Macros

- **Macros**
  - **Macro:**

    > - **Macro** = a **rule or pattern** that specifies how a certain **input sequence** (= a sequence of characters) should be **mapped** to a **replacement sequence**

  - **Syntax** to define a **macro**:

    ```
    #define <identifier>                        <replacement token list>
    #define <identifier>(<parameter list>)      <replacement token list>
    ```

    Meaning:

    > - Every occurence of the `<identifier>` string in the **source code** will be **replaced** by the `<replacement token list>`.

  - **Use** of **macros** in **C**:

    > - Defining **constant**
    > - Defining **simple functions**

- **Defining *symbolic constants* in C**
  - **C** uses the **C pre-processor's** `#define` **macro construct** to define **symbolic constants**

    Example:

    ```
    #define PI   3.14159265358979323846264338327950288419716939937510
    #define MAX  99999

    int main( int argc, char* argv[] )
    {
       int x;
       double y;

       x = MAX;
       y = PI;
    }
    ```

After processing with `gcc -E constant.c`:

```
int main( int argc, char* argv[] )
{
   int x;
   double y;

   x = 99999;
   y = 3.14159265358979323846264338327950288419 7;
}
```

- **Example Program:** (Demo above code) 

  - Prog file: click here

How to run the program:

- **Right click** on link and **save** in a scratch directory
- To compile: `gcc -E constant.c`
- Look at the output on the terminal....

- **Defining *simple functions* in C**

  - You can use the `#define` **macro construct** to define **simple (one line) functions**

Example:

```
#define square(x)    (x*x)

int main( int argc, char* argv[] )
{
   double a, b;

   b = square(a);
}
```

After processing with `gcc -E macro1.c`:

```
int main( int argc, char* argv[] )
{
   double a, b;

   b = (a*a);            ( square(a) ---> (a*a) )
}
```

- **Example Program:** (Demo above code) 

  - Prog file: click here

How to run the program:

- **Right click** on link and **save** in a scratch directory
- To compile: `gcc -E macro1.c`
- Look at the output on the terminal....

- **Caveat:**

- Make **very sure** that you use **brackets** to prevent **errors** caused by **operator precedence**

Example:

```
#define sum(x,y)    x+y         /* Sum: x+y */

int main( int argc, char* argv[] )
{
   double a, b, c;

   c = sum(a, b);      /* Compute a+b          */

   c = 2*sum(a, b);    /* Compute 2*(a+b) ??? */
}
```

After processing with `gcc -E macro2.c`:

```
int main( int argc, char* argv[] )
{
   double a, b, c;

   c = a+b;

   c = 2*a+b;           /* Wrong !!! Not equal to: 2*(a+b) !!! */
}
```

- **Example Program:** (Demo above code) 

  - Prog file: click here

How to run the program:

- **Right click** on link and **save** in a scratch directory
- To compile: `gcc -E macro2.c`
- Look at the output on the terminal....

- **Undefining a macro**
  - **Undefine** a macro:

```
#undef   macro-name
```

Example:

```
#undef  PI
#undef  MAX
#undef  square
```
