**Inline Functions**

- Efficient Function Calls
  - The language reserved word qualifier “inline” can be used to precede ‘simple’ function definitions.

  ```
  inline bool leap(int year)
  {return((((year%4==0)&&(year%100!=0))||(year%400==0));}
  ```

  - Instructs compiler to attempt to generate a copy of the function’s code inline (in place) instead of translating the function normally and generating calls to the function code.
  - Compilers may ignore the inline qualifier and translate the function normally.
  - Can avoid the execution overhead of a function call for small functions.
  - Helps achieve design goal of implementing a system as a set of functions.
  - Execution speed saving is offset by the execution image being larger due to copies of the function.
  - Inline should only be applied to small one-line functions.
  - When used in separate compilation may force extra translation.
  - Inline functions should be used over macro expansions due to the type-checking performed on inline functions.

**Assert**

- Debug Error Checking
  - The assert macro pseudo-function defined in `<assert.h>`, and `<cassert>`, (new style header), is used to check a condition, (pre-condition, post-condition, etc.).

  ```
  assert ( (index>=0) && (index < MAXDIM) );
  ```

  - If the condition is false, assert prints an error message containing the line number, the condition tested, and the file name containing the assert, calls the abort function in `<stdlib.h>` and `<cstdlib>` to halt program execution.
  - If the condition is true execution continues normally.

- Release builds and assertions
  - Assert functions need not be removed after testing is complete.
  - Defining the preprocessor symbolic constant NDEBUG will force the preprocessor to ignore all of the assertions.

  ```
  #define NDEBUG
  ```

- Considerations
  - Assertions do not allow for programs to recover from errors.
  - It is good programming practice to precede all array accesses with assertions for bounds checking.