Introduction to Functions

Structured Programming

(Slides include materials from *The C Programming Language*, 2nd edition, by Kernighan and Ritchie and from *C Teach Yourself*, 3rd editions, by Herbert Shiield)

Functions – a big Topic

- Motivation—why needed
- Examples
- How two functions communicate
- void functions
- Function parameters
- Function returning values
- Local variable concept and scope rule
- Function prototypes & Header files

Definition – *Function*

- A fragment of code that accepts zero or more *values (*called *argument)*, processes those, produces a *result value*, and has zero or more *side effects*.
- A method of *encapsulating* a subset of a program or a system
  - To hide details
  - To be invoked from multiple places
  - To share with others

Functions – the bigger picture

Testing... Testing...

By Kyle Peterson

Today I woke up in a strange room... I can't exactly remember my name so I'm going to call myself Arron. I woke up with another person who said his name is Derrick and he was questioning me on where we were, I told him I don't know but he does not seem to believe me. I started looking around the room and couldn't find anything except for a nail, and without thinking stuck it in my pocket. No doors no windows nothing... Just a single flickering light and a man I've never seen before.

When we were both awake enough, we started feeling around the room, then something happened. I had pressed one of the tiles in and it revealed the numbers 971578. At the same time, I pressed that panel in another panel opened on the opposite side of the room, it was an exit! We both rushed over to the exit and left the room, but we didn't think to memorize those numbers and when we turned around, we watched the room we were just standing in collapse. We turned back around and saw a hallway, so we followed it as if nothing happened. All the sudden Derrick said "STOP", when we stopped, he told me he saw a figure holding a knife down the hallway. Right after he told me this the ground opened beneath us.

We fell for a good while until we hit water like smacking our ankles on cement. Once we swam a good hundred feet, we were able to stand but not really cause our ankles were too sore than all the sudden we both saw it, the figure holding a knife. At this moment we were a little bit un easy until lights in the water filled hallway came on and the figure disappeared. We walked a good hundred more feet until we hit a few stairs and a door. We walked into this room and it seemed like a nice place to rest. It had a TV, 2 beds, nice carpet, a bathroom, and a change of clothes.

We didn't think twice about falling asleep in this room, so we got ready and went to bed. When I woke up however, I didn't see Derrick anywhere and I was in a 3 foot by 3 foot chamber. So again, I spun around the room hitting random panels and finally one of them opened revealing a clear glass chamber with derrick inside. On the upper right-hand corner was a timer counting down from 4 hours, when I realized he might die in there if I don't get him out. I started pressing every other panel in the room when a door and a code board opened. The code board had a sign on it saying "6 Digit Starter Code can you make it". I Looked over at the door and it had 2 arrows, 1 saying Start, and the other saying exit. No matter how much the exit tempted me I decided to go back to the start room trying to figure out how to get to that code.

```
f1(){
        statement1;
        statement2;
        .....
}
```

```
f2(){
        statement1;
        statement2;
        .....

}
```

```
main(){
        statement1;
        statement2;
        .....
}
```

Motivation-why needed

- Functions
  - Modularize a program
- Benefits
  - Divide and conquer
    - Manageable program development
  - Software reusability
    - Use existing functions as building blocks for new programs
  - Avoids code repetition

Example

```
LIKE(){
    printf("Like ");
}

main(){
        printf("I ");
        printf("Like ");
        printf("C");
}
```

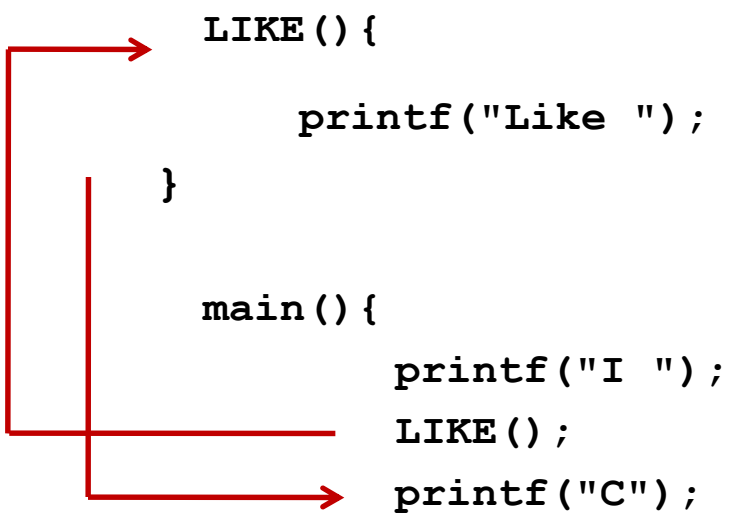
OUTPUT: I Like C

Example

```
LIKE(){
    printf("Like ");
}

main(){
        printf("I ");
        LIKE();
        printf("C");
}
```

OUTPUT: I Like C

Example



```
LIKE(){
    printf("Like ");
}

main(){
        printf("I ");
        LIKE();
        printf("C");
}
```

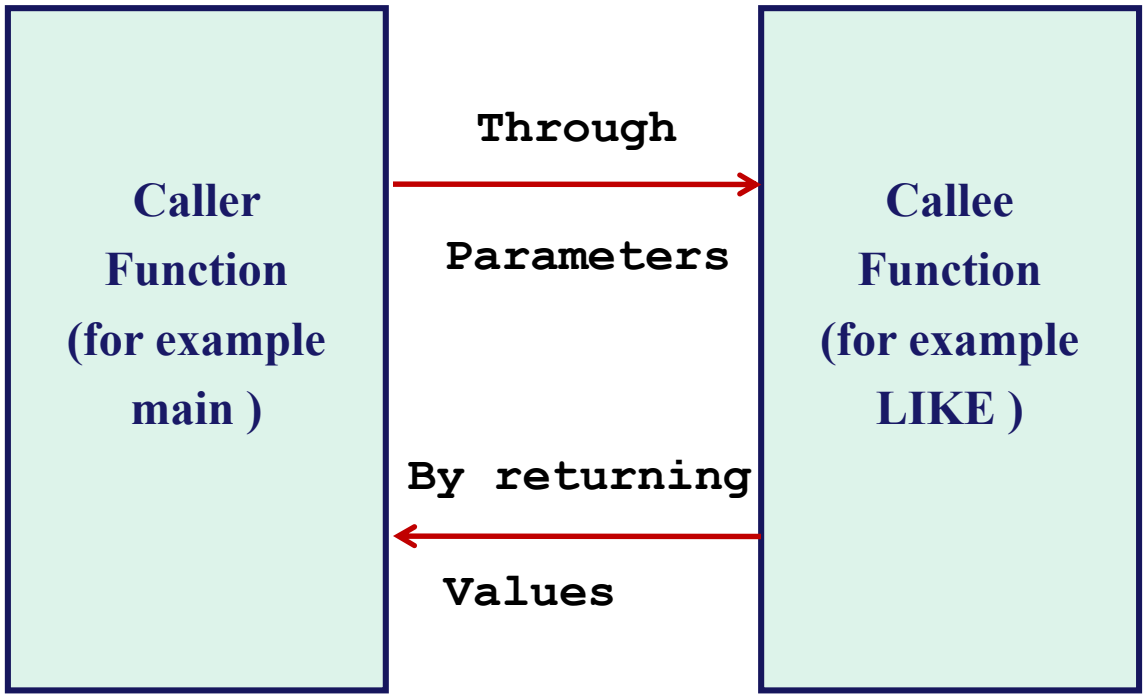
OUTPUT: I Like C

Example--Reusability

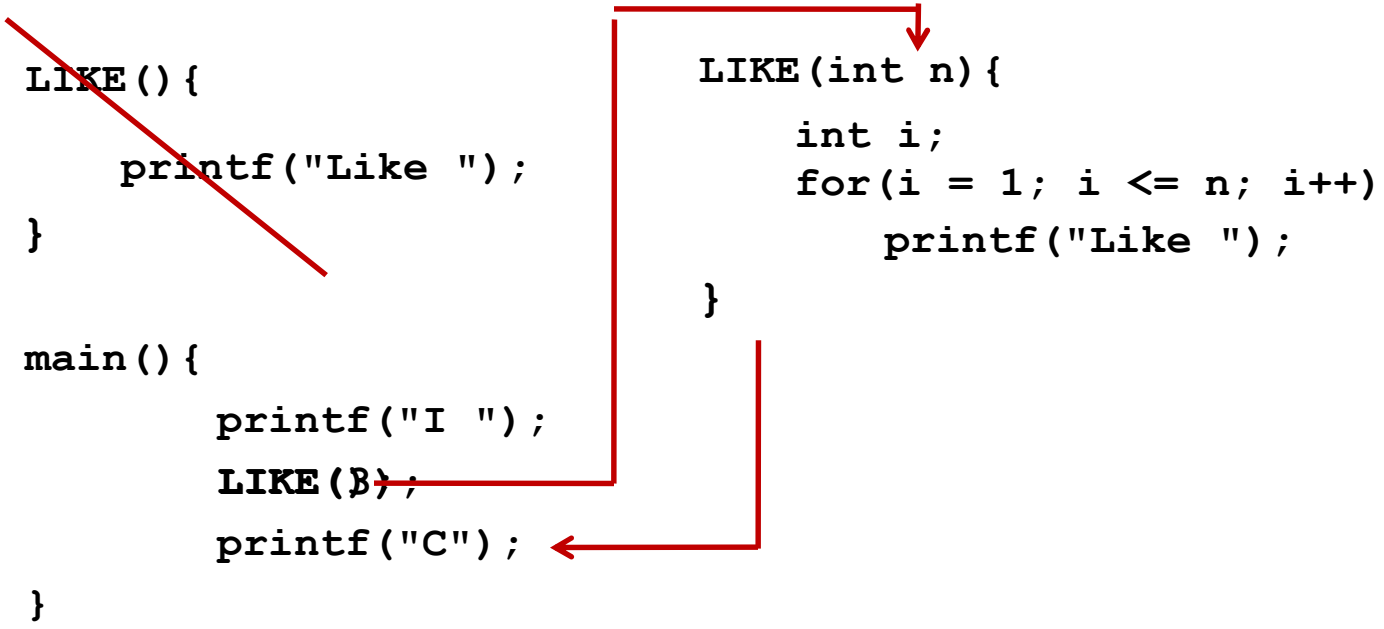
```
LIKE(){
    printf("Like ");
}

main(){
        int i;
        printf("I ");
        for(i = 1; i <= 5; i++)
                LIKE();
        printf("C");
}
```

OUTPUT: I Like Like Like Like Like C

How two functions communicates



Example: Parameter passing



```
LIKE(){
    printf("Like ");
}

main(){
        printf("I ");
        LIKE(3);
        printf("C");
}
```

```
LIKE(int n){
    int i;
    for(i = 1; i <= n; i++)
        printf("Like ");
}
```

OUTPUT: I Like Like Like C

Another Example: Parameter passing

```
oddOREven(int n){
        if(n%2 == 0)
                printf("Even\n");
        else
                printf("Odd\n");
}

main(){

        oddOREven(2);
        oddOREven(3);
}
```

OUTPUT: Even

OUTPUT: Odd

Returning Values from a Function

Steps for returning values from a function

1. **Mention return data type in front of function**
2. **Use return statement to return value**
3. **Catch the value from the caller using assignment (=) operator**

Example: Returning values

```
int max(int a, int b){
        int r;
        if (a > b)
                r = a;
        else
                r = b;
        return r;
}
main(){
        int p;
        p = max(3,4);
        printf("%d\n",p);
}
```

OUTPUT: 4

Three types of functions at a glance

No parameters
No return

```
LIKE(){
    printf("Like ");
}



main(){
  LIKE();
}
```

Has parameter
No return

```
oddOREven(int n){
   if(n%2 == 0)
     printf("Even\n");
   else
     printf("Odd\n");
}


main(){
   oddOREven(2);
}
```

Has parameter
Returns result

```
int max(int a, int b){
        int r;
        if (a > b)
                r = a;
        else
                r = b;
        return r;
}
main(){
   int p = max(4,6);
}
```

Function Definition

- Every function definition has the form

  return-type function-name (*parameter declarations*) **{**

  definitions and statements

  }

- If there is no parameter mention void
- If there is no return mention void

Three types of functions at a glance

No parameters
No return

```
LIKE(){
    printf("Like ");
}


main(){
  LIKE();
}
```

```
oddOREven(int n){
   if(n%2 == 0)
     printf("Even\n");
   else
     printf("Odd\n");
}


main(){
  oddOREven(2);
}
```

```
int max(int a, int b){
        int r;
        if (a > b)
                r = a;
        else
                r = b;
        return r;
}
main(){
  int p = max(4,6);
}
```

Three types of functions at a glance

Has parameter
No return

```
void LIKE(void){
    printf("Like ");
}

main(){
  LIKE();
}
```

```
oddOREven(int n){
   if(n%2 == 0)
     printf("Even\n");
   else
     printf("Odd\n");
}

main(){
   oddOREven(2);
}
```

```
int max(int a, int b){
        int r;
        if (a > b)
                r = a;
        else
                r = b;
        return r;
}
main(){
   int p = max(4,6);
}
```

Three types of functions at a glance

Has parameter
No return

```
void LIKE(void){
    printf("Like ");
}

main(){
  LIKE();
}
```

```
void oddOREven(int n){
   if(n%2 == 0)
     printf("Even\n");
   else
     printf("Odd\n");
}

main(){
   oddOREven(2);
}
```

```
int max(int a, int b){
        int r;
        if (a > b)
                r = a;
        else
                r = b;
        return r;
}
main(){
   int p = max(4,6);
}
```

Local variable concept

```
int max(int a, int b){
        int r;
        if (a > b)
                r = a;
        else
                r = b;
        return r;
}
```

```
main(){
        int p;
        p = max(3,4);
        printf("%d\n",p);
}
```

SELFISH principle: Variables declared within a function can only be used by that function

```
main(){

        max(3,4);
        printf("%d\n",r);
}
```

User input

```
int max(int a, int b){
        int r;
        if (a > b)
                r = a;
        else
                r = b;
        return r;
}
```

SELFISH principle: Variables declared within a function can only be used by that function

```
main(){
        int x,y,p;
        scanf("%d%d",&x,&y);
        p = max(x,y);
        printf("%d\n",p);
}
```

```
main(){
        int p;
        scanf("%d%d",&a,&b);
        p = max(a,b);
        printf("%d\n",p);
}
```

Global Variable Concept

```
int a, b;

int max(void){
        int r;
        if (a > b)
                r = a;
        else
                r = b;
        return r;
}

main(){
        int p;
        scanf("%d%d",&a,&b);
        p = max();
        printf("%d\n",p);
}
```

Function prototype—Why needed?

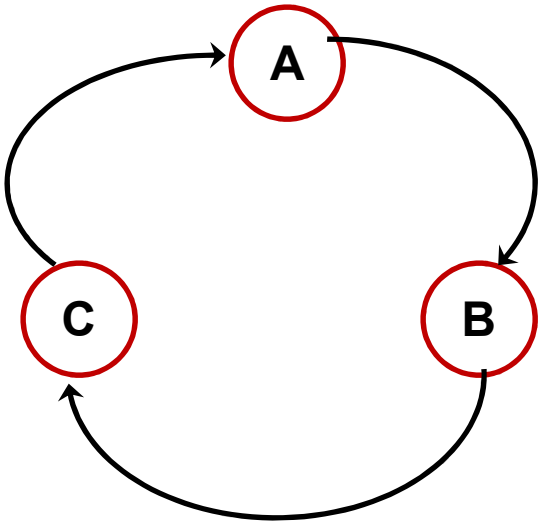
In all these three examples, functions have been written prior to the function call. But it can not be maintained always!!!

```
void LIKE(void){
    printf("Like ");
}

main(){
  LIKE();
}
```

```
void oddOREven(int n){
   if(n%2 == 0)
     printf("Even\n");
   else
     printf("Odd\n");
}

main(){
   oddOREven(2);
}
```

```
int max(int a, int b){
        int r;
        if (a > b)
                r = a;
        else
                r = b;
        return r;
}
main(){
   int p = max(4,6);
}
```

Function prototype—why needed?




```
A(){
      ….
      ….
}

C(){
      ….
      ….
      ….
}

B(){
      ….
      ….
      ….
}

A(){
      ….
      ….
      ….
}
```

Prototype

- Prototype means a replica of a real system

Function Prototype

- Definition:– a *Function Prototype* in *C* is a language construct of the form:–

return-type function-name (*parameter declarations*) ;

- i.e., exactly like a function definition, except with a `';'` instead of a *body* in curly brackets

Prototypes of our previously defined three functions at a glance

```
void LIKE(void);

void LIKE(void){
    printf("Like ");
}

main(){
  LIKE();
}
```

```
void oddOREven(int n);

main(){
  oddOREven();
}

void oddOREven(int n){
   if(n%2 == 0)
     printf("Even\n");
   else
     printf("Odd\n");
}
```

```
int max(int a, int b);

main(){
  int p = max(4,6);
}

int max(int a, int b){
        int r;
        if (a > b)
                r = a;
        else
                r = b;
        return r;
}
```

Library Functions

#include <math.h>

- **sin(x) // radians**
- **cos(x) // radians**
- **tan(x) // radians**
- **atan(x)**
- **atan2(y,x)**
- **exp(x)** // e^x
- **log(x)** // $\log_e x$
- **log10(x)** // $\log_{10} x$
- **sqrt(x)** // $x \geq 0$
- **pow(x, y)** // x^y
- **...**

#include <stdio.h>

- **printf()**
- **fprintf()**
- **scanf()**
- **sscanf()**
- **...**

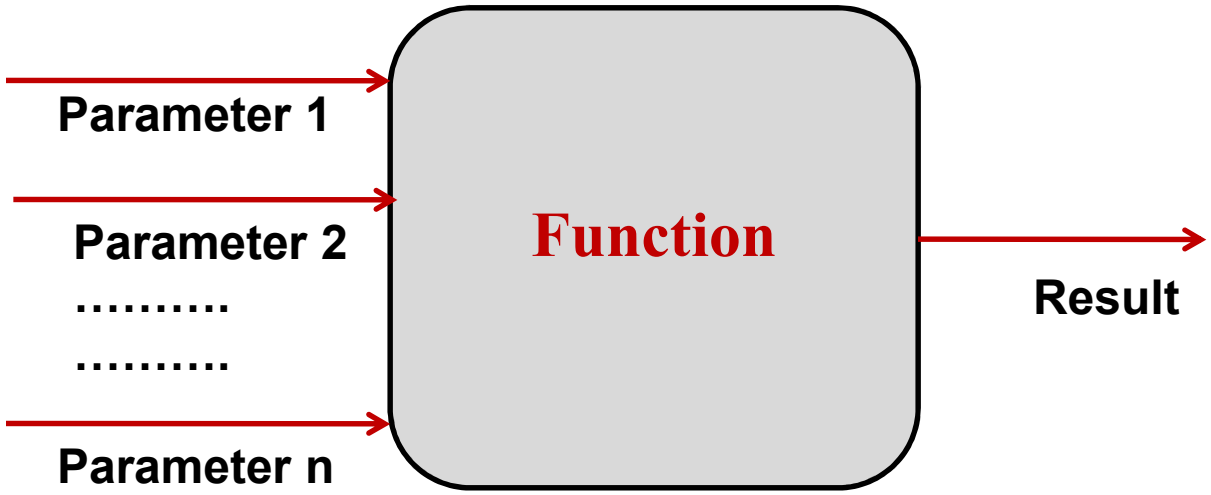
#include <string.h>

- **strcpy()**
- **strcat()**
- **strcmp()**
- **strlen()**
- **...**

Library functions' prototype & Header files

- Function prototypes of library functions are typically provided in *header files*
  - i.e., the '**`.h`**' files that programmers include in their code
- Grouped by related functions and features
  - To make it easier for developers to understand
  - To make it easier for team development
  - To make a package that can be used by someone else

A typical diagram of function

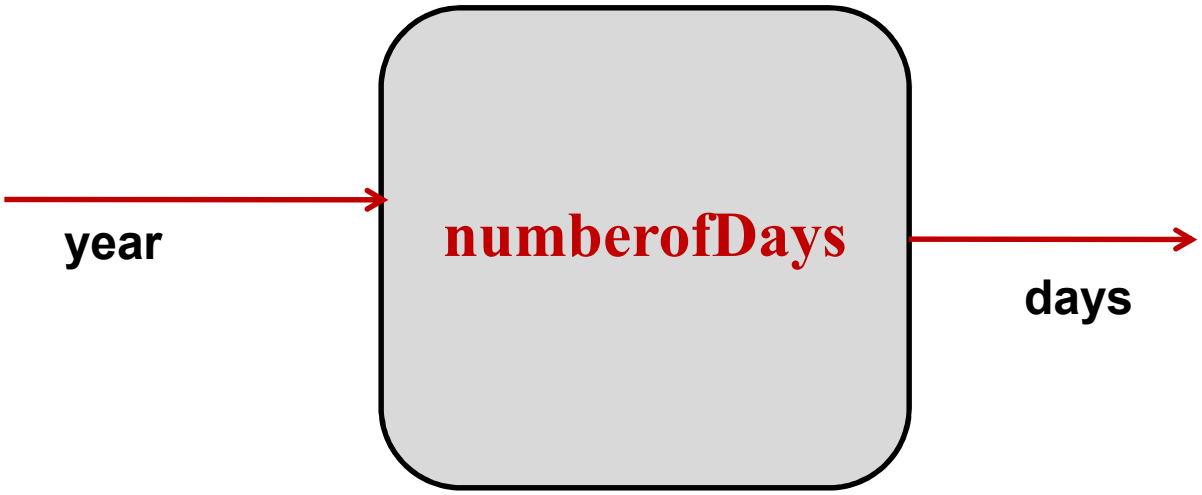



```
return-type function-name (parameter declarations) {
   definitions and statements
}
```

Example

Write down a function that will take a year as a parameter and will return number of days in that year. In your main function take user input for year and use this function, to print two things: (1) number of days in the year, and (2) whether the year is a leap year or not.

Solution diagram




```
return-type function-name (parameter declarations) {
   definitions and statements
}
```

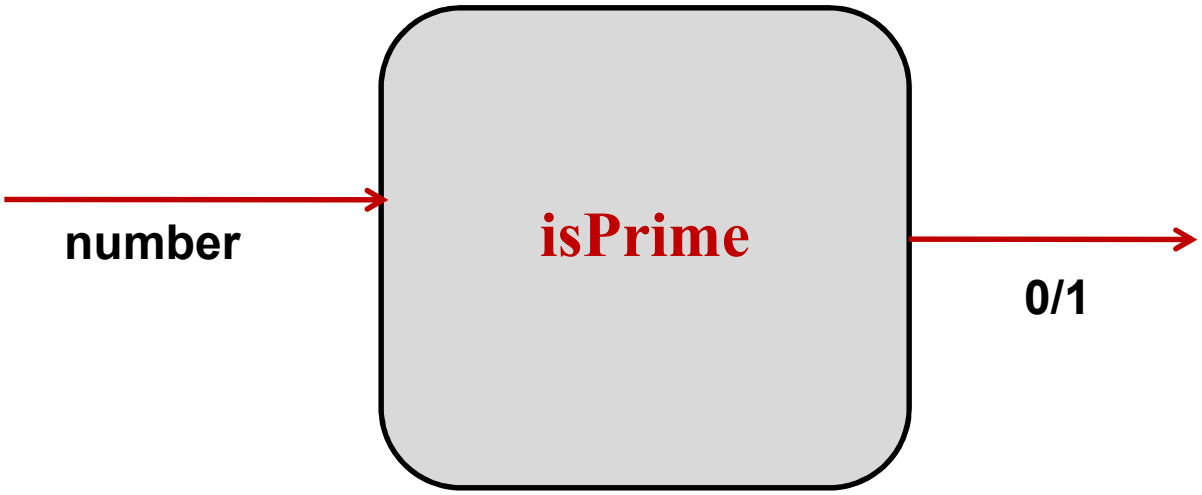
Solution

```
int numberofDays(int year){
        int days;
        if (year%400 == 0)
                days = 366;
        else if (year%100 == 0)
                days = 365;
        else if (year%4 == 0)
                days = 366;
        else    days = 365;
        return days;
}
void main(void){
        int y,p;
        scanf("%d",&y);
        p = numberofDays(y);
        printf("Number of days %d\n",p);
        if(p == 366) printf("LEAP YEAR");
        else    printf("Not a Leap Year ");
}
```

Example: Function returning indirect results

Write down a function that will take a number as a parameter and will return 1 if the number is a prime number and 0 otherwise. In your main function take a number as user input and use this function, to print whether the number is a prime number or not.

Solution diagram



return-type function-name (*parameter declarations*) {

definitions and statements

}

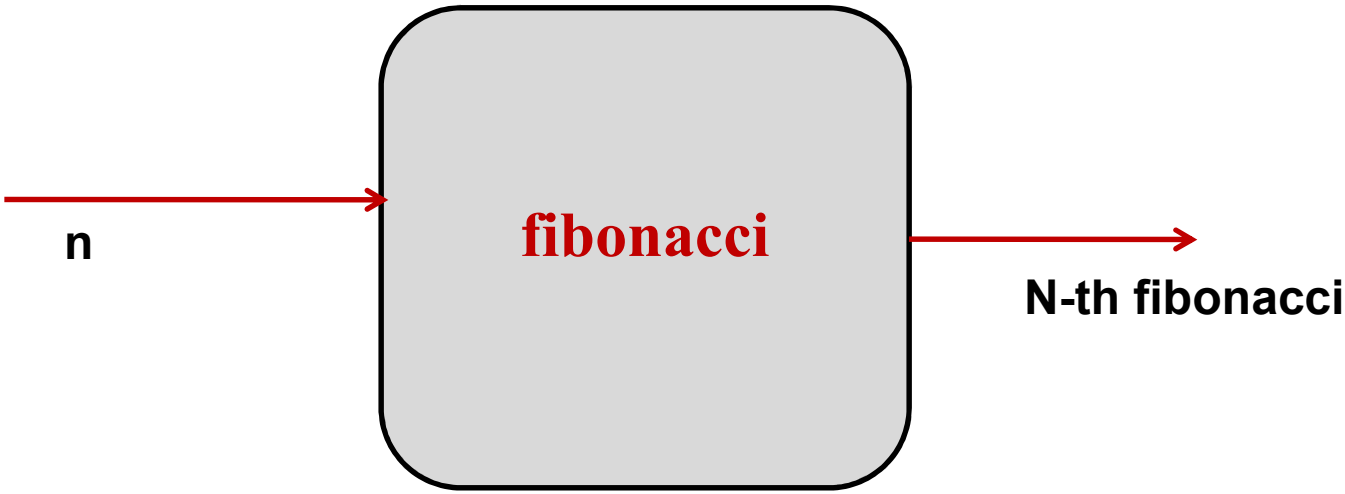
Solution

```
int isPrime(int n){
        int i, c = 0;
        for(i = 1; i <= n; i++){
           if (n%i == 0)
                c++;
        }
        if (c == 2) return 1;
        else return 0;
}


void main(void){
        int n,p;
        scanf("%d",&n);
        p = isPrime(n);
        if(p == 1) printf("YES");
        else   printf("NO");
}
```

Example: Function returning indirect results

Write down a function that will take a number n as a parameter and return n-th Fibonacci number. In your main function take a number N as user input and use this function, to show N-th Fibonacci number.

Solution diagram



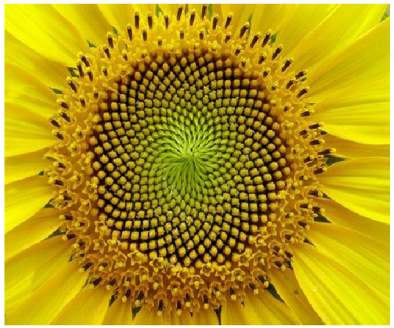

```
return-type function-name (parameter declarations) {
   definitions and statements
}
```

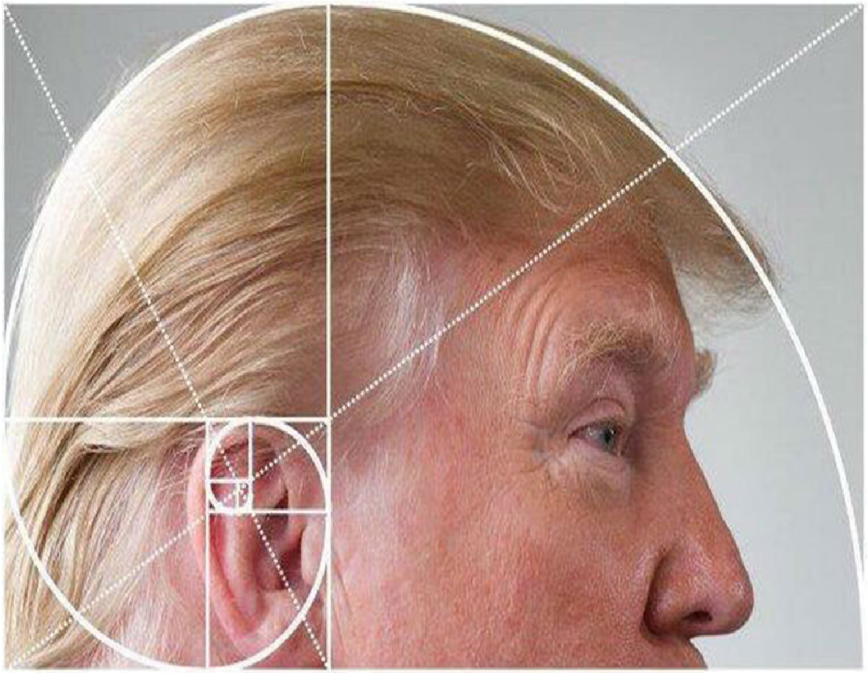
Fibonacci Series

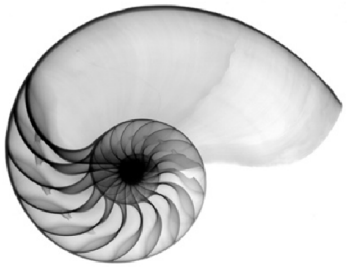
The **first** and **second** numbers in the Fibonacci sequence are 1

1 1 2 3 5 8 13 21 34 ...

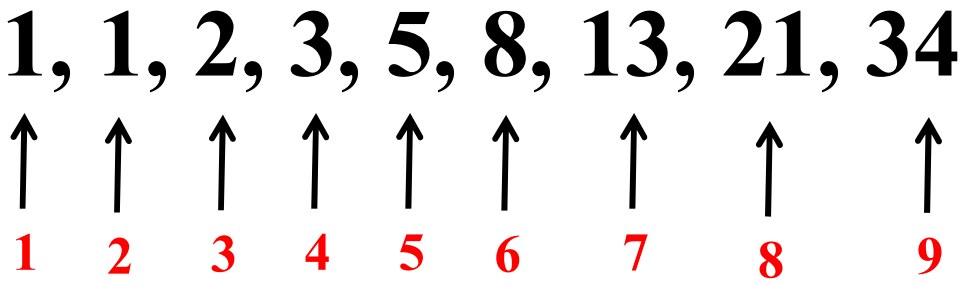
1+1 1+2 2+3 3+5 5+8 8+13







Fibonacci Series Generation



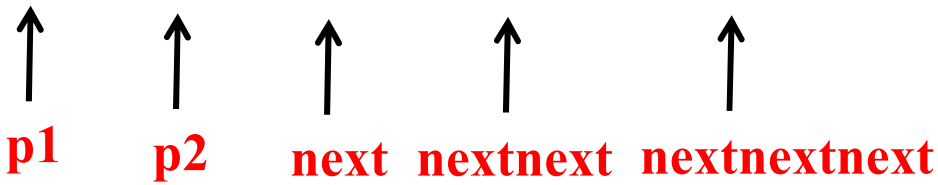
Write down a function that will print n-th Fibonacci number where n will be a parameter to the function.

n = 4 output → 3

n = 7 output →13

Fibonacci Series Generation

1, 1, 2, 3, 5, 8, 13, 21, 34, 55 ..




```
int fibonacci(int n){
        int i, p1,p2,  next;
        p1 = 1;
        p2 = 1;
        next = p1 + p2;
        nextnext = p2 + next;
        nextnextnext = next + nextnext;
        …..
        ….
     return next;
}
```

Solution

```
int fibonacci(int n){
        int i, p1,p2,  next;
         if (n <= 2)
                return 1;
        p1 = p2 = 1;
        for(i = 3; i <= n; i++){
           next = p1+p2;
           p1 = p2;
           p2 = next;
         }
     return next;
}

 void main(void){
         int N,p;
         scanf("%d",&N);
         p = fibonacci(N);
        printf("%d", p);
 }
```

Example: Function accepting array as parameter

Write down a function that will take a set of numbers as a parameter and will find and return maximum value within the set. Show how we can use this function from main to find and print maximum value.

Solution diagram


```
return-type function-name (parameter declarations) {
   definitions and statements
}
```

Solution

```
int findMax(int x[], int n){
        int i, max;
        max = x[0];
        for(i = 1; i < n; i++){
           if (max < x[i])
                max = x[i];
        }
        return max;
}

void main(void){
        int a[] = {34, 21, 65, 78, 90};
        int b[] = {4, 2, 8};
        int r;
        r = findMax(a, 5);
        printf("%d\n", r);
        r = findMax(b, 3);
        printf("%d", r);
}
```

Example: Complex problem

Write down a program that will print all prime factors of a number N given as input. Modularize your program by implementing following three functions:

(1) isFactor(n, x): is x a factor of n?

(2) isPrime(n): is n prime?

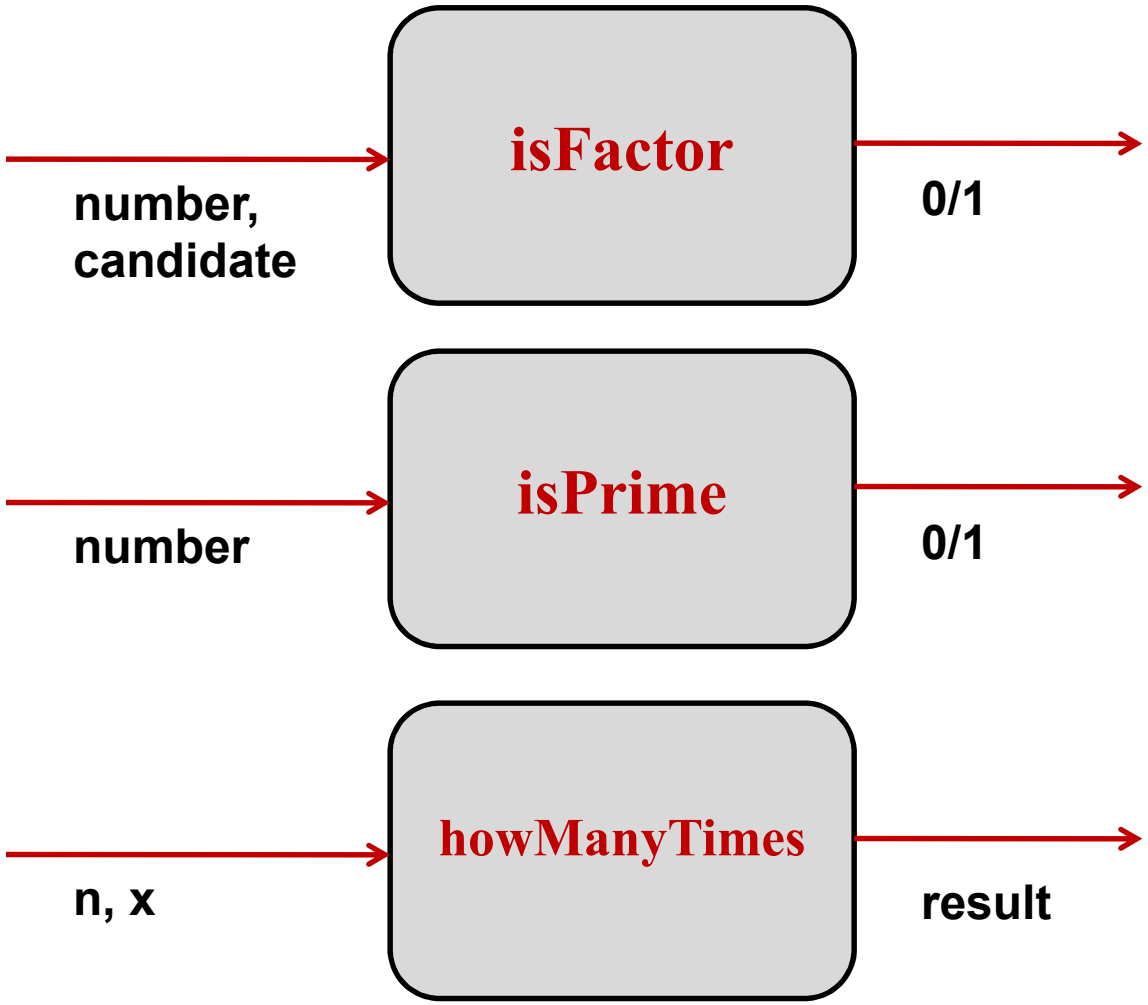
(3) howManyTimes(n, x): what is the power of x in n?

Example: Complex problem

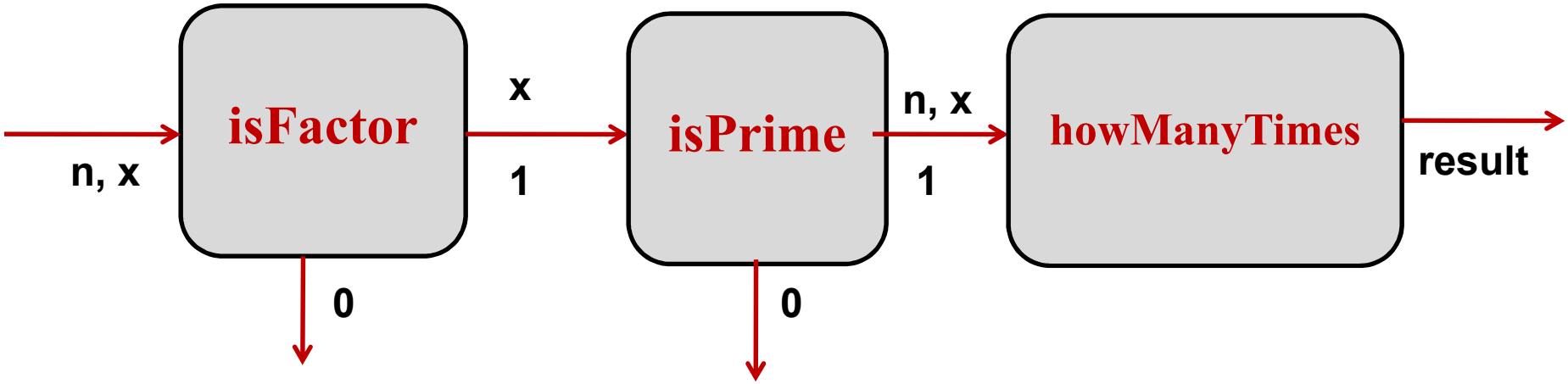
$$24 = 4 \times 6$$
$$= (2 \times 2) \times (2 \times 3)$$
$$= (2 \times 2 \times 2) \times 3$$
$$= 2(3)\ 3(1)$$

$$54 = 2 \times 27$$
$$= 2 \times (3 \times 3 \times 3)$$
$$= 2\ (1)\ 3(3)$$

Solution diagram



Solution diagram




```
return-type function-name (parameter declarations) {
   definitions and statements
}
```

Solution

```
int isFactor(int n, int x){
        return (n%x == 0);
}
int howManyTimes(int n, int x){
        int c = 0;
        while(n%x == 0){
                c++;
                n = n/x;
        }
        return c;
}
                void main(void){
                        int N,i;
                        scanf("%d",&N);
                        for(i = 2; i <= N; i++)
                           if(isFactor(N,i) && isPrime(i))
                                printf("%d(%d)",i,howManyTimes(N,i));
                }
```

```
int isPrime(int n){
        int i;
        if (n == 1) return 0;
        for(i = 2; i*i <= n; i++)
           if (n%i == 0)
                return 0;
        return 1;
}
```

Are we done?

Not really…. Optimized version

Solution

```
int isFactor(int n, int x){
        return (n%x == 0);
}
int howManyTimes(int n, int x){
        int c = 0;
        while(n%x == 0){
                c++;
                n = n/x;
        }
        return c;
}
```

```
int isPrime(int n){
        int i;
        if (n == 1) return 0;
        for(i = 2; i*i <= n; i++)
           if (n%i == 0)
                return 0;
        return 1;
}
```

```
void main(void){
        int N,i;
        scanf("%d",&N);
        for(i = 2; i <= N/2; i++)
           if(isFactor(N,i) && isPrime(i))
                printf("%d(%d)",i,howManyTimes(N,i));
        if(isPrime(N))  printf("%d(%d)",i,howManyTimes(N));
}
```

Questions?