

# C++ for Scientific Computation

Rajeev Singh

## Contents

<b>1</b>	<b>Day 1: Basic input/output</b>	<b>3</b>
1.1	aa_hello_world.c . . . . .	3
1.2	ab_hello_world.cpp . . . . .	4
1.3	ac_hello_world.cpp . . . . .	5
1.4	ad_powers_of_integer.cpp . . . . .	6
1.5	ae_powers_of_real.cpp . . . . .	7
<b>2</b>	<b>Day 2: Pointers/References, Arithmetic/Logical</b>	<b>8</b>
2.1	af_pointer.cpp . . . . .	8
2.2	ag_reference.cpp . . . . .	9
2.3	ah_arithmetic_operators.cpp . . . . .	10
2.4	ai_relational_logical.cpp . . . . .	11
<b>3</b>	<b>Day 3: Scope, Conditional, Loops</b>	<b>12</b>
3.1	aj_blocks_scope.cpp . . . . .	12
3.2	ak_scope.cpp . . . . .	13
3.3	al_if_else.cpp . . . . .	14
3.4	am_for_loop.cpp . . . . .	15
3.5	an_while_loop.cpp . . . . .	16
3.6	ao_do_while_loop.cpp . . . . .	17
3.7	ap_break.cpp . . . . .	18
3.8	aq_break_nested_loop.cpp . . . . .	19
3.9	ar_break_all_loops.cpp . . . . .	20
<b>4</b>	<b>Day 4: Functions, Call by value/reference</b>	<b>21</b>
4.1	as_function_square.cpp . . . . .	21
4.2	at_function_factorial.cpp . . . . .	22
4.3	au_function_call_by_value.cpp . . . . .	23
4.4	av_function_call_by_reference.cpp . . . . .	24
4.5	aw_function_call_by_reference_using_pointers.cpp . . . . .	25
4.6	ax_function_multiple_return_values.cpp . . . . .	26

<b>5</b>	<b>Day 5: Functions- default args, function pointers; Arrays</b>	<b>27</b>
5.1	ay_function_default_arguments.cpp . . . . .	27
5.2	az_function_inline.cpp . . . . .	28
5.3	ba_function_pointers.cpp . . . . .	29
5.4	bb_function_pointers_as_arguments.cpp . . . . .	30
5.5	bc_static_variables.cpp . . . . .	31
5.6	bd_array.cpp . . . . .	32
5.7	be_function_with_array_argument.cpp . . . . .	33
5.8	bf_multidimensional_arrays.cpp . . . . .	34
5.9	bg_array_and_pointer.cpp . . . . .	35
<b>6</b>	<b>Day 6: Dynamic memory, Multidimensional Array, BLAS</b>	<b>36</b>
6.1	bh_dynamic_memory.cpp . . . . .	36
6.2	bi_dynamic_array_size_input.cpp . . . . .	37
6.3	bj_multidimensional_array_with_pointer.cpp . . . . .	38
6.4	bk_multidimensional_dynamic_array.cpp . . . . .	39
6.5	bl_multidimensional_dynamic_array_size_input.cpp . . . . .	40
6.6	bm_multidimensional_array_with_mapping.cpp . . . . .	41
<b>7</b>	<b>Day 7: String, Advanced Datatypes, BLAS, Sparse Matrices</b>	<b>42</b>
7.1	bn_strings.cpp . . . . .	42
7.2	bo_typedef.cpp . . . . .	43
7.3	bp_struct.cpp . . . . .	44
7.4	bq_struct_pointer.cpp . . . . .	46
7.5	br_struct_array.cpp . . . . .	48
<b>8</b>	<b>Day 9: Modules and Namespaces, Classes</b>	<b>50</b>
8.1	bs_struct_with_functions.cpp . . . . .	51
8.2	bt_struct_constructor_destructor.cpp . . . . .	53
<b>9</b>	<b>Day 10: Classes Continued</b>	<b>55</b>
9.1	bu_struct_this_pointer.cpp . . . . .	55
9.2	bv_copy_constructor.cpp . . . . .	57
9.3	bw_default_copy_constructor.cpp . . . . .	59
9.4	bx_struct_with_const_functions.cpp . . . . .	61
9.5	by_struct_visibility.cpp . . . . .	63
9.6	bz_safely_changing_size_of_array.cpp . . . . .	65
<b>10</b>	<b>Day 11: Classes Continued, Templates</b>	<b>67</b>
10.1	ca_operator_overloading.cpp . . . . .	67
10.2	cb_templates.cpp . . . . .	69

# 1 Day 1: Basic input/output

## 1.1 aa\_hello\_world.c

```
// C program to print "Hello World".  
//  
// Rajeev Singh  
// 2013-03-27  
  
#include <stdio.h>  
  
int main() {  
    printf("Hello World from C\n");  
    return 0;  
}
```

## 1.2 ab\_hello\_world.cpp

```
// C++ program to print "Hello World".
//
// Rajeev Singh
// 2013-03-27

#include <iostream>

int main() {
    std::cout << "Hello World from C++"; // << std::endl;
    return 0;
}
```

## 1.3 ac\_hello\_world.cpp

```
// C++ program to print "Hello World".  
//  
// Rajeev Singh  
// 2013-03-27  
  
#include <iostream>  
  
using namespace std;  
  
int main() {  
    cout << "Hello World from C++" << endl;  
    return 0;  
}
```

## 1.4 ad\_powers\_of\_integer.cpp

```
// Program to calculate powers of given integer.
//
// Rajeev Singh
// 2013-03-27

#include <iostream>
#include <cmath>
using namespace std;

int main() {
    //int given_number;
    long int given_number;
    cout << "Enter an integer: ";
    cin  >> given_number;

    cout << "Given number = " << given_number << endl
         << "Square      = " << pow(given_number,2) << endl
         << "Cube        = " << pow(given_number,3) << endl
         << "Forth power  = " << pow(given_number,4) << endl;

    return 0;
}
```

## 1.5 ae\_powers\_of\_real.cpp

```
// Program to calculate powers of given integer.
//
// Rajeev Singh
// 2013-03-27

#include <iostream>
#include <cmath>
using namespace std;

int main() {
    double given_number;
    //long double given_number;
    cout << "Enter a real number : ";
    cin  >> given_number;

    cout << "Given number = " << given_number << endl
         << "Square       = " << pow(given_number,2) << endl
         << "Square root  = " << pow(given_number,1./2) << endl
         << "Cube        = " << pow(given_number,3) << endl
         << "Forth power  = " << pow(given_number,4) << endl;

    return 0;
}
```

## 2 Day 2: Pointers/References, Arithmetic/Logical

### 2.1 af\_pointer.cpp

```
// Program to illustrate pointers.
//
// Rajeev Singh
// 2013-03-28

#include <iostream>
using namespace std;

int main() {
    int *np = NULL;
    int n = 10;

    cout << "Initial" << endl
         << "n    = " << n    << endl
         << "np   = " << np   << endl
         << "*np  = " << "since np is NULL, printing *np gives segmentation fault"
    << endl << endl;

    np = &n;
    cout << "After: np = &n" << endl
         << "n    = " << n    << endl
         << "np   = " << np   << endl
         << "*np  = " << *np  << endl << endl;

    *np = 22;
    cout << "After: *np = 22" << endl
         << "n    = " << n    << endl
         << "np   = " << np   << endl
         << "*np  = " << *np  << endl << endl;

    return 0;
}
```



## 2.2 ag\_reference.cpp

```
// Program to illustrate the use of references (special pointers).
//
// Rajeev Singh
// 2013-03-28

#include <iostream>
using namespace std;

int main() {
    int    n = 5;
    int & r = n;
    int    m;

    cout << "Initial" << endl
         << "n = " << n << endl
         << "r = " << r << endl
         << "m = " << m << endl << endl;

    m = r + 3;      // m == n + 3
    cout << "After: m = r + 3" << endl
         << "n = " << n << endl
         << "r = " << r << endl
         << "m = " << m << endl << endl;

    r = m;          // r still points to n and n == m
    cout << "After: r = m" << endl
         << "n = " << n << endl
         << "r = " << r << endl
         << "m = " << m << endl << endl;

    m = 0;          // r and n are unchanged
    cout << "After: m = 0" << endl
         << "n = " << n << endl
         << "r = " << r << endl
         << "m = " << m << endl << endl;

    int & s = m;
    r = s;          // r still points to n and n == m (== 0)
    cout << "After: r = s where s is new reference to m" << endl
         << "n = " << n << endl
         << "r = " << r << endl
         << "m = " << m << endl << endl;

    return 0;
}
```

## 2.3 ah\_arithmetic\_operators.cpp

```
// Program to illustrate basic arithmetic operators.
//
// Rajeev Singh
// 2013-03-28

#include <iostream>
using namespace std;

int main() {
    int m = 100,
        n = 200;

    cout << "Initial" << endl
         << "m = " << m << endl
         << "n = " << n << endl
         << "m + n = " << m + n << endl
         << "m - n = " << m - n << endl
         << "m * n = " << m * n << endl
         << "m / n = " << m / n << endl
         << "m % n = " << m % n << endl << endl;

    //m = m + 200;
    m += 200;      // both this commands are same
    cout << "After: m += 200" << endl
         << "m = " << m << endl
         << "n = " << n << endl
         << "m + n = " << m + n << endl
         << "m - n = " << m - n << endl
         << "m * n = " << m * n << endl
         << "m / n = " << m / n << endl
         << "m % n = " << m % n << endl << endl;

    m++;
    cout << "After: m++" << endl
         << "m = " << m << endl
         << "n = " << n << endl
         << "m + n = " << m + n << endl
         << "m - n = " << m - n << endl
         << "m * n = " << m * n << endl
         << "m / n = " << m / n << endl
         << "m % n = " << m % n << endl << endl;

    return 0;
}
```

## 2.4 ai\_relational\_logical.cpp

```
// program to illustrate logical and relational operators.
//
// Rajeev Singh
// 2013-03-28

#include <iostream>
using namespace std;

int main() {
    int    x = 2;
    int    y = 4;
    int    z = 4;
    bool    b;

    cout << "x = " << x << endl
         << "y = " << y << endl
         << "z = " << z << endl << endl;

    // z == 4 is not tested
    b = ( x == 2 && y == 3 && z == 4 );
    cout << "b = ( x == 2 && y == 3 && z == 4 )" << endl
         << "b = " << b << endl << endl;

    // only x == 2 is tested
    b = ( x == 2 || y == 3 || z == 4 );
    cout << "b = ( x == 2 || y == 3 || z == 4 )" << endl
         << "b = " << b << endl << endl;

    // correct, since x != 0 in "y/x"
    b = ( x != 0 && y/x > 1 );
    cout << "b = ( x != 0 && y/x > 1 )" << endl
         << "b = " << b << endl << endl;

    return 0;
}
```

## 3 Day 3: Scope, Conditional, Loops

### 3.1 aj\_blocks\_scope.cpp

```
// program to illustrate blocks.
//
// Rajeev Singh
// 2013-03-29

#include <iostream>
using namespace std;

int main() {
    { // block 1
        int n1 = 1;
        double f1 = 0.0;
        cout << "Block 1 " << endl;
        cout << "n1 = " << n1 << endl;
        cout << "f1 = " << f1 << endl;
    }

    { // block 2
        int n1 = 2;
        // n1 has value 2 in this block
        cout << "Block 2 " << endl;
        cout << "n1 = " << n1 << endl;

        //int n1 = 5; // ERROR
    }

    return 0;
}
```

## 3.2 ak\_scope.cpp

```
// program to illustrate scope of variables
//
// Rajeev Singh
// 2013-03-29

#include <iostream>
using namespace std;

int main() {
    { // block 1
        int m, n1 = 1;
        { // block 1.1
            int n2 = 2;
            { // block 1.1.1
                m = n1 + n2; // evaluates to m = 3
                cout << "Block 1.1.1: m = " << m << endl;
            }
        }

        { // block 1.2
            int n2 = 3;
            m = n1 + n2; // evaluates to m = 4
            cout << "Block 1.2 : m = " << m << endl;
        }
    }

    return 0;
}
```

### 3.3 al\_if\_else.cpp

```
// program to illustrate conditional structure
//
// Rajeev Singh
// 2013-03-29

#include <iostream>
using namespace std;

int main() {
    int n = 1;

    if ( n > 0 )
    {
        n = n / n;
    }

    if ( n < 0 ) {
        n += 5; // NOTE: trivial block!
        cout << "hello " << n << endl;
    }
    else if ( n %2 == 0 ) {
        n += 1;
        cout << "hello " << n << endl;
    }
    else {
        n -= 6;
        cout << "hello " << n << endl;
    }

    cout << "n = " << n << endl;

    return 0;
}
```

### 3.4 am\_for\_loop.cpp

```
// program to illustrate for loop
//
// Rajeev Singh
// 2013-03-29

#include <iostream>
using namespace std;

int main() {
    int n = 1;

    for (int i=1; i<10; i++) {
        if (i>5) {
            n *= i;
            cout << "n = " << n << endl;
        }
    }

    return 0;
}
```

### 3.5 an\_while\_loop.cpp

```
// program to illustrate while loop
//
// Rajeev Singh
// 2013-03-29

#include <iostream>
using namespace std;

int main() {
    int n = 1;
    int i = 1;

    while (i < 10) {
        n *= i;
        i++;
        cout << "n = " << n << endl;
    }

    return 0;
}
```



### 3.6 ao\_do\_while\_loop.cpp

```
// program to illustrate do-while loop
//
// Rajeev Singh
// 2013-03-29

#include <iostream>
using namespace std;

int main() {
    int n = 1;
    int i = 100;

    do {
        n *= i;
        i++;
        cout << "n = " << n << endl;
    } while (i < 10);

    return 0;
}
```

### 3.7 ap\_break.cpp

```
// program to illustrate use of break
//
// Rajeev Singh
// 2013-03-29

#include <iostream>
using namespace std;

int main() {
    int n = 1;

    for ( int i = 1; i < 20; i++ ) {
        // avoid overflow
        if ( n > 21474836 )
            break;
        n *= i;
        cout << "n = " << n << endl;
    }

    return 0;
}
```

### 3.8 aq\_break\_nested\_loop.cpp

```
// program to illustrate behavior of break in nested loops
//
// Rajeev Singh
// 2013-03-29

#include <iostream>
using namespace std;

int main() {
    for ( int i = 1; i < 20; i++ ) {
        int n = 1;
        for ( int j = 1; j < i; j++ ) {
            if ( n > 21474836 )
                break;
            n *= j;
        }

        cout << "n = " << n << endl;
    }

    return 0;
}
```

### 3.9 ar\_break\_all\_loops.cpp

```
// program to illustrate breaking all nested loops
//
// Rajeev Singh
// 2013-03-29

#include <iostream>
using namespace std;

int main() {
    int flag = 0;
    for ( int i = 1; i < 20; i++ ) {
        int n = 1;
        for ( int j = 1; j < i; j++ ) {
            if ( n > 21474836 ) {
                flag = 1;
                break;
            }
            n *= j;
        }
        if (flag == 1)
            break;

        cout << "n = " << n << endl;
    }

    return 0;
}
```

## 4 Day 4: Functions, Call by value/reference

### 4.1 as\_function\_square.cpp

```
// program to illustrate defining function
//
// Rajeev Singh
// 2013-03-31

#include <iostream>
using namespace std;

double
square (const double x) {
    return x*x;
}

int main() {
    double a = 2.5;

    cout << "a    = " << a << endl
          << "a^2 = " << square(a) << endl;

    return 0;
}
```

## 4.2 at\_function\_factorial.cpp

```
// program to illustrate defining the factorial function
//
// Rajeev Singh
// 2013-03-31

#include <iostream>
using namespace std;

int
factorial (const int n) {
    int f = 1;
    for (int i = 1; i <= n; i++ )
        f *= i;

    return f;
}

int main() {
    int m = 10;

    cout << "m   = " << m << endl
         << "m!  = " << factorial(m) << endl;

    return 0;
}
```

### 4.3 au\_function\_call\_by\_value.cpp

```
// program to illustrate call by value feature
//
// Rajeev Singh
// 2013-03-31

#include <iostream>
using namespace std;

int
f (int n) {
    n = 10;
    return n;
}

int main() {
    int m = 5;

    cout << "m before calling = " << m << endl;
    cout << "function output   = " << f(m) << endl;
    cout << "m after calling   = " << m << endl;

    return 0;
}
```

## 4.4 av\_function\_call\_by\_reference.cpp

```
// program to illustrate call by reference feature
//
// Rajeev Singh
// 2013-03-31

#include <iostream>
using namespace std;

int
f (int & n) {
    n = 10;
    return n;
}

int main() {
    int m = 5;

    cout << "m before calling = " << m << endl;
    cout << "funtion output    = " << f(m) << endl;
    cout << "m after calling   = " << m << endl;

    return 0;
}
```



## 4.5 aw\_function\_call\_by\_reference\_using\_pointers.cpp

```
// program to illustrate call by reference feature using general  
// pointers  
//  
// Rajeev Singh  
// 2013-03-31  
  
#include <iostream>  
using namespace std;  
  
int  
f (int * n) {  
    *n = 10;  
    return *n;  
}  
  
int main() {  
    int m = 5;  
  
    cout << "m before calling = " << m << endl;  
    cout << "funtion output    = " << f(&m) << endl;  
    cout << "m after calling   = " << m << endl;  
  
    return 0;  
}
```

## 4.6 ax\_function\_multiple\_return\_values.cpp

```
// program to illustrate funtions with multiple return values using
// call by reference
//
// Rajeev Singh
// 2013-03-31

#include <iostream>
using namespace std;

void
min_max ( const int n1, const int n2,
          int & min, int & max );

int main() {
    int m1, m2, min, max;

    cout << "Enter two integers :";
    cin >> m1 >> m2;

    min_max(m1, m2, min, max);
    cout << "m1          = " << m1 << endl
         << "m2          = " << m2 << endl << endl
         << "min(m1,m2) = " << min << endl
         << "max(m1,m2) = " << max << endl;

    return 0;
}

void
min_max ( const int n1, const int n2,
          int & min, int & max ) {
    if ( n1 < n2 ) {
        min = n1;
        max = n2;
    }
    else {
        min = n2;
        max = n1;
    }
}
```

## 5 Day 5: Functions- default args, function pointers; Arrays

### 5.1 ay\_function\_default\_arguments.cpp

```
// program to illustrate function with default arguments
//
// Rajeev Singh
// 2013-04-01

#include <iostream>
using namespace std;

double
square (const double x = 10.0) {
    return x*x;
}

int main() {
    double a = 2.5;

    cout << "a          = " << a << endl;
    cout << "a^2        = " << square(a) << endl;
    cout << "square() = " << square() << endl;

    return 0;
}
```

## 5.2 az\_function\_inline.cpp

```
/* program to illustrate inline functions
 *
 * WARNING: do not inline functions with large bodies. it can cause
 *          the final executable to be very large in size and decrease
 *          performance.
 *
 *
 * Rajeev Singh
 * 2013-04-01
 *
 */

#include <iostream>
using namespace std;

inline double
square (const double x = 10.0) {
    return x*x;
}

int main() {
    double a = 2.5;

    cout << "a          = " << a << endl;
    cout << "a^2        = " << square(a) << endl;
    cout << "square() = " << square() << endl;

    return 0;
}
```

## 5.3 ba\_function\_pointers.cpp

```
/* program to illustrate function pointers
 *
 * Rajeev Singh
 * 2013-04-01
 *
 */

#include <iostream>
using namespace std;

double
square (const double x) {
    return x*x;
}

int main() {
    double a = 2.5;
    double (* pf) (const double x);
    pf = square;

    cout << "a          = " << a << endl;
    cout << "square(a) = " << square(a) << endl;
    cout << "pf(a)      = " << pf(a) << endl;

    return 0;
}
```

## 5.4 bb\_function\_pointers\_as\_arguments.cpp

```
/* program to illustrate function pointers as arguments
 *
 * Rajeev Singh
 * 2013-04-01
 *
 */

#include <iostream>
using namespace std;

double
square (const double x) {
    return x*x;
}

double
cube (const double x) {
    return x*x*x;
}

double
f ( double ( * func ) ( const double x ),
    const double x ) {
    return func( x );
}

int main() {
    double a = 2.5;

    cout << "a          = " << a << endl;
    cout << "f(square, a) = " << f(square, a) << endl;
    cout << "f(cube,   a) = " << f(cube,   a) << endl;

    return 0;
}
```

## 5.5 bc\_static\_variables.cpp

```
/* program to illustrate static variables
 *
 * Rajeev Singh
 * 2013-04-01
 *
 */

#include <iostream>
using namespace std;

double
f ( const double x, long & cnt ) {
    static long counter = 0; // allocated and initialised
                               // once per program

    cnt = ++counter;
    return 2.0*x*x - x;
}

int main() {
    long cnt = 0;

    for ( double x = -10; x <= 10.0; x += 0.1 )
        f( x, cnt );

    cout << "num times f called = " << cnt << endl;

    return 0;
}
```

## 5.6 bd\_array.cpp

```
/* program to illustrate array
 *
 * Rajeev Singh
 * 2013-04-01
 *
 */

#include <iostream>
using namespace std;

int main() {
    double f[5];

    for ( int i = 0; i < 5; i++ )
        f[i] = 2*i;

    cout << "f = " << f << endl;
    for ( int i = 0; i < 5; i++ )
        cout << "f[" << i << "] = " << f[i] << endl;

    cout << "f[5] = " << f[5] << endl; // bug but program still compiles
    // if you lucky such bugs will be detected by segmentation fault

    return 0;
}
```



## 5.7 be\_function\_with\_array\_argument.cpp

```
/* program to illustrate arrays as function arguments
 *
 * Rajeev Singh
 * 2013-04-01
 *
 */

#include <iostream>
using namespace std;

void
copy ( const double x[3], double y[3] ) {
    for ( int i = 0; i < 3; i++ )
        y[i] = x[i];
}

void
add ( const double x[3], double y[3] ) {
    for ( int i = 0; i < 3; i++ )
        y[i] += x[i];
}

int main() {
    double a[3],
           b[] = {0, 0, 0}; // b is automaticall of size 3

    for ( int i = 0; i < 3; i++ )
        a[i] = 2*i;

    cout << "Intial a and b:" << endl;
    for ( int i = 0; i < 3; i++ )
        cout << "a[" << i << "] = " << a[i]
              << " b[" << i << "] = " << b[i]<< endl;

    copy( a, b );
    cout << endl << "After calling copy funtion:" << endl;
    for ( int i = 0; i < 3; i++ )
        cout << "a[" << i << "] = " << a[i]
              << " b[" << i << "] = " << b[i]<< endl;

    add( a, b );
    cout << endl << "After calling sum funtion:" << endl;
    for ( int i = 0; i < 3; i++ )
        cout << "a[" << i << "] = " << a[i]
              << " b[" << i << "] = " << b[i]<< endl;

    return 0;
}
```

## 5.8 bf\_multidimensional\_arrays.cpp

```
/* program to illustrate multidimensional arrays
 *
 * Rajeev Singh
 * 2013-04-01
 *
 */

#include <iostream>
using namespace std;

void
mulvec ( const double M[3][3],
         const double x[3],
         double y[3] ) {
    for ( int i = 0; i < 3; i++ ) {
        y[i] = 0.0;

        for ( int j = 0; j < 3; j++ )
            y[i] += M[i][j] * x[j];
    }
}

int main() {
    double M[3][3],
           x[3], y[3];

    for ( int i = 0; i < 3; i++ ) {
        x[i] = 2*i;
        for ( int j = 0; j < 3; j++ )
            M[i][j] = 3*i+j;
    }

    mulvec(M, x, y);

    cout << "M:" << endl;
    for ( int i = 0; i < 3; i++ ) {
        for ( int j = 0; j < 3; j++ )
            cout << " " << M[i][j];
        cout << endl;
    }

    cout << "x:" << endl;
    for ( int j = 0; j < 3; j++ )
        cout << " " << x[j] << endl;

    cout << "y = M*x:" << endl;
    for ( int j = 0; j < 3; j++ )
        cout << " " << y[j] << endl;

    return 0;
}
```

## 5.9 bg\_array\_and\_pointer.cpp

```
/* program to illustrate pointers as arrays
 *
 * in C/C++ there is NO distinction between a pointer and an array.
 *
 * Rajeev Singh
 * 2013-04-01
 *
 */

#include <iostream>
using namespace std;

int main() {
    int    n[5] = { 2, 3, 5, 7, 11 };
    int * p    = n;
    int * q    = &n[1];

    cout << "n:" << endl;
    for ( int j = 0; j < 5; j++ )
        cout << " " << n[j] << endl;

    cout << "p:" << endl;
    for ( int j = 0; j < 5; j++ )
        cout << " " << p[j] << endl;

    cout << "q:" << endl;
    for ( int j = 0; j < 5; j++ )
        cout << " " << q[j] << endl;

    return 0;
}
```

## 6 Day 6: Dynamic memory, Multidimensional Array, BLAS

### 6.1 bh\_dynamic\_memory.cpp

```
/* program to illustrate dynamic memory
 *
 * this example shows the C++ way of doing the job. this will not work
 * for C.
 *
 * Rajeev Singh
 * 2013-04-02
 *
 */

#include <iostream>
using namespace std;

int main() {
    int    n = 10;
    double * v = new double[n];

    for ( int i = 0; i < n; i++ )
        v[i] = double( i*i );

    cout << "n = " << n << endl;
    cout << "v:" << endl;
    for ( int j = 0; j < n; j++ )
        cout << " " << v[j] << endl;

    delete[] v;
    return 0;
}
```

## 6.2 bi\_dynamic\_array\_size\_input.cpp

```
/* program to illustrate dynamic memory
*
* Rajeev Singh
* 2013-04-02
*
*/

#include <iostream>
using namespace std;

int main() {
    int n;

    cout << "Enter the size of the array: ";
    cin >> n;

    double * v = new double[n];

    for ( int i = 0; i < n; i++ )
        v[i] = double( i*i );

    cout << "n = " << n << endl;
    cout << "v:" << endl;
    for ( int j = 0; j < n; j++ )
        cout << " " << v[j] << endl;

    delete[] v;
    return 0;
}
```

## 6.3 bj\_multidimensional\_array\_with\_pointer.cpp

```
/* program to illustrate dynamic memory
 *
 * Rajeev Singh
 * 2013-04-02
 *
 */

#include <iostream>
using namespace std;

int main() {
    int n1[4], n2[4], n3[4], n4[4];
    int * p1 = n1;           // p1 -> pointer
    int * p2 = n2;
    int * p3 = n3;
    int * p4 = n4;

    int *p[4] = {p1, p2, p3, p4}; // p -> pointer of pointers

    for (int i = 0; i < 4; i++ )
        for (int j = 0; j < 4; j++ )
            p[i][j] = 4*i+j;

    cout << "p:" << endl;
    for (int i = 0; i < 4; i++ ) {
        for (int j = 0; j < 4; j++ )
            cout << " " << p[i][j];
        cout << endl;
    }

    return 0;
}
```

## 6.4 bk\_multidimensional\_dynamic\_array.cpp

```
/* program to illustrate dynamic memory
 *
 * Rajeev Singh
 * 2013-04-02
 *
 */

#include <iostream>
using namespace std;

int main() {
    int * p1 = new int[4];
    int * p2 = new int[4];
    int * p3 = new int[4];
    int * p4 = new int[4];

    int **p = new int*[4];

    p[0] = p1;
    p[1] = p2;
    p[2] = p3;
    p[3] = p4;

    for (int i = 0; i < 4; i++ )
        for (int j = 0; j < 4; j++ )
            p[i][j] = 4*i+j;

    cout << "p:" << endl;
    for (int i = 0; i < 4; i++ ) {
        for (int j = 0; j < 4; j++ )
            cout << " " << p[i][j];
        cout << endl;
    }

    return 0;
}
```

## 6.5 bl\_multidimensional\_dynamic\_array\_size\_input.cpp

```
/* program to illustrate dynamic memory
 *
 * Rajeev Singh
 * 2013-04-02
 *
 */

#include <iostream>
using namespace std;

int main() {
    int m, n;
    cout << "Enter the size of the matrix: ";
    cin >> m >> n;

    int **p = new int*[m];

    for (int i = 0; i < m; i++ )
        p[i] = new int[n];

    for (int i = 0; i < m; i++ )
        for (int j = 0; j < n; j++ )
            p[i][j] = n*i+j;

    cout << "p:" << endl;
    for (int i = 0; i < m; i++ ) {
        for (int j = 0; j < n; j++ )
            cout << " " << p[i][j];
        cout << endl;
    }

    return 0;
}
```



## 6.6 bm\_multidimensional\_array\_with\_mapping.cpp

```
/* program to illustrate dynamic memory
 *
 * NOTE: using pointer of pointers can be slower than using mappings
 * for big arrays for the following reason:
 *
 * pointer of pointers -> two access to RAM to get an element
 * mapping              -> single access to RAM for the same
 *
 * accessing RAM is much more expensive than simple integer
 * multiplication and addition
 *
 * Rajeev Singh
 * 2013-04-02
 */

#include <iostream>
using namespace std;

int main() {
    int m, n;
    cout << "Enter the size of the matrix: ";
    cin >> m >> n;

    int *p = new int[m*n];

    for (int i = 0; i < m; i++ )
        for (int j = 0; j < n; j++ )
            p[n*i+j] = n*i+j;

    cout << "p:" << endl;
    for (int i = 0; i < m; i++ ) {
        for (int j = 0; j < n; j++ )
            cout << " " << p[n*i+j];
        cout << endl;
    }

    return 0;
}
```

GO TO THE TALK TO DISCUSS BLAS

## 7 Day 7: String, Advanced Datatypes, BLAS, Sparse Matrices

### 7.1 bn\_strings.cpp

```
/* program to illustrate strings as array of characters
 *
 * Rajeev Singh
 * 2013-04-03
 *
 */

#include <iostream>
using namespace std;

int main() {
    char str1[] = { 'S', 't', 'r', 'i', 'n', 'g', '\0' };
    char str2[] = "String"; // '\0' is appended automatically
    char str3[] = "This is a very long \
string";

    cout << str1 << endl;
    cout << str2 << endl;
    cout << str3 << endl;

    return 0;
}
```

## 7.2 bo\_typedef.cpp

```
/* program to illustrate renaming datatypes using typedef
 *
 * Rajeev Singh
 * 2013-04-03
 *
 */

#include <iostream>
using namespace std;

int main() {
    typedef char * string_t;
    string_t str2 = "String"; // '\0' is appended automatically
    string_t str3 = "This is a very long \
string";

    cout << str2 << endl;
    cout << str3 << endl;

    return 0;
}
```

## 7.3 bp\_struct.cpp

```
/* program to illustrate defining new datatypes using struct
 *
 * Rajeev Singh
 * 2013-04-03
 *
 */

#include <iostream>
using namespace std;

typedef double real_t;

struct vector_t {
    size_t size;
    real_t * coeffs;
};

void
add_vec ( const vector_t & x,
          const vector_t & y,
          vector_t &      z ) {
    for ( int i = 0; i < x.size; i++ )
        z.coeffs[i] = x.coeffs[i] + y.coeffs[i];
}

int main() {
    int      n = 10;
    vector_t a, b, c;
    a.size = n;
    b.size = n;
    c.size = n;

    a.coeffs = new real_t[n];
    b.coeffs = new real_t[n];
    c.coeffs = new real_t[n];

    for ( int i = 0; i < n; i++ ) {
        a.coeffs[i] = i;
        b.coeffs[i] = 2*i;
    }

    add_vec( a, b, c );

    cout << "a:" << endl;
    cout << "a.size = " << a.size << endl;
    cout << "a.coeffs:" << endl;
    for ( int i = 0; i < n; i++ )
        cout << "      " << a.coeffs[i] << endl;

    cout << "b:" << endl;
    cout << "b.size = " << b.size << endl;
    cout << "b.coeffs:" << endl;
```

```

    for ( int i = 0; i < n; i++ )
        cout << "          " << b.coeffs[i] << endl;

    cout << "c:" << endl;
    cout << "c.size = " << c.size << endl;
    cout << "c.coeffs:" << endl;
    for ( int i = 0; i < n; i++ )
        cout << "          " << c.coeffs[i] << endl;

    delete[] a.coeffs;
    delete[] b.coeffs;
    delete[] c.coeffs;
    return 0;
}

```

## 7.4 bq\_struct\_pointer.cpp

```
/* program to illustrate using pointers to struct
 *
 * Rajeev Singh
 * 2013-04-03
 *
 */

#include <iostream>
using namespace std;

typedef double real_t;

struct vector_t {
    size_t size;
    real_t * coeffs;
};

void
add_vec ( const vector_t * x,
          const vector_t * y,
          vector_t *      z ) {
    for ( int i = 0; i < x->size; i++ )
        z->coeffs[i] = x->coeffs[i] + y->coeffs[i];
}

int main() {
    int n = 10;
    vector_t * a = new vector_t,
              * b = new vector_t,
              * c = new vector_t;

    a->size = n;
    b->size = n;
    c->size = n;

    a->coeffs = new real_t[n];
    b->coeffs = new real_t[n];
    c->coeffs = new real_t[n];

    for ( int i = 0; i < n; i++ ) {
        a->coeffs[i] = i;
        b->coeffs[i] = 2*i;
    }

    add_vec( a, b, c );

    cout << "a:" << endl;
    cout << "a->size = " << a->size << endl;
    cout << "a->coeffs:" << endl;
    for ( int i = 0; i < n; i++ )
        cout << "      " << a->coeffs[i] << endl;
```

```

cout << "b:" << endl;
cout << "b->size = " << b->size << endl;
cout << "b->coeffs:" << endl;
for ( int i = 0; i < n; i++ )
    cout << "          " << b->coeffs[i] << endl;

cout << "c:" << endl;
cout << "c->size = " << c->size << endl;
cout << "c->coeffs:" << endl;
for ( int i = 0; i < n; i++ )
    cout << "          " << c->coeffs[i] << endl;

delete[] a->coeffs;
delete[] b->coeffs;
delete[] c->coeffs;

delete a;
delete b;
delete c;

return 0;
}

```

## 7.5 br\_struct\_array.cpp

```
/* program to illustrate using struct array
 *
 * Rajeev Singh
 * 2013-04-03
 *
 */

#include <iostream>
using namespace std;

typedef double real_t;

struct vector_t {
    size_t size;
    real_t * coeffs;
};

void
add_vec ( const vector_t * x,
          const vector_t * y,
          vector_t *      z ) {
    for ( int i = 0; i < x->size; i++ )
        z->coeffs[i] = x->coeffs[i] + y->coeffs[i];
}

int main() {
    int n = 10;
    vector_t * a = new vector_t[3];

    for ( int i = 0; i < 3; i++ ) {
        a[i].size = n;
        a[i].coeffs = new real_t[n];
    }

    for ( int i = 0; i < n; i++ ) {
        a[0].coeffs[i] = i;
        a[1].coeffs[i] = 2*i;
    }

    add_vec( &a[0], &a[1], &a[2] );

    cout << "a[0]:" << endl;
    cout << "a[0].size = " << a[0].size << endl;
    cout << "a[0].coeffs:" << endl;
    for ( int i = 0; i < n; i++ )
        cout << "          " << a[0].coeffs[i] << endl;

    cout << "a[1]:" << endl;
    cout << "a[1].size = " << a[1].size << endl;
    cout << "a[1].coeffs:" << endl;
    for ( int i = 0; i < n; i++ )
        cout << "          " << a[1].coeffs[i] << endl;
```



```

cout << "a[2]:" << endl;
cout << "a[2].size = " << a[2].size << endl;
cout << "a[2].coeffs:" << endl;
for ( int i = 0; i < n; i++ )
    cout << "          " << a[2].coeffs[i] << endl;

for ( int i = 0; i < 3; i++ )
    delete[] a[i].coeffs;

delete[] a;

return 0;
}

```

**GO TO THE TALK TO DISCUSS BLAS AND SPARSE MATRICES**

## 8 Day 9: Modules and Namespaces, Classes

GO TO THE TALK TO DISCUSS MODULES AND NAMESPACES

## 8.1 bs\_struct\_with\_functions.cpp

```
/* program to illustrate using struct with functions as members
 *
 * Rajeev Singh
 * 2013-04-05
 *
 */

#include <iostream>
using namespace std;
typedef double real_t;

struct vector_t {
    size_t size;
    real_t * coeffs;

    void init ( const unsigned n );
    void del   ();
    void fill  ( const real_t f );
    void scale ( const real_t f );
    void print ();
};

int main() {
    vector_t x;

    x.init( 10 );
    x.print();
    x.fill( 1.0 );
    x.print();
    x.scale( 5.0 );
    x.print();
    x.del();

    return 0;
}

void vector_t::init (const unsigned n ) {
    size = n;
    coeffs = new real_t[n];
}

void vector_t::del () {
    delete[] coeffs;
}

void vector_t::fill ( const real_t f ) {
    for (int i = 0; i < size; i++ )
        coeffs[i] = f;
}

void vector_t::scale ( const real_t f ) {
    for (int i = 0; i < size; i++ )
```

```

        coeffs[i] *= f;
    }

    void vector_t::print () {
        cout << "size = " << size << endl;
        cout << "coeffs:" << endl;
        for (int i = 0; i < size; i++ )
            cout << "        " << coeffs[i] << endl;
    }

```

## 8.2 bt\_struct\_constructor\_destructor.cpp

```
/* program to illustrate using struct with special functions for  
 * construction and destruction of objects  
 *  
 * Rajeev Singh  
 * 2013-04-05  
 *  
 */  
  
#include <iostream>  
using namespace std;  
typedef double real_t;  
  
struct vector_t {  
    size_t size;  
    real_t * coeffs;  
  
    vector_t ( const unsigned n );  
    ~vector_t ();  
    void fill ( const real_t f );  
    void scale ( const real_t f );  
    void print ();  
};  
  
int main() {  
    vector_t x(10);  
  
    x.print();  
    x.fill( 1.0 );  
    x.print();  
    x.scale( 5.0 );  
    x.print();  
  
    return 0;  
}  
  
vector_t::vector_t (const unsigned n ) {  
    size = n;  
    coeffs = new real_t[n];  
}  
  
vector_t::~~vector_t () {  
    delete[] coeffs;  
}  
  
void vector_t::fill ( const real_t f ) {  
    for (int i = 0; i < size; i++ )  
        coeffs[i] = f;  
}  
  
void vector_t::scale ( const real_t f ) {  
    for (int i = 0; i < size; i++ )  
        coeffs[i] *= f;  
}
```

```
}  
  
void vector_t::print () {  
    cout << "size = " << size << endl;  
    cout << "coeffs:" << endl;  
    for (int i = 0; i < size; i++ )  
        cout << "          " << coeffs[i] << endl;  
}
```

## 9 Day 10: Classes Continued

### 9.1 bu\_struct\_this\_pointer.cpp

```
/* program to illustrate the use of this pointer  
 * members  
 *  
 * Rajeev Singh  
 * 2013-04-07  
 *  
 */  
  
#include <iostream>  
using namespace std;  
typedef double real_t;  
  
struct vector_t {  
private:  
    size_t size;  
    real_t * coeffs;  
  
public:  
    vector_t ( const unsigned n );  
    ~vector_t ();  
    void fill ( const real_t f );  
    void scale ( const real_t f );  
    void add ( const real_t alpha, const vector_t & a );  
    void print () const;  
};  
  
int main() {  
    vector_t x(10), y(10);  
  
    x.fill( 1.0 );  
    y.fill( 2.0 );  
    cout << "x:" << endl;  
    x.print();  
    cout << "y:" << endl;  
    y.print();  
    x.add( 10.0, y );  
    cout << "x:" << endl;  
    x.print();  
    cout << "y:" << endl;  
    y.print();  
  
    return 0;  
}  
  
vector_t::vector_t (const unsigned n ) {  
    size = n;  
    coeffs = new real_t[n];  
}  
  
vector_t::~~vector_t () {
```

```

        delete[] coeffs;
    }

    void vector_t::fill ( const real_t f ) {
        for (int i = 0; i < size; i++ )
            coeffs[i] = f;
    }

    void vector_t::scale ( const real_t f ) {
        for (int i = 0; i < size; i++ )
            coeffs[i] *= f;
    }

    void vector_t::add ( const real_t alpha, const vector_t & a ) {
        for (int i = 0; i < this->size; i++ )
            this->coeffs[i] += alpha * a.coeffs[i];
    }

    void vector_t::print () const {
        cout << "size = " << size << endl;
        cout << "coeffs:" << endl;
        for (int i = 0; i < size; i++ )
            cout << "        " << coeffs[i] << endl;
    }

```



## 9.2 bv\_copy\_constructor.cpp

```
/* program to illustrate the use of copy constructor
 * members
 *
 * Rajeev Singh
 * 2013-04-07
 *
 */

#include <iostream>
using namespace std;
typedef double real_t;

struct vector_t {
private:
    size_t size;
    real_t * coeffs;

public:
    vector_t ( const unsigned n );
    vector_t ( const vector_t & a );
    ~vector_t ();
    void fill ( const real_t f );
    void scale ( const real_t f );
    void add ( const real_t alpha, const vector_t & a );
    void print () const;
};

int main() {
    vector_t x(10);

    x.fill( 1.0 );
    cout << "x:" << endl;
    x.print();

    vector_t y(x);
    cout << "y:" << endl;
    y.print();

    x.scale( 5.0 );
    cout << "x:" << endl;
    x.print();
    cout << "y:" << endl;
    y.print();

    return 0;
}

vector_t::vector_t (const unsigned n ) {
    size = n;
    coeffs = new real_t[n];
}
```

```

vector_t::vector_t (const vector_t & a ) {
    this->size = a.size;
    this->coeffs = new real_t[a.size];
    for (int i = 0; i < a.size; i++ )
        this->coeffs[i] = a.coeffs[i];
}

vector_t::~~vector_t () {
    delete[] coeffs;
}

void vector_t::fill ( const real_t f ) {
    for (int i = 0; i < size; i++ )
        coeffs[i] = f;
}

void vector_t::scale ( const real_t f ) {
    for (int i = 0; i < size; i++ )
        coeffs[i] *= f;
}

void vector_t::add ( const real_t alpha, const vector_t & a ) {
    for (int i = 0; i < this->size; i++ )
        this->coeffs[i] += alpha * a.coeffs[i];
}

void vector_t::print () const {
    cout << "size = " << size << endl;
    cout << "coeffs:" << endl;
    for (int i = 0; i < size; i++ )
        cout << "        " << coeffs[i] << endl;
}

```

### 9.3 bw\_default\_copy\_constructor.cpp

```
/* program to illustrate the problem with the default copy constructor
* members
*
* Rajeev Singh
* 2013-04-07
*
*/

#include <iostream>
using namespace std;
typedef double real_t;

struct vector_t {
private:
    size_t size;
    real_t * coeffs;

public:
    vector_t ( const unsigned n );
    //~vector_t ();
    void fill ( const real_t f );
    void scale ( const real_t f );
    void add ( const real_t alpha, const vector_t & a );
    void print () const;
};

int main() {
    vector_t x(10);

    x.fill( 1.0 );
    cout << "x:" << endl;
    x.print();

    vector_t y(x);
    cout << "y:" << endl;
    y.print();

    x.scale( 5.0 );
    cout << "x:" << endl;
    x.print();
    cout << "y:" << endl;
    y.print();

    return 0;
}

vector_t::vector_t (const unsigned n ) {
    size = n;
    coeffs = new real_t[n];
}

/*
```

```

vector_t::~~vector_t () {
    delete[] coeffs;
}
*/

void vector_t::fill ( const real_t f ) {
    for (int i = 0; i < size; i++ )
        coeffs[i] = f;
}

void vector_t::scale ( const real_t f ) {
    for (int i = 0; i < size; i++ )
        coeffs[i] *= f;
}

void vector_t::add ( const real_t alpha, const vector_t & a ) {
    for (int i = 0; i < this->size; i++ )
        this->coeffs[i] += alpha * a.coeffs[i];
}

void vector_t::print () const {
    cout << "size = " << size << endl;
    cout << "coeffs:" << endl;
    for (int i = 0; i < size; i++ )
        cout << "          " << coeffs[i] << endl;
}

```

## 9.4 bx\_struct\_with\_const\_functions.cpp

```
/* program to illustrate using struct with const functions
*/
* Rajeev Singh
* 2013-04-05
*/

#include <iostream>
using namespace std;
typedef double real_t;

struct vector_t {
    size_t size;
    real_t * coeffs;

    vector_t ( const unsigned n );
    ~vector_t ();
    void fill ( const real_t f );
    void scale ( const real_t f );
    void print () const;
};

int main() {
    const vector_t x(10);

    x.print();
    //x.fill( 1.0 );    // error
    //x.print();
    //x.scale( 5.0 );    // error
    //x.print();

    return 0;
}

vector_t::vector_t (const unsigned n ) {
    size = n;
    coeffs = new real_t[n];
}

vector_t::~~vector_t () {
    delete[] coeffs;
}

void vector_t::fill ( const real_t f ) {
    for (int i = 0; i < size; i++ )
        coeffs[i] = f;
}

void vector_t::scale ( const real_t f ) {
    for (int i = 0; i < size; i++ )
        coeffs[i] *= f;
}
```

```
void vector_t::print () const {  
    cout << "size = " << size << endl;  
    cout << "coeffs:" << endl;  
    for (int i = 0; i < size; i++ )  
        cout << "          " << coeffs[i] << endl;  
}
```

## 9.5 by\_struct\_visibility.cpp

```
/* program to illustrate using struct with different visibility for
* members
*
* Rajeev Singh
* 2013-04-05
*
*/

#include <iostream>
using namespace std;
typedef double real_t;

struct vector_t {
private:
    size_t size;
    real_t * coeffs;

public:
    vector_t ( const unsigned n );
    ~vector_t ();
    void fill ( const real_t f );
    void scale ( const real_t f );
    void print () const;
};

int main() {
    vector_t x(10);

    x.print();
    x.fill( 1.0 );
    x.print();
    x.scale( 5.0 );
    x.print();

    //cout << x.size << endl; // error

    return 0;
}

vector_t::vector_t (const unsigned n ) {
    size = n;
    coeffs = new real_t[n];
}

vector_t::~~vector_t () {
    delete[] coeffs;
}

void vector_t::fill ( const real_t f ) {
    for (int i = 0; i < size; i++ )
        coeffs[i] = f;
}
```

```

void vector_t::scale ( const real_t f ) {
    for (int i = 0; i < size; i++ )
        coeffs[i] *= f;
}

void vector_t::print () const {
    cout << "size = " << size << endl;
    cout << "coeffs:" << endl;
    for (int i = 0; i < size; i++ )
        cout << "          " << coeffs[i] << endl;
}

```



## 9.6 bz\_safely\_changing\_size\_of\_array.cpp

```
/* program to illustrate using visibility to change the size of array
 * safely
 *
 * Rajeev Singh
 * 2013-04-07
 *
 */

#include <iostream>
using namespace std;
typedef double real_t;

struct vector_t {
private:
    size_t size;
    real_t * coeffs;

public:
    vector_t ( const unsigned n );
    ~vector_t ();
    int get_size ();
    void set_size ( const unsigned n );
    void fill ( const real_t f );
    void scale ( const real_t f );
    void print () const;
};

int main() {
    vector_t x(10);

    x.print();
    x.fill( 1.0 );
    x.print();
    x.scale( 5.0 );
    x.print();
    cout << "Size of x = " << x.get_size() << endl;
    x.set_size( 4 );
    x.print();

    return 0;
}

vector_t::vector_t (const unsigned n ) {
    size = n;
    coeffs = new real_t[n];
}

vector_t::~~vector_t () {
    delete[] coeffs;
}

int vector_t::get_size () {
```

```

        return size;
    }

    void vector_t::set_size ( const unsigned n ) {
        if (size != n) {
            size = n;
            delete coeffs; // delete the old data
            coeffs = new real_t[n];
        }
    }

    void vector_t::fill ( const real_t f ) {
        for (int i = 0; i < size; i++ )
            coeffs[i] = f;
    }

    void vector_t::scale ( const real_t f ) {
        for (int i = 0; i < size; i++ )
            coeffs[i] *= f;
    }

    void vector_t::print () const {
        cout << "size = " << size << endl;
        cout << "coeffs:" << endl;
        for (int i = 0; i < size; i++ )
            cout << "        " << coeffs[i] << endl;
    }

```

## 10 Day 11: Classes Continued, Templates

### 10.1 ca\_operator\_overloading.cpp

```
/* program to illustrate operator overloading
 * safely
 *
 * Rajeev Singh
 * 2013-04-08
 *
 */

#include <iostream>
using namespace std;
typedef double real_t;

struct vector_t {
private:
    size_t size;
    real_t * coeffs;

public:
    vector_t ( const unsigned n );
    ~vector_t ();
    int get_size ();
    void set_size ( const unsigned n );
    void fill ( const real_t f );
    void scale ( const real_t f );
    void print () const;

    vector_t & operator += (const vector_t & a) {
        for (size_t i = 0; i < size; i++ )
            coeffs[i] += a.coeffs[i];
        return *this;
    }

    vector_t & operator -= (const vector_t & a) {
        for (size_t i = 0; i < size; i++ )
            coeffs[i] -= a.coeffs[i];
        return *this;
    }
};

int main() {
    vector_t x(4), y(4);

    x.fill( 3.0 );
    y.fill( 1.0 );
    x.print();
    y.print();

    x += y;
    x.print();
    y.print();
}
```

```

        return 0;
    }

    vector_t::vector_t (const unsigned n ) {
        size = n;
        coeffs = new real_t[n];
    }

    vector_t::~~vector_t () {
        delete[] coeffs;
    }

    int  vector_t::get_size () {
        return size;
    }

    void vector_t::set_size ( const unsigned n ) {
        if (size != n) {
            size = n;
            delete coeffs;  // delete the old data
            coeffs = new real_t[n];
        }
    }

    void vector_t::fill ( const real_t f ) {
        for (int i = 0; i < size; i++ )
            coeffs[i] = f;
    }

    void vector_t::scale ( const real_t f ) {
        for (int i = 0; i < size; i++ )
            coeffs[i] *= f;
    }

    void vector_t::print () const {
        cout << "size = " << size << endl;
        cout << "coeffs:" << endl;
        for (int i = 0; i < size; i++ )
            cout << "        " << coeffs[i] << endl;
    }

```

## 10.2 cb\_templates.cpp

```
/* program to illustrate generic programming using templates  
 * safely  
 *  
 * Rajeev Singh  
 * 2013-04-08  
 *  
 */  
  
#include <iostream>  
using namespace std;  
  
template <typename T>  
T square(const T f) {  
    return f*f;  
}  
  
int main() {  
    double x = square( 2.1 );  
    int     m = square( 2 );  
  
    cout << "x = : " << x << endl;  
    cout << "m = : " << m << endl;  
  
    return 0;  
}
```

GO TO THE TALK TO DISCUSS BLAS AND TEMPLATES