variables)** and **code (methods/functions)**

- **Structure of a C program**
  - Structure of a **C** program:
    - A **C** program consists of a **collection** of
      - **Data structures/types definitions**
      - **(Global) variables**
      - **Functions (with local variables and statements)**

      stored in **one or more *files***.

  Schematically:



(The figure above only showed: **variables** and **methods** - you can see that these constructs are **similar** to those in **Java**.

I **did not** show you any **data structure/type definitions** because these constructs are quite **different** from **Java**.

`struc` and `typedef` will be discussed **later**....)

- The **start** of the **execution** of a **C program**:
  - The **execution** of a **C** program begins with the `main()` ***function***

- **Terminology: Function, method, procedure, subroutine....**
  - **Historical note:**
    - What we now call a **method**, was ***traditionally*** called:
      - a **subroutine** (or subprogram), or
      - a **procedure**, or
      - a **function**
  - Personal practice:
    - In this course, I was use **function** instead of **method** because that's still the common practice in **C**.

- **Everyone's *first* C program: Hello World**
  - The **Hello World** program in **C**:

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

```
int main( int argc, char* argv[] )
{
   printf( "Hello World !\n" );
}
```

- **Example Program:** (Demo above code) *Example*
  - Prog file: click here

How to run the program:

> - **Right click** on link and **save** in a scratch directory
> - To compile: `gcc hello.c` (output file is named `a.out`)
> - To run: `./a.out`

Or:

> - **Right click** on link and **save** in a scratch directory
> - To compile: `gcc -o hello hello.c` (option **-o** renames output file to `hello`)
> - To run: `./hello`

- **Explantion:**
  - `#include <stdio.h>`
    - The `#` symbol starts a **command** for the **C pre-processor**
    - The `#include` **command** instructs the **C pre-processor** to **read in** the file `stdio.h` from the **System include directory** (this is traditionally the directory: `/usr/include`)
    - The file `stdio.h (/usr/include/stdio.h)` is **C**'s **standard IO** include file

      This file contains **constant and variable definitions** to allow **C programs** to perform commonly used **input/output operations**.

      Take a look at the file `/usr/include/stdio.h`....

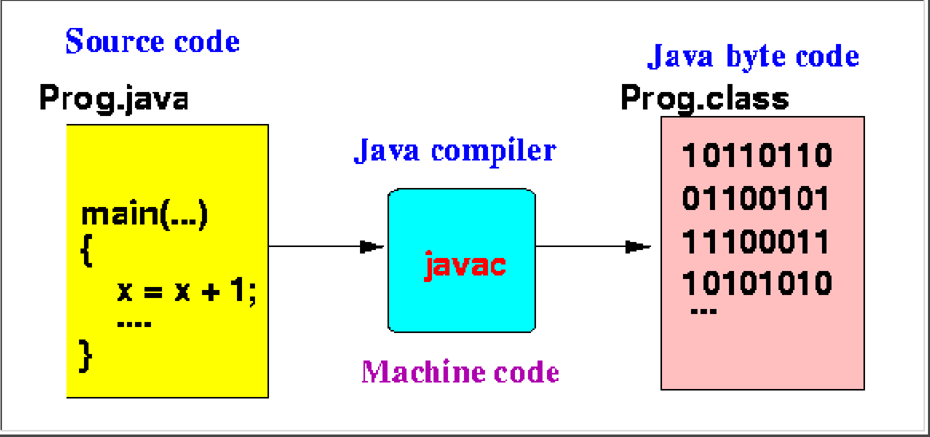
  - `int main( int argc, char* argv[] )`
    - This line is the **header** of the **definition** of the `main()` ***function***
    - The function `main()` will return an **integer error code** (may be used by a **shell script** to check for the outcome)
    - The **parameters** of `main()` are:

```
int   argc     =  number of parameter strings
char* argv[]   =  array of String
                     argv[0] = first argument
                     argv[1] = second argument
                     ...
                     argv[argc-1] = last argument
```

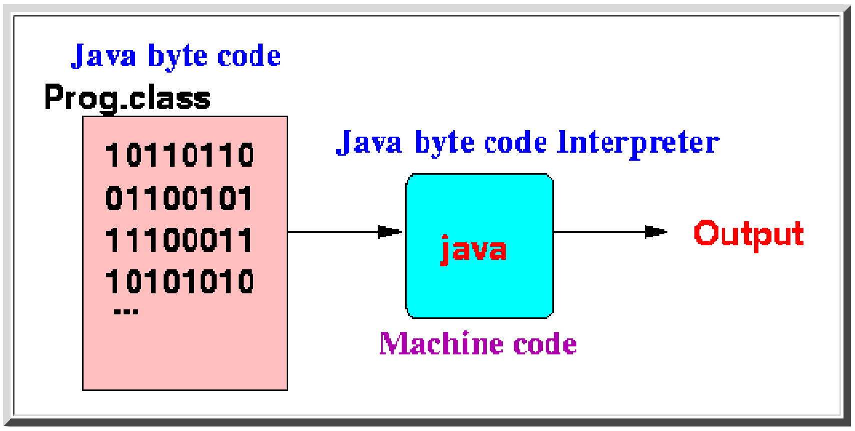
- `printf("Hello World !\n");`
  - This is the **statement** inside the **body** of the `main()` ***function***
  - `printf` is the **C** library function to print outputs to the **terminal**

- **Terminology in C**
  - Some terminilogy used in **C**:
    - `stdio.h` = header file containing definitions for the **standard Input/Output** operations
    - `stdin` = the ***name*** of the **standard input** device (which is the **keyboard**)
    - `stdout` = the ***name*** of the **standard output** device (which is the **screen** or **terminal**)

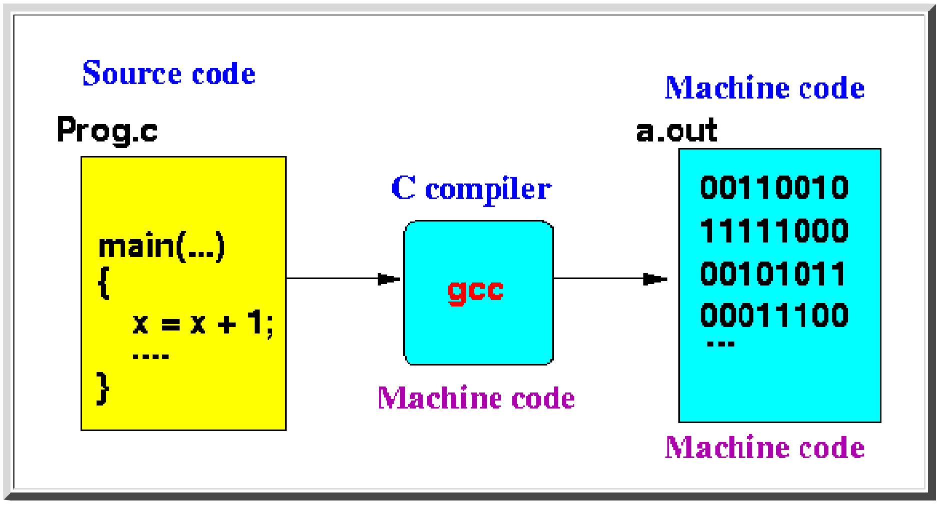
- **The C compilation process**
  - **Fact:**
    - **Unlike *Java***, the **C** compiler translates the **C program *source code*** into **machine executable instructions (code)**
  - This is the process used to **compile** and then **run** a ***Java* program**:
    - **Compile** the **Java source code *using*** a **Java compiler (`javac`** (output is a file containing **Java byte code** - instruction code for a **virtual** machine)



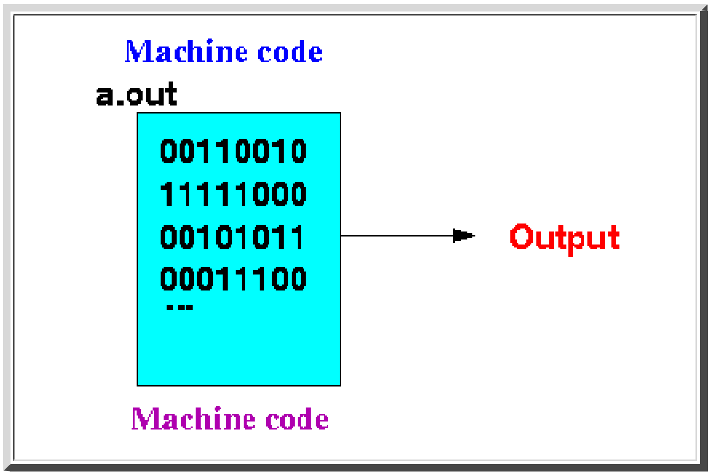
    - **Execute** the **Java source code *using*** a **Java byte code *interpreter* (`java`)** (execution may generate output to the terminal or file(s)).



- This is the process used to **compile** and then **run** a ***C*** **program**:
  - **Compile** the ***C*** **source code** ***using*** a **C compiler** (gcc (output is a file containing **machine executable code** - instruction code for a ***actual*** computer !!!)



  - **Execute** the **machine code** ***directly*** by the **computer**: (execution may generate output to the terminal or file(s)).



- **Facts:**

- **Interpretation** (executing code using an **interpreter**) is **very *inefficient***.
- Due to the fact that **C** program source is translated **machine code**, **C programs** run multiple times (at least 10) faster than **Java programs**.