IIT Bombay

Computer Programming

Dr. Deepak B Phatak
Dr. Supratik Chakraborty
Department of Computer Science and Engineering
IIT Bombay

Session: Introduction to Functions in Programming

Quick Recap of Relevant Topics

IIT Bombay

- Various constructs to help us write useful programs
  - Assignment statements
  - Input/output statements
  - Expressions
  - Sequential and conditional statements
  - Iteration/looping constructs

All encapsulated within "main( )"

Overview of This Lecture

- Break away from the monopoly of "main()"
  - Have other sub-units of a program that can compute
  - Reduce the burden of programming everything in "main()"
- Functions
  - Simple uses in programs
  - A contract-centric view of programming

An Encoding/Decoding Example

IIT Bombay

- We want to store quiz 1 and quiz 2 marks of CS101 students in an encoded form

  So that others cannot figure out the actual marks

- Encoding strategy:

  The ordered pair of marks (m, n) is encoded as $2^m \times 3^n$

- Assume all marks are integers in {1, 2, … 10}

C++ Program

IIT Bombay

```
int main()  {
```

```
for (i = 0, twoRaisedQ1 = 1; i < q1Marks; i++, twoRaisedQ1 *= 2) { };
for (j = 0, threeRaisedQ2 = 1; j < q2Marks; j++, threeRaisedQ2 *= 3) { };
cipher = twoRaisedQ1 * threeRaisedQ2;
```

```
    cin >> q1Ma      Marks;  // Read quiz1 and quiz2 marks
    // Compute cipher from q1Marks and q2Marks
    // Store count and cipher in appropriate file
  }
  return 0;
}
```

C++ Program Fragment

IIT Bombay

```
          ⋮
for (count = 1; count <= numStudents; count++)  {
   cout << “Give quiz1 and quiz2 marks of student “ << count << “: “;
   cin >> q1Marks >> q2Marks;  // Read quiz1 and quiz2 marks
  for (i = 0  twoRaisedQ1 = 1; i < q1Marks; i++, twoRaisedQ1 *= 2) { };
  for (j = 0  threeRaisedQ2 = 1; j < q2Marks; j++, threeRaisedQ2 *= 3) { };
  cipher =  woRaisedQ1 * threeRaisedQ2;
 // Store c   nt and cipher in appropriate file
}
```

Mix of longish code fragments with different purpose:
Hurts readability/understandability of code

C++ Program Fragment

```
        ⋮
for (count = 1; count <= numStudents; count++)  {
   cout << "Give quiz1 and quiz2 marks of student " << count << ": ";
   cin >> q1Marks >> q2Marks;  // Read quiz1 and quiz2 marks
  for (i = 0, twoRaisedQ1 = 1; i < q1Marks; i++, twoRaisedQ1 *= 2) { };
  for (j = 0, threeRaisedQ2 = 1; j < q2Marks; j++, threeRaisedQ2 *= 3) { };
  cipher = twoRaisedQ1 * threeRaisedQ2;
 // Store count and cipher in appropriate file
}
```

Repeated code: Bad, bad, bad !!!

Recipe for introducing errors in some copy of code

Can We Do Better?

IIT Bombay

```
        •
        •
        •
for (count = 1; count <= numStudents; count++)  {
    cout << "Give quiz1 and quiz2 marks of student " << count << ": ";
    cin >> q1Marks >> q2Marks;  // Read quiz1 and quiz2 marks
  for (i = 0, twoRaisedQ1 = 1; i < q1Marks; i++, twoRaisedQ1 *= 2) { };
  for (j = 0, threeRaisedQ2 = 1; j < q2Marks; j++, threeRaisedQ2 *= 3) { };
  cipher = twoRaisedQ1 * threeRaisedQ2;
  // Store count and cipher in appropriate file
}
```

Can we encapsulate this as another computational sub-task?
Takes q1Marks and q2Marks as input and gives us cipher

Function in A C++ Program

IIT Bombay

```
        •
        •
        •
for (count = 1; count <= numStudents; count++)  {
    cout << “Give quiz1 and quiz2 marks of student “ << count << “: “;
    cin >> q1Marks >> q2Marks;  // Read quiz1 and quiz2 marks
    cipher = myEncode(q1Marks, q2Marks);
   // Store c      and cipher in approp   ate file
```

Name of computational subtask
or **function**
(recall naming conventions)
We **invoked/called** “myEncode”

Inputs to function
(similar to inputs provided
to “main” by user)

Functions in A C++ Program

IIT Bombay

```
        ⋮
for (count = 1; count <= numStudents; count++)  {
    cout << "Give quiz1 and quiz2 marks of student " << count << ": ";
    cin >> q1Marks >> q2Marks;  // Read quiz1 and quiz2 marks
    cipher = myEncode(q1Marks, q2Marks);
   // Store count and cipher in appropriate file
  }
        ⋮
```

Evaluates to a value and has a type (int in this case)

A C++ Program with Function

IIT Bombay

```
#include <iostream>
using namespace std;
int myEncode(int q1Marks, int q2Marks);
int main() {
    …
    for ( … ) {  …
       cipher = myEncode(q1Marks, q2Marks);
       …
    }
   …
}
```

Need to specify somewhere

- **"myEncode" is a function,**
- **it takes integer inputs, and**
- **it computes integer value**

Used by compiler to enforce correct usage of "myEncode" and also to allocate space (we'll soon see)

A C++ Program with Function

IIT Bombay

```
#
u
i
i
    …
    for ( … ) {  …
      cipher = myEncode(q1Marks, q2Marks);

    }
    ..
}
```

Also need to specify somewhere instructions for "myEncode" to compute cipher from its two ordered inputs

```
int myEncode(int q1Marks,
                      int q2Marks)
{



}
```

```
for (i = 0, twoRaisedQ1 = 1; i < q1Marks; i++, twoRaisedQ1 *= 2) { };
for (j = 0, threeRaisedQ2 = 1; j < q2Marks; j++, threeRaisedQ2 *= 3) { };
cipher = twoRaisedQ1 * threeRaisedQ2;
```

A C++ Program with Function

IIT Bombay

Variables being used in "myEncode"
Are they the same as in "main"?

NO in general, unless we require "main" and "myEncode" to share variables

```
int myEncode(int q1Marks,
                    int q2Marks)
{


}
```

```
  …
  for (
    cipl      myEncode(q1Marks, q2Marks);

  }
  ..
}
```

```
for (i = 0, twoRaisedQ1 = 1; i < q1Marks; i++, twoRaisedQ1 *= 2) { };
for (j = 0, threeRaisedQ2 = 1; j < q2Marks; j++, threeRaisedQ2 *= 3) { };
cipher = twoRaisedQ1 * threeRaisedQ2;
```

A C++ Program with Function

IIT Bombay

```
#
u
i
i
    …
    for ( … ) {  …
      cipher = myEncode(q1Marks, q2Marks);

    }
    …
}
```

Local variables of "myEncode"

No confusion/mixing up with variables declared in "main"

```
int myEncode(int q1Marks,
                          int q2Marks)
{ int i, j, twoRaisedQ1;
  int twoRaisedQ1, cipher;

}
```

```
for (i = 0, twoRaisedQ1 = 1; i < q1Marks; i++, twoRaisedQ1 *= 2) { };
for (j = 0, threeRaisedQ2 = 1; j < q2Marks; j++, threeRaisedQ2 *= 3) { };
cipher = twoRaisedQ1 * threeRaisedQ2;
```

A C++ Program with Function

IIT Bombay

Formal Parameters of "myEncode"

View them like **local variables** of "myEncode"

```
    ...
    for ( ... ) {  ...
      cipher = myEncode(q1Marks, q2Marks);

    }
    ..
}
```

```
ncode(int q1Marks,
      int q2Marks)
{ int i, j, twoRaisedQ1;
  int twoRaisedQ1, cipher;

}
```

```
for (i = 0, twoRaisedQ1 = 1; i < q1Marks; i++, twoRaisedQ1 *= 2) { };
for (j = 0, threeRaisedQ2 = 1; j < q2Marks; j++, threeRaisedQ2 *= 3) { };
cipher = twoRaisedQ1 * threeRaisedQ2;
```

A C++ Program with Function

IIT Bombay

How will the value of cipher computed by "myEncode" be passed (returned) to "main"?

```
    …
    for (
      cip          myEncode(q1Marks, q2Marks);

    }
    ..
}
```

```
int myEncode(int q1Marks,
                  int q2Marks)
{ int i, j, twoRaisedQ1;
  int twoRaisedQ1, cipher;

  return cipher;
}
```

```
for    = 0, twoRaisedQ1 = 1; i < q1Marks; i++, twoRaisedQ1 *= 2) { };
for  j = 0, threeRaisedQ2 = 1; j < q2Marks; j++, threeRaisedQ2 *= 3) { };
cipher = twoRaisedQ1 * threeRaisedQ2;
```

A C++ Program with Function

```
#include <iostream>
using namespace std;
int myEncode(int q1Marks, int q2Marks);
int main() {
   }
   ..
}
```

```
int myEncode(int q1Marks,
                     int q2Marks)
{ int i, j, twoRaisedQ1;
  int twoRaisedQ1, cipher;

  return cipher;
}
```

Type must match declared type of what this function evaluates to

```
for (i = 0, twoRaisedQ1 = 1; i < q1Marks; i++, twoRaisedQ1 *= 2) { };
for (j = 0, threeRaisedQ2 = 1; j < q2Marks; j++, threeRaisedQ2 *= 3) { };
cipher = twoRaisedQ1 * threeRaisedQ2;
```

Contract View of Functions

```
#include <iostream>
using namespace std;
int myEncode(int q1Marks, int q2Marks);
int main() {
   …
   for ( … ) {  …
     cipher = myEncode(q1Marks, q2Marks);
     …
   }
   …
}
```

Ensure pre-condition of “myEncode” before invoking

Guaranteed post-condition of “myEncode” on returning

```
// PRECONDITION:
// 1 <= q1Marks <= 10
// 1 <= q2Marks <= 10
int myEncode(int q1Marks,
                  int q2Marks)
{
     BLACK BOX
}
// POSTCONDITION:
//   Returned value =
//   2 q1Marks x 3 q2Marks
// No side effects (later lecture)
```

$2^{q1Marks} \times 3^{q2Marks}$

Function Within A Function?

IIT Bombay

// PRECONDITION: 1 <= q1Marks <= 10, 1 <= q2Marks <= 10

```
int myEncode(int q1Marks, q2Marks) {
  int i, j, twoRaisedQ1, threeRaisedQ2, cipher;
  for (i = 0, twoRaisedQ1 = 1; i < q1Marks; i++, twoRaisedQ1 *= 2) { };
   for (j = 0, threeRaisedQ2 = 1; j < q2Marks; j++, threeRaisedQ2 *= 3) { };
   cipher = twoRaisedQ1 *
   return cipher;
}
```

Repeated code !!!
Why not use another function
int power(int base, int exponent) ?

// POSTCONDITION: Returned value = $2^{q1Marks} \times 3^{q2Marks}$ and no side effects

Function Within A Function: Why Not?

```
// PRECONDITION:  1 <= q1Marks <= 10, 1 <= q2Marks <= 10
int myEncode(int q1Marks, q2Marks) {
  int i, j, twoRaisedQ1, threeRaisedQ2, cipher;
  twoRaisedQ1 = power(2, q1Marks);
  threeRaisedQ2 = power(3, q2Marks);
  cipher = twoRaisedQ1 * threeRaisedQ2;
  return cipher;
 }
```

// POSTCONDITION: Returned value = $2^{q1Marks} \times 3^{q2Marks}$ and no side effects

Another C++ Function

IIT Bombay

```
// PRECONDITION: integer base > 0, integer exponent >= 0
int power(int base, int exponent)
{ int i, result;
  for (i = 0, result = 1; i < exponent; i++, result *= base) { };
  return result;
}
```

// POSTCONDITION: result = base ^{exponent}, no side effects

Overall Program Structure

```
#include <iostream>
using namespace std;
int myEncode(int q1Marks,int q2Marks);
int power(int base, int exponent);
int main() { …
  for ( … ) {  …
   cipher = myEncode(q1Marks, q2Marks);
  …}
…}
```

```
// PRECONDITION:  …
int myEncode(int q1Marks,
                         int q2Marks)
{ …
  twoRaisedQ1 = power(2, q1Marks);
  threeRaisedQ2 = power(3, q2Marks);
 … }
// POSTCONDITION:  …
```

```
// PRECONDITION:  …
int power(int base, int exponent)
{ … }
// POSTCONDITION:  …
```

Summary

IIT Bombay

- Simple use of functions in programming

  Enables modular programming, separation of concerns

- Contract view of functions
  - Pre-conditions
  - Post-conditions