Program Patterns
Recursion, Update/Copy/Move Patterns

J.W. Choi

2024

Recursion
(Recursive Function)

Recursion Example: Factorial

- 0! = 1   (recursion termination condition)
- n! = n * (n-1)!   for n >= 1

- 3! = 3 * 2!
- 2! = 2 * 1!
- 1! = 1 * 0!
- 0! = 1

- 3! = 3 * (2 * (1 * (1))) = 6

Factorial in C

pseudo code for factorial

```
if n=0
   factorial = 1
else
   factorial = n * factorial(n-1)
```

```c
int factorial (int n) {
   if (n==0)
      return (1);
   else
      return (n * factorial(n-1));
}
```

Recursion

- Solution in terms of itself
- Must terminate

- Recursive functions have equivalent iterative functions.

Essence of Recursion

- Remember
  - n = n -1
    - make the problem smaller
  - n = 0 or 1
    - terminate recursion

- Experiment
  - n = 2, 3, 4, or 5

- Draw a Recursion Diagram

How Recursion Works: First Way to Understand It

```
void main(){
    result = factorial(3);
}

int factorial (int n) {
    if (n==0)
        return (1);
    else
        return (n * factorial(n-1));
}
```

Evaluating Recursion (Using an Array)

```
void main(){
    result = factorial(3);
}

int factorial (int n) {
   if (n==0)
      return (1);
   else
      return (n * factorial(n-1));
}
```

(Step by Step Illustration) Start

result =	factorial(3)

(1/8)

return (3 *	factorial(2)
result =	factorial(3)

(2/8)

return (2 *	factorial(1)
return (3 *	factorial(2)
result =	factorial(3)

(3/8)

return (1 *	factorial(0)
return (2 *	factorial(1)
return (3 *	factorial(2)
result =	factorial(3)

(4/8)

return (1)	
return (1 *	factorial(0)
return (2 *	factorial(1)
return (3 *	factorial(2)
result =	factorial(3)

(5/8)

return (1 *	1
return (2 *	factorial(1)
return (3 *	factorial(2)
result =	factorial(3)

(6/8)

return (2 *	1
return (3 *	factorial(2)
result =	factorial(3)

(7/8)

return (3 *	2
result =	factorial(3)

(8/8)

result =	6

How Recursion Works: Second Way to Understand It

```
void main(){
   result = factorial(3);
}

int factorial (int n) {
   if (n==0)
      return (1);
   else
      return (n * factorial(n-1));
}
```

1. factorial(3)
   3 * factorial(2)
2. factorial(2)
   2 * factorial(1)
3. factorial(1)
   1 * factorial(0)
4. factorial(0)
   return 1

5. 1 * 1
6. 2 * (1 * 1)
7. 3 * (2 * (1 * 1))

Recursion Diagram (1)

factorial(3)
return (3 * factorial(2))

factorial(2)
return (2 * factorial(1))

factorial(1)
return (1 * factorial(0))

factorial(0)
return (1)

Recursion Diagram (2)

```
factorial(3)
3 * factorial(2)
      2 * factorial(1)
            1 * factorial(0)
                   return 1

            1 * 1
      2 * (1 * 1)
3 * (2 * (1 * 1))
```

Simple Thinking
(*Imagine Each Call Is to a Different Function)

```
void main(){
    result = factorial(3);
}

int factorial (int n) {
    if (n==0)
        return (1);
    else
        return (n * factorial(n-1));
}
```

1. factorial1(3)
   3 * factorial(2)
2. factorial2(2)
   2 * factorial(1)
3. factorial3(1)
   1 * factorial(0)
4. factorial4(0)
   return 1

5. 1 * 1
6. 2 * (1 * 1)
7. 3 * (2 * (1 * 1))

Exercise: Write a recursive C function that returns the n^{th} number in a Fibonacci sequence, and draw a recursion diagram.

- Fibonacci sequence

  0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,...

- How to formulate this?
  - What is the pattern?
  - How to terminate the recursion?

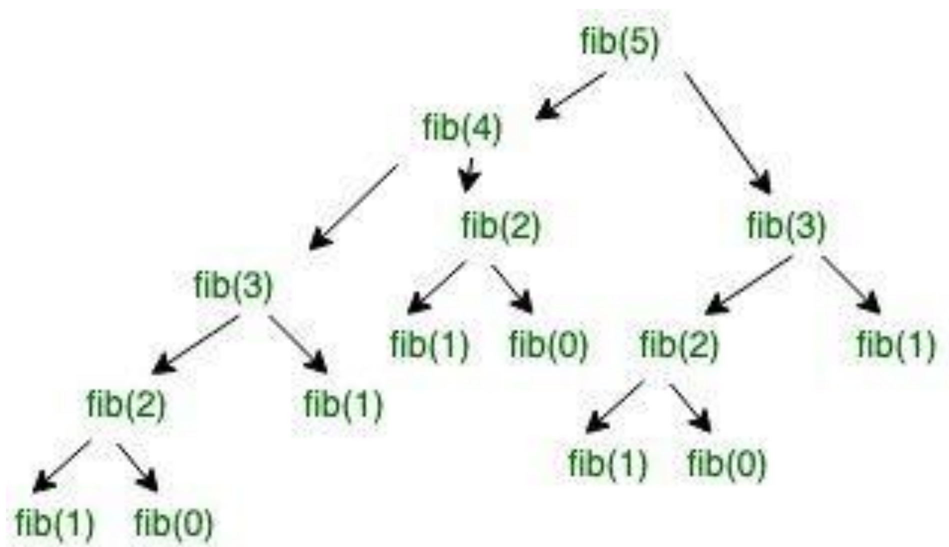
Solution (formulation and code)

- Fib(n) = n   for n < 2
- Fib(n) = Fib(n-1) + Fib(n-2) for n >=2

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

Recursion Diagram (for n=5)

- Fib(n) = n   for n < 2
- Fib(n) = Fib(n-1) + Fib(n-2) for n >=2



Exercise: Draw a Recursion Diagram for wrt_backward (using “okay₩n” as input)

```
#include <stdio.h>
void wrt_backward(void);

void main () {
    printf (“input a line”);
    wrt_backward();
    printf (“₩n”);
}

void wrt_backward() {
    int c;
    if ((c = getchar()) != ‘₩n’)
        wrt_backward();
    putchar(c);
}
```

Recursive Function vs. Iterative Function

- For a recursive function, there is an equivalent iterative function.
- Recursive function may be more compact.
  - Often used for operations on data structures

Exercise: Write an Iterative Factorial Function

int iter_factorial (int n)

Solution

```
int iter_factorial (int n) {
    int result = 1;
    int k;
    for (k = 1; k <= n; k++) {
        result = result * k;
    }
    return result;
}
```

Exercise: Write a Recursive Function

- for adding the first n elements of an array a[ ]

```
int sum_of (int a[], int n)
```

Solution

```
int sum_of (int a[], int n) {
    if (n < 1 || n > MAX) {
        printf ("array boundary error");
        exit(1);
    } else
        if (n == 1)
            return a[0];
        else
            return (a[n-1] + sum_of (a, n-1));
```

Exercise: Draw a Recursion Diagram

- For adding the first 4 elements of array (20, 30, 10, 50, 15, 45, 80, 25)

Exercise: Write a Recursive power Function

- double power (float, int)

Solution

```
double power(float val, int pow) {

    if (pow == 0)       /* power(x, 0) returns 1 */
        return(1.0);
    else
        return (power (val, pow - 1) * val); }
```

Exercise

- Draw a recursion diagram for power (100, 3)

Exercise : binary search

- Draw a recursion diagram for n=5, and list contains 2,5,8,12,16,23,38,56,72,91
- State what the program is designed to do.
  - To find this, experiment with a few lists of different sizes (n), and different contents.
  - Note: The list must be pre-sorted; for example,

    (1,3,4,6,8,10,11),

    (23,24,25, 27,29,30,35,40,45,46,50)

Example Program

```c
int binarySearch(int arr[], int left, int right, int key) {

    // checking if there are elements in the subarray
    if (right >= left) {

        // calculating mid point
        int mid = left + (right - left) / 2;

        // If the key is present at the middle itself
        if (arr[mid] == key)
            return mid;

        // If key is smaller than arr[mid], then it can only
        // be present in left subarray
        if (arr[mid] > key)  {
            return binarySearch(arr, left, mid - 1, key);
        }

        // Else the key can only be present in right
        // subarray
        return binarySearch(arr, mid + 1, right, key);
    }

    // We reach here when element is not present in array
    return -1;
}
```

```c
int main(void) {
    int arr[] = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91};
    int size = sizeof(arr) / sizeof(arr[0]);

    // element to be searched
    int key = 23;


    int index = binarySearch(arr, 0, size - 1, key);
    if (index == -1) {
        printf("Element is not present in array");
    }
    else {
        printf("Element is present at index %d", index);
    }

    return 0;
}
```

Program Patterns

Data Processing Program Patterns

- Data Search
- Data Update
- Data Copying & Moving
- Data Transformation
- Data Reorganization
- Data Derivation

Roadmap

- Data update
- Data copying/moving

Some WORD Operations
(Update After Search)

- Case Changes
  - all lower case, all uppercase, first letter uppercase
- Font Changes
  - bold, italic
  - Size
  - type
- Color Changes
- Move
- Copy
- Delete
- Update
- Insert

Types of Data Update

- insert
- append
- update/replace
- delete

- Often there are constraints on update.
  - semantic constraints
  - physical constraints

Constraints

- A constraint is a condition the data must satisfy for insert, update, and delete.
- (A Program) Must check constraints on insert, update and delete of data

Types of Constraint

- Semantic constraints on data
  - data type
    - char, int, float, ADT,…
  - data value range
    - (17..65), (>16000 && <250000),…
  - uniqueness of key
  - null-value allowed
  - conditional value
    - < avg (age), > min (salary),…
  - …
- Physical constraints on data
  - sort order (ascending or descending order)
  - physical size

Value Range Constraint

constraint: (>=17 && <=65)

Age
17
20
20
25
25
65

insert 75 (invalid)
insert 45 (valid)

Uniqueness Constraint

constraint: (e.g.) name must be unique.

name
Bae
Chung
Hong
Kong
Kim
Lee

insert Kim (invalid)
insert Choi (valid)

Conditional Value Constraint

constraint: (e.g.)
(age must be > or < 5 of the average age)

age

20
20
30
30

insert 75 (invalid)
insert 29 (valid)

Roadmap

- Data update
- Data copying/moving

Data Copying & Moving

- data copying
  - strcpy
  - strcat
- data moving
- data compaction

Data Copying

Data Copying

- From one data structure to another
  - between the same type of data structure
    - from a 1-dimensional array to a 1-dimensional array
    - from a singly linked list to a singly linked list
    - …
  - between different types of data structure
    - from an array to a linked list
    - from a singly linked list to a doubly linked list
    - …
- Data synchronization problem
  - Update to one copy may need to be made to all other copies.
- Data copy integrity problem
  - need to verify correctness when copying

Data Synchronization

Chung
changed to
Jeong

	copy 1		copy 2	copy 3
	Bae		Bae	Bae
Jeong	Chung	→	Jeong	Jeong
	Hong		Hong	Hong
	Kong		Kong	Kong
	Kim		Kim	Kim
	Lee		Lee	Lee

Integrity Problem on Data Copying/Transmission

original	copy	copy
Bae	Bae	Bae
Chung	Chung	Chung
Hong	Hong	Hing
Kong	Kong	King
Kim	Kin	Kom
Lee	Lee	Lee

Verification Data

- Checksum
  - Use computation on all data being copied or moved
  - exclusive OR, add, some other function are used
- Special data
  - total count or average or timestamp or some other data can be separately stored or computed (to verify other data).

Checksum

checksum = Bae ^ Chung ^ Hong ^ Kong ^....

(* ^ is bit-wise exclusive OR (later))

Checksum is computed for the original.

Checksum is computed for the copy and compared with the checksum for the original.

If the two checksums are the same, copying was OK.

original	copy
Bae	Bae
Chung	Chung
Hong	Hong
Kong	Kong
Kim	Kim
Lee	Lee
checksum	checksum

Data Moving

Data Moving in an Array Due To Insert

45	25	34	17	45	36	

insert 52

logic:

define array-2.

copy data before the insert point in array-1
to array-2.

append the new data to array-2.

append data after the insert point in array-1
to array-2

Data Moving in an Array Due To Delete

45	25	34	17	45	36	

delete

logic-1

leave NULL for the deleted array element.

logic-2:

move all data after the delete point in the array one position to the left

Data Compaction

Data Compaction

- For main memory
  - Garbage collection
  - Fill unused memory locations by moving data from other memory locations.
- For hard disk drive
  - Fill unused memory locations by moving data from other memory locations.
  - Form blocks of unused memory locations for allocation later.

Illustration

45		34			36	

45	34	36				

Lab: Inserting a new element into a struct array with constraints

```
struct {
   char RRN[13];            // constraint: unique
   char name[20];
   float salary;
   float bonus;             //  constraint: bonus < salary
} employee[1000];
```

- Write the following C program:
  - Read RRN, name, salary and bonus and insert it to a struct array employee
  - Check constraints before insertion and print suitable error message if it is invalid
    - Two kinds of error messages – see above inline comments

End of Class