SM2-21st

C Programming Language

Chapter 8: Recursion

A/Prof Tay Seng Chuan

Warden, Prince George’s Park Residences

Office of the Provost

Ground Rules

- Switch off your handphone and pager
- Switch off your laptop computer and keep it
- No talking while lecture is going on
- No gossiping while the lecture is going on
- Raise your hand if you have question to ask
- Be on time for lecture
- Be on time to come back from the recess break to continue the lecture
- Bring your lecturenotes to lecture

Iteration versus Recursion

- Most of the time, you can express a problem more elegantly using recursion.
- E.g. summation of numbers from 1 to n (in iterative form)

```
sum(n)  =   1 + 2 + … + (n-1) + n
```

$$= \sum_{i=1}^{n} i$$

```
        =   for (i=1;i<=n;i++)
               sum = sum+i;
            return sum;
```

In Recursion Form

- Summation of numbers from 1 to n using *recursion*.

```
sum(n)  =   1 + 2 + 3 +(n-1)+ n
```

$$= \begin{cases} \texttt{1} & \texttt{if (n==1)} \\ \texttt{sum(n-1) + n} & \texttt{if (n>1)} \end{cases}$$

```
        =   if (n==1) return 1;
           else return sum(n-1) + n;
```

Recursion - basic idea

- In top-down design, you break up a problem into simpler sub-problems.
- In recursion, one or more of these sub-problems are simpler instances of the original problem.
- In practice, these algorithms can be implemented by methods calling themselves.

Another Example of Recursion

- Product of numbers from 1 to n using recursion.

```
factorial(n) = n*(n-1)*(n-2)*...*2*1
                       ┌ 1                  if (n==1)
                     = {
                       └n*factorial(n-1)    if  (n>1)

                     =   if (n==1) return 1;
                         else return n*factorial(n-1);
```

Visualizing execution of a program containing recursion

- With non-recursive programs, it is natural to visualize execution by imagining control stepping through the source code.
- This can be confusing for programs containing recursion.
- Instead, it is useful to imagine each call of a method generating a copy of the method, so that if the same method is called several times, several copies are present.

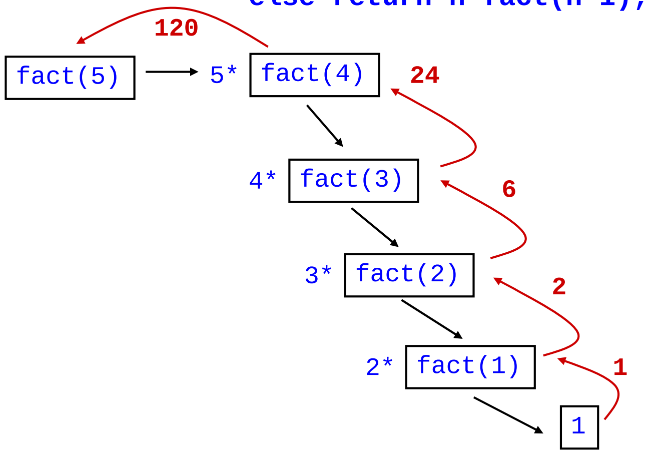
Scope

- When the method is called
  - caller is suspended,
  - “state” of caller saved in stack (LIFO – Last in first out),
  - new space allocated for variables of new method.
- With recursive call, same things happen.

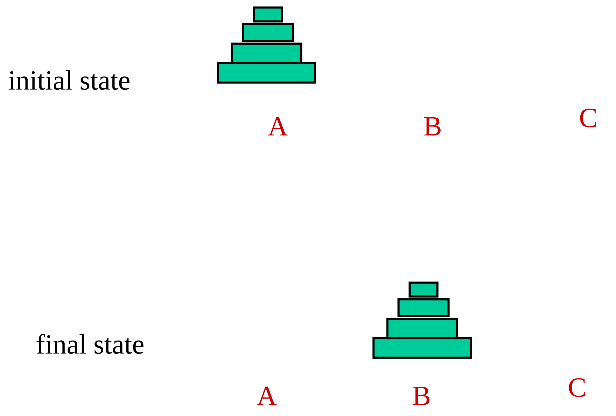
How Recursion Works?

- Given.

```
fact(n) =   if (n==1) return 1;
            else return n*fact(n-1);
```



Tower of Hanoi

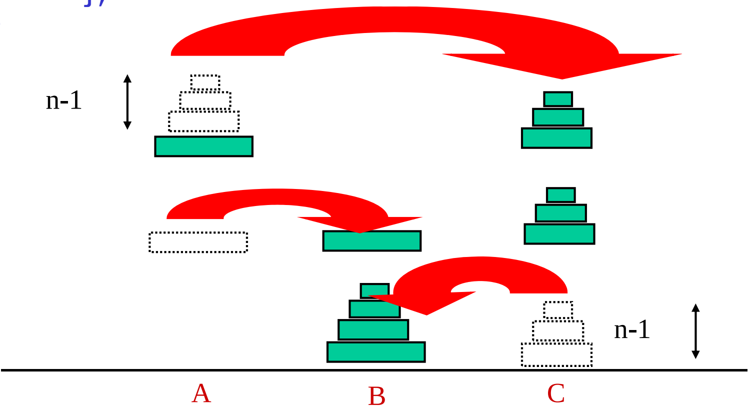


Tower of Hanoi (move n disc from A to B)

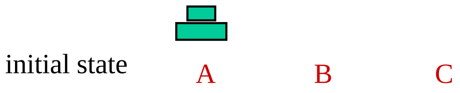
```
void tower (int n,char A,char B,char c)
{
  if (n==1) move(A,B); (take the disc from A to B)
  else {
          tower(n-1,A,C,B); (move n-1 disc from A to C)

          move(A,B); (take the disc from A to B)

          tower(n-1,C,B,A); (move n-1 disc from C to B)
      };
}
```

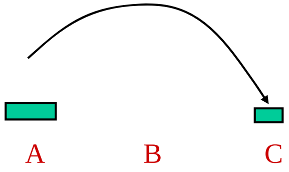


If we were to move 2 discs from A to B

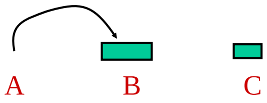


-We will need 3 steps:

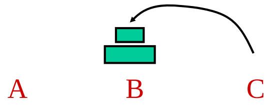
Move one disc on top from A → C



Move one disc on top from A → B



- Move one disc on top from C → B



T (4, A, B, C)

T (3, A, C, B) → T (3, A, C, B)

T (2, A, B, C) → T (2, A, B, C)

A→ C

A→B

C→B

A→ C

T (2, B, C, A) → T (2, B, C, A)

B→ A

B→C

A→C

A →B

T (3, C, B, A) → T (3, C, B, A)

T (2, C, A, B) → T (2, C, A, B)

C→ B

C→A

B→A

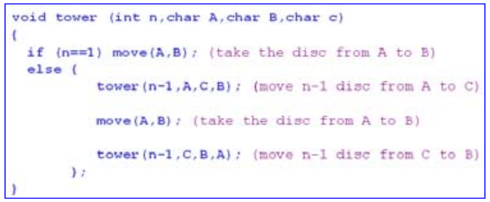
C→B

T (2, A, B, C) → T (2, A, B, C)

A→ C

A→B

C→B



```
void tower (int n,char A,char B,char c)
{
  if (n==1) move(A,B); (take the disc from A to B)
  else {
          tower(n-1,A,C,B); (move n-1 disc from A to C)

          move(A,B); (take the disc from A to B)

          tower(n-1,C,B,A); (move n-1 disc from C to B)
      };
}
```

A→ C

A→B

C→B

A→C

B→ A

B→C

A→C

A→B

C→ B

C→A

B→A

C→B

A→ C

A→B

C→B

Recursion - how to

Ask the following

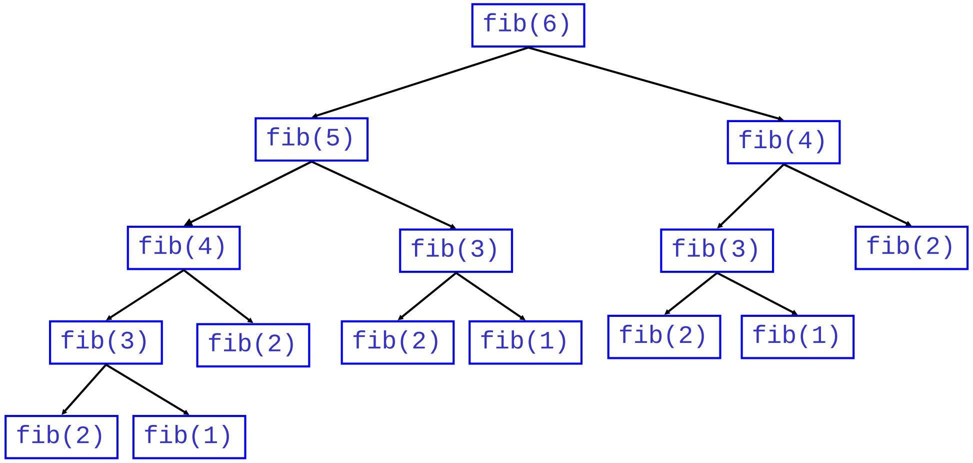
- How can you solve the problem using the solution of a “simpler” instance of the problem?
- Can you be sure to have a “simplest” input? (If so, include separate treatment of this case.)
- Can you be sure to reach the “simplest” input?

Fibonacci numbers

- Fibonacci series is a sequence where the first two numbers are 1, and a number in the sequence is the sum of the previous two numbers, i.e., 1, 1, 2, 3, 5, 8,…
- Naïve method for calculating the *n*th Fibonacci number recursively:

```
int fib(int n)
{
   if (n <= 2)
      return 1;
   else
      return fib(n-1)+fib(n-2);
}
```

Tracing Fibonacci Calls



fib(6)
fib(5)
fib(4)
fib(4)
fib(3)
fib(3)
fib(2)
fib(3)
fib(2)
fib(2)
fib(1)
fib(2)
fib(1)
fib(2)
fib(1)

Verify the Output of function `f`:

```
int f (int x)
{
  if (x>9) return 9;
  else if (x>5) return 5;
       else return 3+f(x+1);
}
```

```
f(1) = 20
f(2) = 17
f(3) = 14
f(7) = ?
f(199) = ?
```

Complete the following function in iterative form to produce the integral value of `this1` in reverse magnitude, i.e. `reverse (1234)` will return the integral value `4321`.

```
int reverse (int this1)
{
    :
    :
  return ..;
}
```

Answer:

```
int reverse (int this1)
{
  int sum=0, remainder;

  while (this1 !=0)
  {
    remainder = this1%10;
    sum = sum*10 + remainder;
    this1 = this1/10;
  }

  return sum;
}
```

Now write the function in recursive form!

```
int reverse (int this1)
{
  int sum=0, remainder;

  while (this1 !=0)
  {
    remainder = this1%10;
    sum = sum*10 + remainder;
    this1 = this1/10;
  }

  return sum;
}
```

`reverse (1234) = 4321.`

Answer:

```
int recur (int this1)
{
  static int sum=0;
  int remainder;

  if (this1==0) return 0;
  else
  {
    remainder = this1%10;
    sum = sum*10 + remainder;
    recur(this1/10);
  }

  return sum;
}
```

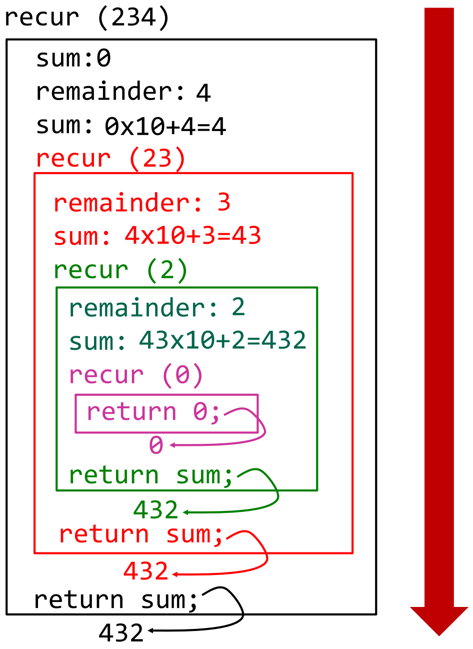
`recur (1234) = 4321.`

```
int recur (int this1)
{
  static int sum=0;
  int remainder;

  if (this1==0) return 0;
  else
  {
    remainder = this1%10;
    sum = sum*10 + remainder;
    recur(this1/10);
  }

  return sum;
}
```

`recur (234) = 432.`



```
recur (234)
  sum:0
  remainder: 4
  sum: 0x10+4=4
  recur (23)
    remainder: 3
    sum: 4x10+3=43
    recur (2)
      remainder: 2
      sum: 43x10+2=432
      recur (0)
        return 0;
        0
      return sum;
      432
    return sum;
    432
  return sum;
  432
```

```
int recur (int this1)
{
  static int sum=0;
  int remainder;

  if (this1==0) return 0;
  else
  {
    remainder = this1%10;
    sum = sum*10 + remainder;
    recur(this1/10);
  }

  return sum;
}
```

`recur (1234) = 4321.`

```
int recur2 (int this1)
{
  static int sum=0;
  int remainder;

  if (this1==0) return 0;
  else
  {
    recur2(this1/10);
    remainder = this1%10;
    sum = sum*10 + remainder;
  }

  return sum;
}
```

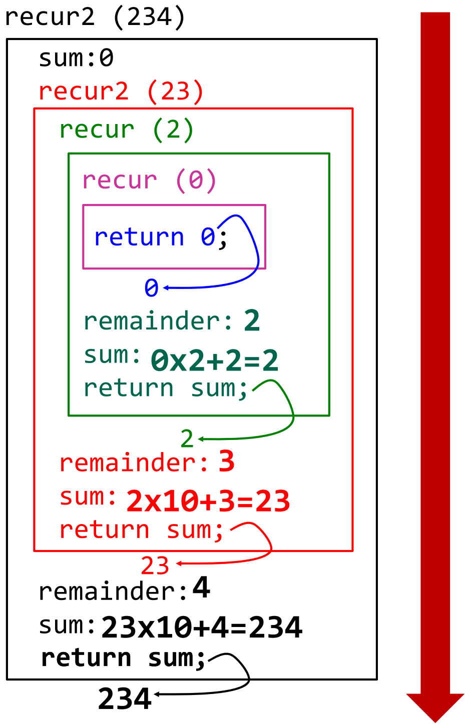
`recur2 (1234) = 1234.`

```
int recur2 (int this1)
{
  static int sum=0;
  int remainder;

  if (this1==0) return 0;
  else
  {
    recur2(this1/10);
    remainder = this1%10;
    sum = sum*10 + remainder;
  }

  return sum;
}
```

recur2 (1234) = 1234.



recur2 (234)

sum:0

recur2 (23)

recur (2)

recur (0)

return 0;

0

remainder: 2

sum: 0x2+2=2

return sum;

2

remainder: 3

sum: 2x10+3=23

return sum;

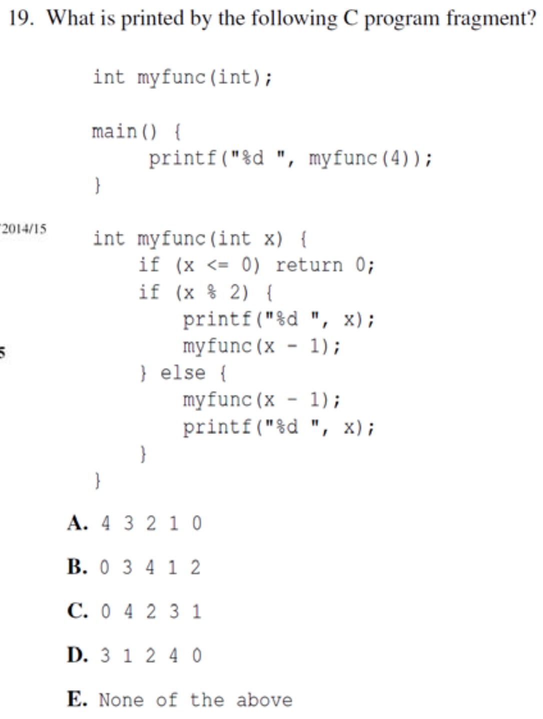
23

remainder:4

sum:23x10+4=234

return sum;

234



19. What is printed by the following C program fragment?

```
int myfunc(int);

main() {
     printf("%d ", myfunc(4));
}

int myfunc(int x) {
    if (x <= 0) return 0;
    if (x % 2) {
        printf("%d ", x);
        myfunc(x - 1);
    } else {
        myfunc(x - 1);
        printf("%d ", x);
    }
}
```

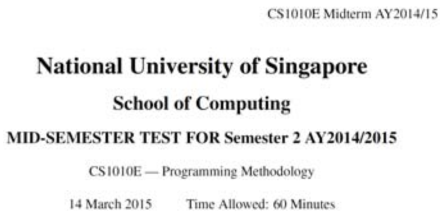
A. 4 3 2 1 0

B. 0 3 4 1 2

C. 0 4 2 3 1

D. 3 1 2 4 0

E. None of the above



CS1010E Midterm AY2014/15

National University of Singapore

School of Computing

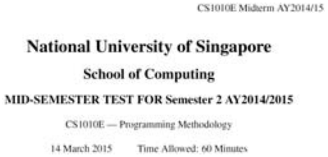
MID-SEMESTER TEST FOR Semester 2 AY2014/2015

CS1010E — Programming Methodology

14 March 2015 Time Allowed: 60 Minutes

SM2-21st

A/Prof Tay's Explanations to



CS1010E Midterm AY2014/15

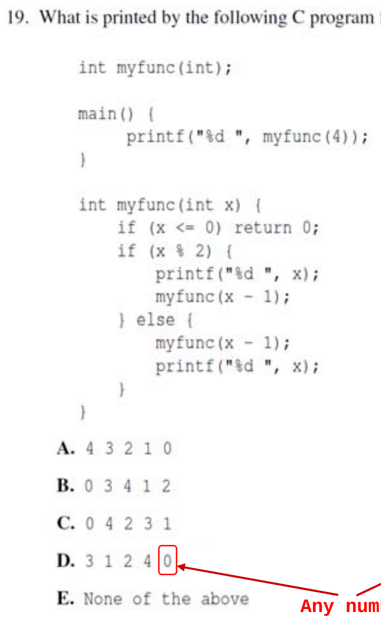
National University of Singapore

School of Computing

MID-SEMESTER TEST FOR Semester 2 AY2014/2015

CS1010E — Programming Methodology

14 March 2015 Time Allowed: 60 Minutes



19. What is printed by the following C program fragment?

```
int myfunc(int);

main() {
     printf("%d ", myfunc(4));
}

int myfunc(int x) {
    if (x <= 0) return 0;
    if (x % 2) {
        printf("%d ", x);
        myfunc(x - 1);
    } else {
        myfunc(x - 1);
        printf("%d ", x);
    }
}
```

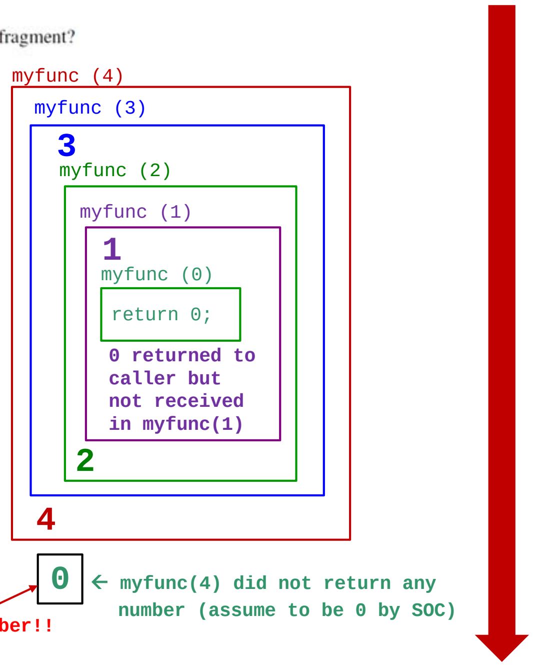
A. 4 3 2 1 0

B. 0 3 4 1 2

C. 0 4 2 3 1

D. 3 1 2 4 0

E. None of the above



myfunc (4)

myfunc (3)

3

myfunc (2)

myfunc (1)

1

myfunc (0)

return 0;

0 returned to caller but not received in myfunc(1)

2

4

0 ← myfunc(4) did not return any number (assume to be 0 by SOC)

Any number!!

19. What is printed by the following C program fragment?

```
int myfunc(int);

main() {
     printf("%d ", myfunc(4));
}

int myfunc(int x) {
    if (x <= 0) return 2214;
    if (x % 2) {
        printf("%d ", x);
        myfunc(x - 1);
    } else {
        myfunc(x - 1);
        printf("%d ", x);
    }
}
```

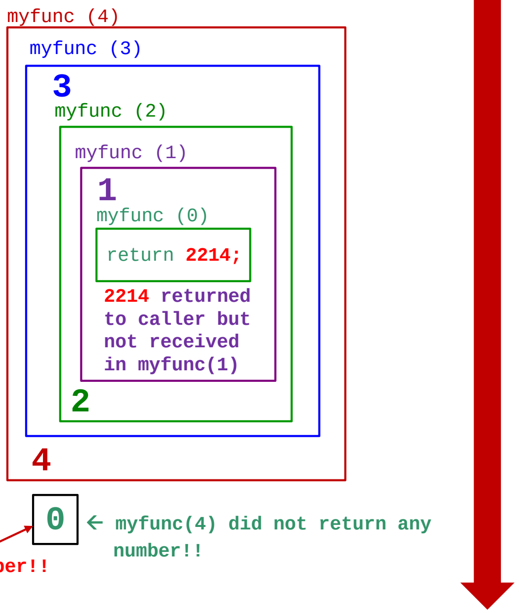
A. 4 3 2 1 0

B. 0 3 4 1 2

C. 0 4 2 3 1

D. 3 1 2 4 0

E. None of the above



myfunc (4)

myfunc (3)

3

myfunc (2)

myfunc (1)

1

myfunc (0)

return 2214;

2214 returned to caller but not received in myfunc(1)

2

4

0

Any number!!

← myfunc(4) did not return any number!!