



Recursion I

CS2263 – Systems Software Development

Learning Outcomes

At the conclusion of this lecture students should be able to:

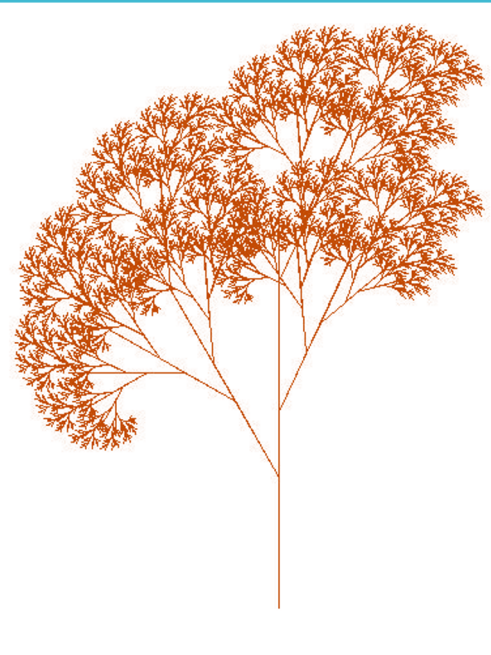
- Explain the strategy of recursion
- Explain the general format of recursion
- Program a simple case of recursion



References

- Lu, Yung-Hsiang. 2015. Intermediate C Programming. CRC Press. New York. (Chapter 12)





Recursion ...

<https://upload.wikimedia.org/wikipedia/commons/f/f7/RecursiveTree.JPG>

- Used to address problems that can be broken down into simpler versions of themselves
- Functions that call themselves. The classic is factorials:
 $\text{factorial}(n) = n * \text{factorial}(n-1)$
 $\text{factorial}(n-1) = n-1 * \text{factorial}(n-2)$
 $\text{factorial}(n-2) = n-2 * \text{factorial}(n-3)$
...
- A bit more concrete:
 $\text{factorial}(3) = 3 * \text{factorial}(2)$
 $\text{factorial}(2) = 2 * \text{factorial}(1)$
 $\text{factorial}(1) = 1$



Recursion Steps

1. Identify the argument(s) of a problem
2. Express the solution based on the argument(s)
3. Determine the simple case(s) where the solutions are “obvious”
4. Derive the relationships between the complex case(s) and the simpler case(s)



General Form of Recursion

Base Case

- The case to which we have an answer

General Case

- The case that expresses the solution in terms of a call to itself with a smaller version of the problem



C: General Form of Recursion

```
T func(arguments){  
    if (base case) //by checking arguments  
        solve the problem  
    else // recursive case  
        func(simplified arguments)  
}
```



What the Stack is Happening?

C is pass-by-value; argument copies are used in each stack frame

If we were passing by address, this wouldn't work.

printf()s are placed to report after the recursive call so that the stack addressing is presented as it would appear in the stack diagram.

```
int factorial(int iVal){
    int iFac;
    if(iVal == 1){
        printf("iVal (%p): %d\n", &iVal, iVal);
        return 1;
    }

    iFac = iVal * factorial(iVal-1);
    printf("iVal (%p): %d\n", &iVal, iVal);
    return iFac;
}

/*
 * Example Output for 5:
 *
 * iVal (0x7ffeed85a8b8): 1
 * iVal (0x7ffeed85a8e8): 2
 * iVal (0x7ffeed85a918): 3
 * iVal (0x7ffeed85a948): 4
 * iVal (0x7ffeed85a978): 5
 */
```



Bigger Example: Balls of Two Colours

- Unlimited red and blue balls in a bag.
- Game selects n balls,
 - red balls cannot be selected one after another
- Order matters (red, blue) \neq (blue, red)
- How many different ways can the balls be selected?

$f(n) =$		n
	2	1
	3	2
	$f(n-1) + f(n-2)$	>2



Balls of Two Colours (I)

```
int f(int m) {  
    if (m <= 0){  
        // Invalid. Number must be positive.  
        return -1;  
    }  
    // Base cases  
    if (m == 1)  
        return 2; // f(1) = 2  
    if (m == 2)  
        return 3; // f(2) = 3  
    // Recursive case  
    int a = f(m - 1); // Episode A  
    int b = f(m - 2); // Episode B  
    return (a + b);  
}
```



Balls of Two Colours Realized (II)

```
int main(int argc, char * argv[]){  
    int c, n;  
    if (argc < 2){  
        printf("need 1 integer.\n");  
        return EXIT_FAILURE;  
    }  
    n = (int) strtol(argv[1], NULL, 10);  
    c = f(n);  
    printf("f(%d) = %d.\n", n, c);  
    return EXIT_SUCCESS;  
}
```



Recursion Runtime Stack: Main

Balls of two colours
progression

```
int main(int argc, char * argv[]){
    int c, n;
    if (argc < 2){
        printf("need 1 integer.\n");
        return EXIT_FAILURE;
    }
    n = (int) strtol(argv[1], NULL, 10);
    c = f(n);
    printf("f(%d) = %d.\n", n, c);
    return EXIT_SUCCESS;
}
```

Frame	Symbol	Address	Value
	n	998	3
main	c	999	garbage



Recursion Runtime Stack

Balls of two colours
progression

Call and step into f()

```
int f(int m) {  
    // Base cases  
    if (m <= 0) return -1; // Invalid. Number must be positive.  
    if (m == 1) return 2; // f(1) = 2  
    if (m == 2) return 3; // f(2) = 3  
    // Recursive case  
    int a = f(m - 1); // Episode A  
    int b = f(m - 2); // Episode B  
    return (a + b);  
}
```

Frame	Symbol	Address	Value
	b	992	garbage
	a	993	garbage
	m	994	3
f	Value Address	995	999
	Return	996	RL
	n	998	3
main	c	999	garbage



Recursion Runtime Stack

Balls of two colours
progression

Call and step into f(),
Episode A

```
int a = f(m - 1); // Episode A
int b = f(m - 2); // Episode B
return (a + b);
```

Frame	Symbol	Address	Value
	b	988	garbage
	a	989	garbage
	m	990	2
f	Value Address	991	994
	Return	992	RL
	b	993	garbage
	a	994	garbage
	m	995	3
f	Value Address	996	999
	Return	997	RL
	n	998	3
main	c	999	garbage



Recursion Runtime Stack

Balls of two colours
progression

Return from f()
Episode A back into
f()

```
int a = f(m - 1); // Episode A
int b = f(m - 2); // Episode B
return (a + b);
```

Frame	Symbol	Address	Value
	b	993	garbage
	a	994	3
	m	995	3
f	Value Address	996	100
	Return	997	RL
	n	998	999
main	c	999	garbage



Recursion Runtime Stack

Balls of two colours
progression

Call and step into f(),
Episode B

```
int a = f(m - 1); // Episode A
int b = f(m - 2); // Episode B
return (a + b);
```

Frame	Symbol	Address	Value
	b	988	garbage
	a	989	garbage
	m	990	1
f	Value Address	991	993
	Return	992	RL
	b	993	garbage
	a	994	3
	m	995	3
f	Value Address	996	100
	Return	997	RL
	n	998	999
main	c	999	garbage



Recursion Runtime Stack

Balls of two colours
progression

Return from f(),
Episode B back into
f()

```
int a = f(m - 1); // Episode A
int b = f(m - 2); // Episode B
return (a + b);
```

Frame	Symbol	Address	Value
	b	993	2
	a	994	3
	m	995	3
f	Value Address	996	999
	Return	997	RL
	n	998	3
main	c	999	garbage



Recursion Runtime Stack: Main

Balls of two colours
progression

Completion of f()

```
int main(int argc, char * argv[]){
    int c, n;
    if (argc < 2){
        printf("need 1 integer.\n");
        return EXIT_FAILURE;
    }
    n = (int) strtol(argv[1], NULL, 10);
    c = f(n);
    printf("f(%d) = %d.\n", n, c);
    return EXIT_SUCCESS;
}
```

Frame	Symbol	Address	Value
	n	998	3
main	c	999	5

