IIT Bombay

Computer Programming

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

Session: Flow of Control in Function Call

Quick Recap of Relevant Topics

IIT Bombay

- Use of simple functions in programs
  - Encapsulating computational sub-tasks as functions
  - Invoking functions from other functions
  - Functions returning values of specified types
  - Modular development of programs
- Contract-centric view of programming with functions

Overview of This Lecture

IIT Bombay

- Flow of control in a function call and return
- Activation records and call stack

Recall: Encoding 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}

Recall: C++ Program Structure

IIT Bombay

```
#include <iostream>
using namespace std;
int myEncode
int power(int
int main() { …
  for ( … ) { …
   cip
…
}
```

1 <= q1Marks <= 10

1 <= q2Marks <= 10

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

can be represented

as int (4 bytes)

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

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

Recall: C++ Program Structure

IIT Bombay

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

return value =
$2^{q1Marks} \times 3^{q2Marks}$

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

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

Recall: C++ Program Structure

IIT Bombay

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

base $^{\text{exponent}}$ can be represented as int (4 bytes)

base > 0, exponent >= 0,
1 <= base $^{\text{exponent}}$ <= $2^{31} - 1$

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

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

Recall: C++ Program Structure

IIT Bombay

```
#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:  …
```

return value = base $^{\text{exponent}}$

Flow of Control: An Animation

IIT Bombay

```
#include <iostream>
using namespace std;
int myEncode(int q1Marks,int q2Marks);
int power(int base, int exponent);
int main() { …
  for ( … ) { …
   cipher = myEncode(q1Marks, q2Marks);
  … }
  …
return 0; }
```

```
int myEncode(int q1Marks,
                    int q2Marks)
{ …
  twoRaisedQ1 = power(2, q1Marks);
  threeRaisedQ2 = power(3, q2Marks);
  …
  return cipher;
}
```

```
int power(int base, int exponent)
{ …
  return result;
}
```

Flow of Control: A Closer Look

IIT Bombay

Operating System (OS) calls **main**

main calls **myEncode(q1Marks, q2Marks)**

  myEncode calls **power(2, q1Marks)**

    power returns to **myEncode**, where **power(2, q1Marks)** called

  myEncode calls **power(3, q2Marks)**

    power returns to **myEncode**, where **power(3, q2Marks)** called

  myEncode returns to **main**, where **myEncode(q1Marks, q2Marks)** called

main returns to **OS**

myEncode @ power(2

OS @ main

LAST IN FIRST OUT ACCESS

Call Stack

IIT Bombay

- We need to store "information" about function calls in a way that allows last-in-first-out (LIFO) access

  A stack (think, stack of papers) does exactly that

- **Call stack** used to store "information" about function calls

  Resides in a special, reserved part of main memory

- **What "information" must be stored in the call stack?**

Recall Flow of Control

IIT Bombay

Must remember:

- **Where to return in calling function**
- **Values of local variables in calling function at time of function call**

```
#
u
i
i
i
  for ( … ) {  …
   ipher = myEncode(q1Marks, q2Marks);
  …}
…
return 0; }
```

```
yEncode(int q1Marks,
                  int q2Marks)

aisedQ1 = power(2, q1Marks);
RaisedQ2 = power(3, q2Marks);

return cipher;
}
```

```
int power(int base, int exponent)
{ …
  return result;
}
```

Memory For An Executing Program (Process)

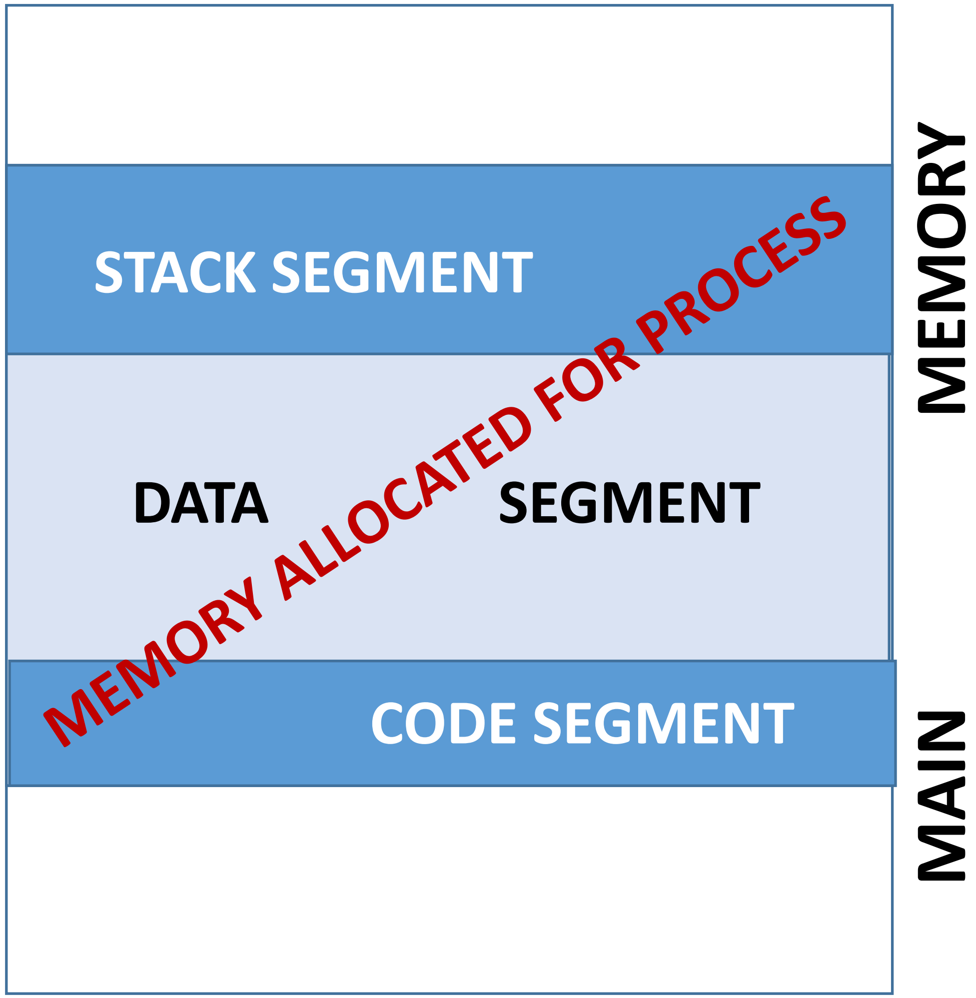
IIT Bombay

- Operating system allocates a part of main memory for use by a process
- Divided into:

  Code segment: Stores executable instructions in program

  Data segment: For dynamically allocated data (later lecture)

  Stack segment: Call stack



Where To Return From Called Function?

IIT Bombay

- Program stored in code segment of main memory
- Every (machine language) instruction has a memory address
- Program counter (PC)

  Special CPU register that holds memory address of current instruction being executed
- When **myEncode** is called from **main**, value of PC must be saved.

  On returning from **myEncode**, execution should resume from instruction at this address.

Activation Frame/Record

IIT Bombay

Entry in call stack for each function called

- E.g., **main** **(caller)** calling **myEncode** **(callee)**

Activation record contains

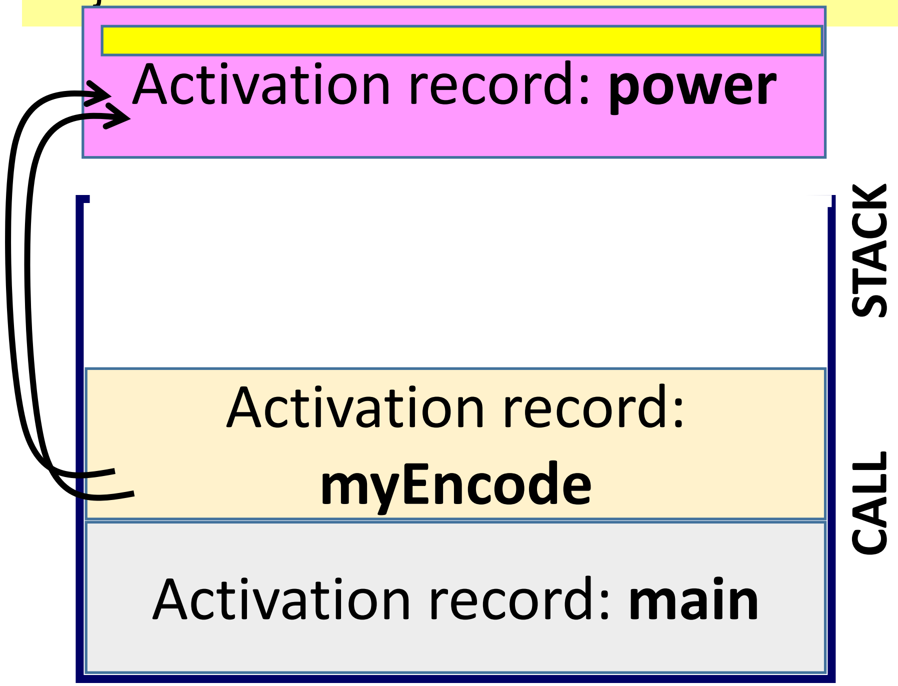
- Memory for all local variables of callee (**myEncode**)
- PC value in caller when callee was called (address of instruction in **main** that calls **myEncode**)
- Space for return value of callee
- Additional book-keeping information (let’s not worry …)

Activation Records in Call Stack

When a function (caller) calls a function (callee)

- a **fresh** activation record for callee created
- Values of function parameters from caller copied to space allocated for formal parameters of callee
- PC of caller saved
- Other book-keeping information updated
- Activation record for callee pushed on call stack

```
int
myEncode(int q1Marks, int q2Marks)
{ ….
  twoRaisedQ1 = power(2, q1Marks);
 …}
```



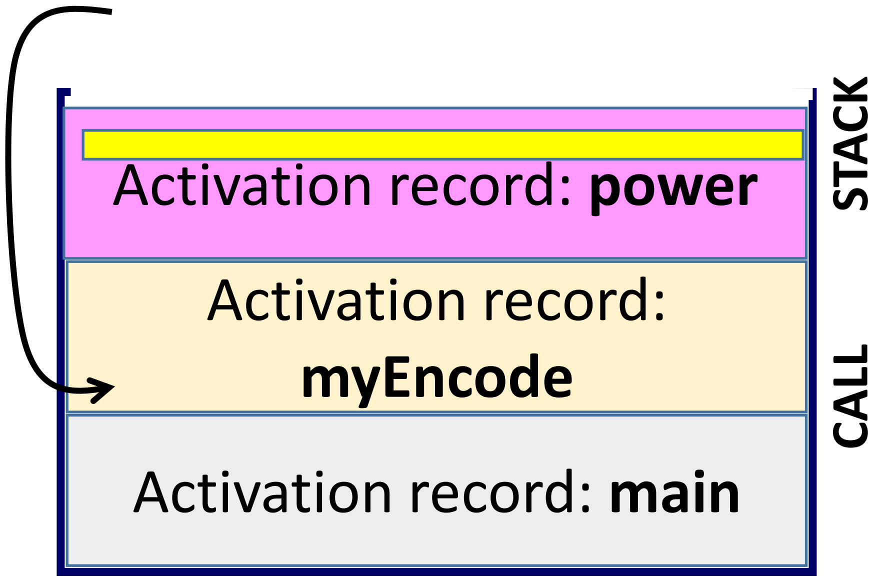
Activation Records in Call Stack

IIT Bombay

When a function (callee) returns

- Callee's activation record popped from call stack
- Return value from popped activation record copied to activation record of caller (now on top of stack)
- Value of PC saved in popped activation record loaded in PC of CPU
- Free activation record of callee
- Resume execution of instruction at location given by updated PC

```
int power(int base, int exponent)
{ ….
  return result;
 …}
```



Summary

IIT Bombay

- Flow of control in function call and return
- Memory layout of a process
- Call stack and activation records