C++ Programming
Lecture 4: Variables and Functions - Part II

Nguyen, Thien Binh
Mechatronics and Sensor Technology
Vietnamese-German University

20 March 2019

VGU
Vietnamese - German University

Outline

3. The Preprocessor
   - `#include` Directive
   - `#define` Directive
   - Conditional Compilation with `#ifdef`, `#ifndef`, `#elseif`, `#endif`
   - Header Files
4. Applications to Linear Algebra: Coding 3

VGU
Vietnamese - German University

Outline

The Preprocessor

- Prior to compilation, the code goes through a phase known as *translation* in which a *preprocessor* takes place.
- The preprocessor ignores all code contents but looks for special directives starting with `#` and makes appropriate changes/substitutions
- The following directives are noteworthy: `#include`, `#define`, and the conditional compilation directivesñ `#ifdef`, `#ifndef`, `#elseif`, `#endif`

VGU
Vietnamese - German University

#include Directive

```
1 #include <iostream>
2 #include <cmath>
3 #include <cassert>
4 #include "user-header-file.h"
```

- When the preprocessor scans and finds the #include, it will replace the directive by all the preprocessed contents of associate header file.
- A < > bracket is used for standard ANSI C++ libraries, e.g., iostream, cmath, or cassert, whereas a quotation " " is used for user-defined header files.
- The #include directive is mainly used to substitute header files .h into source files .cpp

VGU
Vietnamese - German University

#include Directive

```
1   double  TriArea(double height, double base);
2   void Print(double result);
```

Listing 1: ExampleDeclare.h

```
1  #include <iostream>
2  #include "ExampleDeclare.h"
3  using namespace std;
4
5  double  TriArea(double height, double base)
6  {
7    double  area;
8    area = 0.5*height*base;
9    return area;
10 }
11
12 void Print(double result)
13 {
14   cout << "Result = " << result << endl;
15 }
```

Listing 2: ExampleDefine.cpp

VGU Vietnamese - German University

- ExampleDeclare.h and ExampleDefine.cpp are equivalent to

```
1  // all preprocessed contents of
2  // /usr/include/g++/iostream
3  // the contents of ExampleDeclare.h
4  double  TriArea(double height, double base);
5  void Print(double result);
6  using namespace std;
7
8  double  TriArea(double height, double base)
9  {
10   double  area;
11   area = 0.5*height*base;
12   return area;
13 }
14
15 void Print(double result)
16 {
17   cout << "Result = " << result << endl;
18 }
```

#define Directive

```
1   #define IDENTIFIER tokens
```

- When a processor scans and finds `#define`, it will textually substitute all occurrences of `IDENTIFIER` with `tokens`
- The `#define` directive is mostly used for defining and giving meaningful names for global constants

```
1   #define PI          3.14159265358979323846264338
2   #define LIGHTSPEED  2.997925e8
3   #define ZERO        0.00000000000000000000000000
```

VGU
Vietnamese - German University

#define Directive

- The #define directive can also be used to define simple functions, e.g.,

```
1    #define SQUARE(X) ((x) * (X))
```

then

```
1    y = SQUARE(4.0);
```

is equivalent to

```
1    y = ( (4.0) * (4.0) );
```

- It is always considered a better practice to use const to define constants instead of #define (see Lecture 2)

VGU
Vietnamese - German University

Conditional Compilation

```
1  #define CONDITION_1
2
3  #ifdef CONDITION_1
4    // code segment 1
5  #endif
6
7  #ifndef CONDITION_2
8    // code segment 2
9  #endif
```

- `#ifdef`, `#ifndef`, `#elseif`, `#endif` can be used to determine which part of the code is going to be compiled and which is not.
- `code segment 1` will be compiled if `CONDITION_1` is defined. On the contrary, `code segment 2` will be compiled if `CONDITION_2` is *not* defined.

VGU
Vietnamese - German University

Conditional Compilation

- Example: What is printed out to the screen?

```
#include <iostream>
using namespace std;
#define COMPILE
void printsomething()
{
  #ifdef COMPILE
    cout << "code segment 1" << endl;
  #endif

  #ifndef COMPILE
    cout << "code segment 2" << endl;
  #endif
}

int main()
{
  printsomething();
  return 0;
}
```

Conditional Compilation

- Example: What is printed out to the screen? Why is that?

```
#include <iostream>
using namespace std;
void printsomething()
{
  #ifdef COMPILE
    cout << "code segment 1" << endl;
  #endif

  #ifndef COMPILE
    cout << "code segment 2" << endl;
  #endif
}

#define COMPILE
int main()
{
  printsomething();
  return 0;
}
```

Conditional Compilation

- Example: What is printed out to the screen? Why is that?

```
#include <iostream>
using namespace std;
void printsomething()
{
  #ifdef COMPILE
    cout << "code segment 1" << endl;
  #endif
  #ifndef COMPILE
    cout << "code segment 2" << endl;
  #endif
}
#define COMPILE
int main()
{
  printsomething();
  return 0;
}
```

$\Rightarrow$ *A preprocessor* ignores all code contents or sequences but looks only for directives from top to bottom of the code.

Conditional Compilation

- The `#ifdef`, `#ifndef`, `#endif` together with `#define` directive are of particularly useful in creating header guards for header files which prevents multiple definition.

Header Files

- A C++ files are basically classified into 2 types:
  - source files with the extension `.cpp` which contain all variables, functions, and classes definitions,
  - header files with the extension `.h` in which functions and classes are declared. Short `inline` functions and global constants may also be defined in header files.
- Header files are included into source files with the `#include` directive.
- Using header files enhance code readability and abstraction since users could justify the use of, e.g., a class, by inspecting its member data and methods declared in the header file
- Header files also serve as the interface for packaged libraries. It is common that a shared C++ library has its source files precompiled for the reason of security or copyrights, and users just need to include the library's header file in order to use it.

VGU
Vietnamese - German University

Header Files I

- Although a header file can be included in as many files as wanted, this could
  - increase the overhead cost as a preprocessor has to substitute all the contents of the header file at the inclusion location
  - return errors if there are variables or non-inline functions defined more than once.
- Example: What is wrong with the following code?

```
1  #include <iostream>
2  using namespace std;
3
4  // global variable;
5  double area;
6
7  // functions
8  double recArea(const double& side1, const double& side2);
9  void printArea()
10 {
```

Header Files II

```
11    cout << "The area is " << area << endl;
12 }
```

Listing 3: RecArea.h

```
1  #include "RecAreaDeclared.h"
2
3  // definition for RecArea
4  double recArea(const double& side1, const double& side2)
5  {
6    return side1 * side2;
7  }
```

Listing 4: RecArea.cpp

VGU
Vietnamese - German University

Header Files III

```cpp
#include <iostream>
#include "RecAreaDeclared.h"
#include "RecAreaDefined.h"
using namespace std;
int main()
{
  double side1(5), side2(10);
  area = recArea(side1, side2);
  printArea();
  return 0;
}
```

Listing 5: RecAreaMain.cpp

VGU Vietnamese - German University

Header Files I

- Example: What is wrong with the following code? ⇒ *area and recArea are defined twice.*
- Substituted code:

```
1  #include <iostream>
2  using namespace std;
3
4  //=== from RecAreaDeclared.h
5  // global variable;
6  double area;
7
8  // functions
9  double recArea(const double& side1, const double& side2);
10 void printArea()
11 {
12    cout << "The area is " << area << endl;
13 }
14
```

Vietnamese - German University

Header Files II

```
15 //=== from RecAreaDefined.h
16 // global variable;
17 double area;
18
19 // functions
20 double recArea(const double& side1, const double& side2);
21 void printArea()
22 {
23   cout << "The area is " << area << endl;
24 }
25
26 // definition for RecArea
27 double recArea(const double& side1, const double& side2)
28 {
29   return side1 * side2;
30 }
31
32
33
```

Vietnamese - German University

Header Files III

```
34 int main()
35 {
36   double side1(5), side2(10);
37   area = recArea(side1, side2);
38   printArea();
39   return 0;
40 }
```

Header Files I

- Header Guards: to prevent multiple definitions of the same variable or function, or unnecessary inclusion of header files.
- Guarded header files:

```
1  #ifndef _RECAREA_DECLARED_     // header guard
2  #define _RECAREA_DECLARED_     // header guard
3  #include <iostream>
4  using namespace std;
5  // global variable;
6  double area;
7  // functions
8  double recArea(const double& side1, const double& side2);
9  void printArea()
10 {
11    cout << "The area is " << area << endl;
12 }
13 #endif
```

Listing 6: RecArea.h

Header Files II

```cpp
#ifndef _RECAREA_DEFINED_ // header guard
#define _RECAREA_DEFINED_ // header guard

#include "RecAreaDeclared.h"

// definition for RecArea
double recArea(const double& side1, const double& side2)
{
  return side1 * side2;
}

#endif
```

Listing 7: RecArea.cpp

VGU Vietnamese - German University

Header Files III

```cpp
#include <iostream>
#include "RecAreaDeclared.h"
#include "RecAreaDefined.h"
using namespace std;
int main()
{
  double side1(5), side2(10);
  area = recArea(side1, side2);
  printArea();
  return 0;
}
```

Listing 8: RecAreaMain.cpp

VGU
Vietnamese - German University

Header Files

- Header Guards: to prevent multiple definitions of the same variable or function, or unnecessary inclusion of header files.
- `RecAreaDeclared.h`: initially, since `_RECAREA_DECLARED_` was not defined, the whole file will be compiled due to the `#ifndef` directive which defines condition `_RECAREA_DECLARED_`. When included for the second time, since `_RECAREA_DECLARED_` has been defined, the whole file is ignored.
  $\Rightarrow$ *No matter how many times* `RecAreaDeclared.h` *is included, the file is in fact compiled just ONCE.*

Outline

Applications to Linear Algebra

In linear algebra, let $\mathbf{v}, \mathbf{w} \in \mathbb{R}^m$ and $A, B \in \mathbb{R}^{m \times n}$ where m and n is the number of rows (size of vectors) and number of columns, respectively, e.g.,

$$\mathbf{v} = \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_i \\ \vdots \\ v_{m-1} \end{bmatrix}, \quad A = \begin{bmatrix} a_{00} & a_{01} & \cdots & a_{0,j} & \cdots & a_{0,n-1} \\ a_{10} & a_{11} & \cdots & a_{1,j} & \cdots & a_{1,n-1} \\ \vdots & \vdots & \cdots & \vdots & \ddots & \vdots \\ a_{i,0} & a_{i,1} & \cdots & a_{i,j} & \cdots & a_{2,n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,2} & \cdots & a_{n-1,n-1} \end{bmatrix},$$

or in short form,

$$\mathbf{v} = (v_i), \mathbf{w} = (w_i), A = (a_{ij}), B = (b_{ij})$$

See Lecture 2 for basic vector and matrix operations

VGU Vietnamese - German University

Applications to Linear Algebra

Coding 3: Write C++ *functions* to perform the following tasks:

1. Dynamically allocate and de-allocate memories for storing $\mathbf{v}$ and A
2. Initialize the allocated $\mathbf{v}$ and A with zero entries
3. Set the entries of $\mathbf{v}$ and A with random values
4. Set entry v_i and a_{ij} with a given value
5. Print to the screen values of $\mathbf{v}$ and A
6. Compute the p-norm of $\mathbf{v}$ using the following formula

$$\|\mathbf{v}\|_p = \left(\sum_{i=0}^{n-1} |v_i|^p \right)^{1/p}, \tag{1}$$

for any $p > 0$ finite, and for $p \to \infty$,

$$\|\mathbf{v}\|_\infty = \max_{i=0}^{n-1} |v_i|, \tag{2}$$

VGU Vietnamese - German University

Applications to Linear Algebra

7. Compute the determinant $\texttt{det}(A)$ using the following formula

$$det(A) = \sum_{j=0}^{n-1} (-1)^{n-1} a_{0,j} det(\hat{A}_{j,n-1}), \tag{3}$$

where $\hat{A}_{0j}$ is a square $(n-1) \times (n-1)$ matrix formed by removing row 0 and column j of A. Use recursion for this function.

8. Compute the addition, subtraction of 2 vectors, 2 matrices of the same size

$$(\mathbf{v} \pm \mathbf{w})_i = v_i \pm w_i, \quad (A \pm B)_{ij} = A_{ij} \pm B_{ij} \tag{4}$$

9. Compute the scalar-vector, scalar-matrix, vector-matrix, matrix-matrix multiplication using function overloading. Here,

$$(\alpha \mathbf{v})_i = \alpha * v_i, \quad (\alpha A)_{ij} = \alpha * A_i j \tag{5}$$

VGU
Vietnamese - German University

Applications to Linear Algebra

$$(A\mathbf{v})_i = \sum_{j=0}^{n-1} A_{ij} * v_j, \quad (AB)_{ij} = \sum_{k=0}^{n-1} A_{ik} * B_{kj} \tag{6}$$

10. Compute the dot product of 2 vectors, i.e.,

$$\mathbf{v} \cdot \mathbf{w} = \sum_{i=0}^{n-1} v_i * w_i \tag{7}$$

11. Compute the cross product of 2 vectors, i.e.,

$$\mathbf{v} \times \mathbf{w} = \begin{bmatrix} v_0 w_0 & v_0 w_1 & v_0 w_2 & \cdots & v_0 w_{n-1} \\ v_1 w_0 & v_1 w_1 & v_1 w_2 & \cdots & v_1 w_{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ v_{n-1} w_0 & v_{n-1} w_1 & v_{n-1} w_2 & \cdots & v_{n-1} w_{n-1} \end{bmatrix} \tag{8}$$

VGU Vietnamese - German University

Applications to Linear Algebra

Suggestions:

- *Declare* all functions relating to *vector* operations in a separate header file and name it `vector.h`.
- *Define* all functions relating to *vector* operations in a separate source file and name it `vector.cpp` with `vector.h` included.
- *Declare* all functions relating to *matrix* operations in a separate header file and name it `matrix.h`.
- *Define* all functions relating to *matrix* operations in a separate source file and name it `matrix.cpp` with `matrix.h` included.
- Remember to use header guards for the header files.
- Remember to use `assert` to check whether the sizes of the vectors and matrices involving in the calculations are correct.
- For all functions, use the interface and input arguments given in the `main()` function in Listing 9 below.

VGU Vietnamese - German University

Applications to Linear Algebra I

- Use the following `main` file to test your implementation. Use MATLAB to double check the output results.

```
#include <iostream>
// header file for all global constants
#include "constants.h"
// header file for all functions relating to vector operations
#include "vector.h"
// header file for all functions relating to matrix operations
#include "matrix.h"
using namespace std;

int main()
{
  double alpha(TWO), beta(ZERO);
  int size_v, size_w, size_t;
  double *v, *w, *t;
  int numRows_A, numCols_A;
  int numRows_B, numCols_B;
  int numRows_C, numCols_C;
  int numRows_D, numCols_D;
```

Applications to Linear Algebra II

```
19    int numRows_E, numCols_E;
20    double **A, **B, **C, **D, **E;
21
22    /*
23    * 1. initialize vector and matrix size
24    */
25    size_v = 5; size_w = 5; size_t = 5;
26    numRows_A = 5; numCols_A = 4;
27    numRows_B = 5; numCols_B = 4;
28    numRows_C = 5; numCols_C = 4;
29    numRows_D = 4; numCols_D = 5;
30    numRows_E = 5; numCols_E = 5;
31
32    /*
33    * 2. allocate vectors and matrices
34    */
35    v = allocate(size_v);
36    w = allocate(size_w);
37    t = allocate(size_t);
38    A = allocate(numRows_A, numCols_A);
39    B = allocate(numRows_B, numCols_B);
40    C = allocate(numRows_C, numCols_C);
```

Vietnamese - German University

Applications to Linear Algebra III

```
41    D = allocate(numRows_D, numCols_D);
42    E = allocate(numRows_E, numCols_E);
43
44    /*
45    * 3. initialize vectors and matrices
46    */
47    v = randVec(size_v);     // random entries
48    w = randVec(size_w);     // random entries
49    t = zeroVec(size_t);     // zero entries
50    A = randMat(numRows_A, numCols_A);  // random entries
51    B = randMat(numRows_B, numCols_B);     // random entries
52    C = zeroMat(numRows_C, numCols_C);     // zero entries
53    D = randMat(numRows_D, numCols_D);  // random entries
54    E = zeroMat(numRows_E, numCols_E);     // zero entries
55
56    /*
57    * 4. use setEntry to set v_i and a_ij
58    */
59    setVec(v, size_v, 3) = 10.0;
60    // should return error by assert
61    setVec(w, size_w, 5) = 1.0;
62    setMat(A, numRows_A, numCols_A, 2, 2) = 5.0;
```

Vietnamese - German University

Applications to Linear Algebra IV

```
63
64  /*
65  * 5. print out the initialized vectors and matrices
66  */
67  printVec(v, size_v);
68  printVec(w, size_w);
69  printVec(t, size_t);
70  printMat(A, numRows_A, numCols_A);
71  printMat(B, numRows_B, numCols_B);
72  printMat(C, numRows_C, numCols_C);
73  printMat(D, numRows_D, numCols_D);
74  printMat(E, numRows_E, numCols_E);
75
76  /*
77  * 6. compute vector norms
78  */
79  double norm2_v(ZERO), norminf_v(ZERO);
80  norm2_v = normVec(v, size_v, 2);     // 2-norm
81  norminf_v = normVec(v, size_v, p_INF);  // infinite-norm
82
83  /*
84  * 7. operations
```

Vietnamese - German University

Applications to Linear Algebra V

```
85    */
86    //====== for vectors ========
87    //=== adding 2 vectors
88    t = addVec(v, size_v, w, size_w);
89    printVec(t, size_t);
90
91    //=== scalar-vector multiplication
92    t = mulScaVec(alpha, v, size_v);
93    printVec(t, size_t);
94
95    //=== dot product of 2 vectors of the same size
96    beta = dotProd(v, size_v, w, size_w);
97
98    //==== cross product of 2 vectors
99    E = crossProd(v, size_v, w, size_w);
00    printMat(E, numRows_E, numCols_E);
01
02    //====== for matrices ========
03    //==== adding 2 matrices
04    C = addMat(A, numRows_A, numCols_A,
05               B, numRows_B, numCols_B);
06    printMat(C, numRows_C, numCols_C);
```

Applications to Linear Algebra VI

```
07    // reset E
08    E = zeroMat(numRows_E, numCols_E);
09    // should return error
10    E = addMat(C, numRows_C, numCols_C,
11               B, numRows_B, numCols_B);
12    //==== matrix-matrix multiplication
13    E = mulMat(A, numRows_A, numCols_A
14               D, numRows_D, numCols_D);
15    printMat(E, numRows_E, numCols_E);
16    //==== matrix-scalar multiplication
17    // operator overloaded
18    E = mulMat(E, numRows_E, numCols_E, alpha);
19    printMat(E, numRows_E, numCols_E);
20    //==== matrix-vector multiplication
21    t = mulMat(E, numRows_E, numCols_E, v, size_v);
22    printVec(t, size_t);
23
24    /*
25    * 8. compute the determinant of a matrix
26    */
27    double det_A(ZERO), det_E(ZERO);
28    // returns error since A is not square
```

Applications to Linear Algebra VII

```
29    det_A = detMat(A, numRows_A, numCols_A);
30    det_E = detMat(A, numRows_E, numCols_E);
31
32    /*
33    * 9. de-allocation
34    */
35    deallocate(v);
36    deallocate(w);
37    deallocate(t);;
38    deallocate(A, numRows_A);
39    deallocate(B, numRows_B);
40    deallocate(C, numRows_C);
41    deallocate(D, numRows_D);
42    deallocate(E, numRows_E);
43
44    return 0;
45 }
```

Listing 9: Coding2.cpp

VGU Vietnamese - German University

Reading

1. Capper, *Introducing C++ for Scientists, Engineers, and Mathematicians*, **Chapter 5**
2. Pitt-Francis, and Whiteley, *Guide to Scientific Computing in C++*, **Chapter 5**