Problem Solving using Functions

Rupesh Nasre.

IIT Madras
November 2022

Functions make programs modular.

- Common functionality
- Logical steps
- Can be parameterized, can return a value
- Examples
  - printf, scanf, pow, strlen, strstr, ...
  - You will define your own.

Modular Functionality

```
takeInput(&input);
output = process(input);
storeOutput(output);
```

```
for (int ii = 2; ii < N; ++ii)
    if (isPrime(ii))
        printf("%d\n", ii);
```

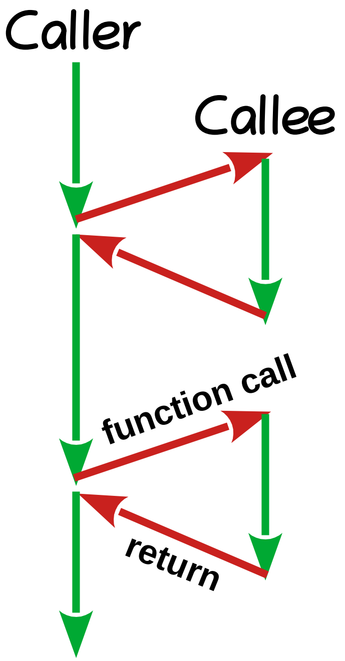
```
for (int ii = 2; ii < N; ++ii) {
  findFactors(ii, &factors, &nfactors);
  for (int ff = 0; ff < nfactors; ++ff)
      if (isPrime(factors[ff]))
          printf("%d\n", factors[ff]);
```

```
gets(str);
for (int ii = 0; str[ii] != '\0'; ++ii)
    if (!valid(str[ii]))
        error(str);
```

```
#define ARGS left, right, top, bot
while (left <= right && top <= bot) {
    leftToRight(ARGS);   top++;
    topToBot(ARGS);      right--;
    rightToLeft(ARGS);   bot--;
    botToTop(ARGS);      left++;
}
```

```
initFreq(&freq);
for (int ii = 0; str[ii] != '\0'; ++ii)
    findFreq(str, ii, &freq);
int maxIndex = findMax(freq);
printAlpha(maxIndex);
```

```
gets(forward);
reverse(forward, &reverse);
if (strcmp(forward, reverse) == 0)
    printf("Palindrome\n");
```



isPrime is a function which takes an integer and returns 0 or 1.

Caller

```
for (int ii = 2; ii < N; ++ii)
    if (isPrime(ii))
        printf("%d\n", ii);
```

Go back to the caller.

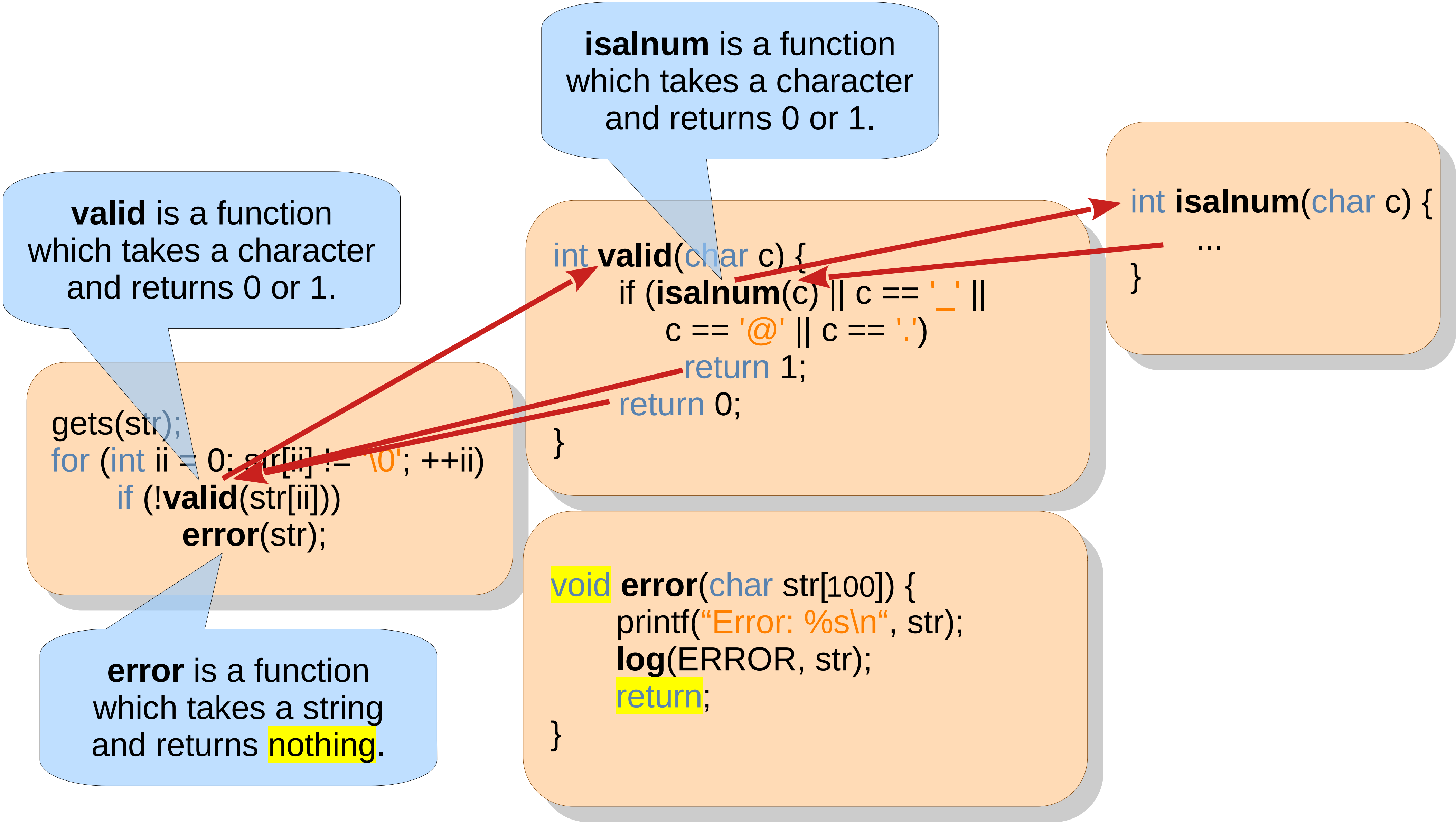
Callee

```
int isPrime(int num) {
    for (int ii = 2; ii < num; ++ii)
        if (num % ii == 0) return 0;
    return 1;
}
```

As a good software engineering practice, it is useful if changing the **implementation** of the callee does not change the caller.

```
int isPrime(int num) {
  if (num % 2 == 0) return 0;
  for (int ii=3; ii <= sqrt(num); ii += 2)
     if (num % ii == 0) return 0;
     return 1;
}
```

```
int isPrime(int num) {
    int p100[] = {2, 3, 5, …, 89, 97};
    int nump100 = sizeof(p100) / sizeof(p100[0]);
    for (int ii = 0; ii < nump100; ++ii)
        if (num % p100[ii] == 0) return 0;
    for (int ii = 101; ii*ii < num; ii += 2)
        if (num % ii == 0) return 0;
    return 1;
}
```



valid is a function
which takes a character
and returns 0 or 1.

```
gets(str);
for (int ii = 0; str[ii] != '\0'; ++ii)
    if (!valid(str[ii]))
        error(str);
```

isalnum is a function
which takes a character
and returns 0 or 1.

```
int valid(char c) {
    if (isalnum(c) || c == '_' ||
        c == '@' || c == '.')
            return 1;
    return 0;
}
```

```
int isalnum(char c) {
    ...
}
```

error is a function
which takes a string
and returns nothing.

```
void error(char str[100]) {
    printf(“Error: %s\n“, str);
    log(ERROR, str);
    return;
}
```

#define vs Functions

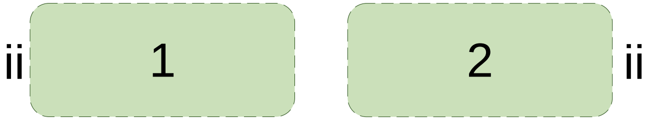
- You can pass arbitrary text to #defines
  - e.g., macro(a =, -b -) is possible.
  - Functions must follow strict typing rules.
- Compiler performs checks on both the preprocessed output and function calls. But it is easier to understand (and debug) function calls.
  - Use *gcc -E file.c* for the preprocessed output.
- Functions can be separately linked. Macros need to be present in each compilation unit.
- Typically, for very short codes (one or two lines), we use macros; for others, we use functions.

Scoping

```
int gg = 5;
int main() {
        int jj = gg + myfun();
}
```

```
int myfun() {
        int ii = 1;
        { int ii = 2;      // allowed.
        } printf("ii = %d\n", ii);  // 1
        return ii + 1;
}
```

- Each variable belongs to a scope. The same scope cannot define two variables of the same name.
- Each function has a scope. A scope can be created using braces {...}.



ii 1 2 ii

Try scope.c.

- A variable of the same name can be redefined in a different scope.It refers to the one in the latest scope.
- A global scope is accessible to all the functions.
  - Useful to avoid passing parameters
  - Should be used sparingly, only for large important data
- A variable may overwrite another in between a scope.

Functions with No Arguments

Stateless

```
double pi() {
      return 3.141592653589793238;
}
```

```
void warning() { printf("WARNING\n");  }
void error()    { printf("ERROR\n");      }
```

```
#define USD2INR       82.23
float getUSDExchangeRate() {
      return USD2INR;
}
```

Typically, the output remains fixed for functions with zero arguments.

But how can you change it?

Stateful

```
int getNextNumber() {
      return rand() % 100;
}
```

```
int numStudents = 0;
int getNextID() {
      return ++numStudents;
}
```

```
int getNextID() {
      static int numStudents = 0;
      return ++numStudents;
}
```

```
int main() {
   for (int ii = 0; ii < 5; ++ii)
      printf("CS22B0%02d\n", getNextID());
}
```

```
CS22B001
CS22B002
CS22B003
CS22B004
CS22B005
```

Scope and Lifetime

Variable	Scope	Lifetime
auto (local variables)	Function / Block	Function / Block
global	Global	100%
static local	Function / Block	100%
static global	File	100%
Dynamically allocated (malloc)	Global (via pointers)	Until deallocated (free) or 100%

Source: scopelife.c

```
gcc scopelife.c scopelife2.c
```

Argumentative Functions

```
int hasName(char message[N], char name[M]) {
        if (strstr(message, name))
                return 1;
        return 0;
}
```

```
int isPrime(int num) {
        for (int ii = 2; ii < num; ++ii)
                if (num % ii == 0) return 0;
        return 1;
}
```

```
int nextPermutation(char s[]) {
        int ll = strlen(s);
        int ii, na = 0;

        for (ii = ll - 1; ii >= 0; --ii)
                if (s[ii] != 'a' + strlen(s) - 1) {
                        ++s[ii];
                        return 0;
                } else {
                        s[ii] = 'a';
                        ++na;
                }
        if (na == ll) return 1;
}
```

Write a function to find permutations of first N letters.

```
aaa
aab
aba
abb
baa
bab
bba
bbb
```

Let's Swap

```
void swap(int x, int y) {
        int temp = x;
        x = y;
        y = temp;
}
```

✗

```
void swap(int x, int y) {
        x = x + y;
        y = x – y;
        x = x – y;
}
```

✗

```
void swap(int x, int y) {
        x = x ^ y;
        y = x ^ y;
        x = x ^ y;
}
```

✗

```
void swap(int *x, int *y) {
        int temp = *x;
        *x = *y;
        *y = temp;
}
```

✓

```
void swap(int *x, int *y) {
        *x = *x + *y;
        *y = *x – *y;
        *x = *x – *y;
}
```

✓

```
void swap(int *x, int *y) {
        *x = *x ^ *y;
        *y = *x ^ *y;
        *x = *x ^ *y;
}
```

✓

```
int main() {
        int x = 1, y = 2;
        swap(&x, &y);
        printf(“%d %d\n“, x, y);
}
```

Recursion

- A function can call itself.
- A function f1 can call f2, && f2 can call f1.
  - Recursion creates cycles.
- Useful to model certain computations naturally.
- Has similarity with induction.

Drawing hands, by M C Escher
Photo courtesy: meer.com

Do you find anything interesting when you google for recursion?

Recursive Printing

```
void rPrint(int x) {
        if (x == 0) return;
        else {
                printf(“%d\n“, x);
                rPrint(--x);
        }
}
// call as rPrint(10);
```

```
void rPrint(int x) {
        if (x == 0) return;
        else {
                printf(“%d\n“, x);
                rPrint(x - 1);
        }
}
// call as rPrint(10);
```

```
void rPrint(int x) {
        if (x > 0) {
                printf(“%d\n“, x);
                rPrint(x - 1);
        }
}
// call as rPrint(10);
```

```
void rPrint(int s, int e) {
        if (s <= e) {
                printf(“%d\n“, s);
                rPrint(s + 1, e);
        }
}
// call as rPrint(1, 10);
```

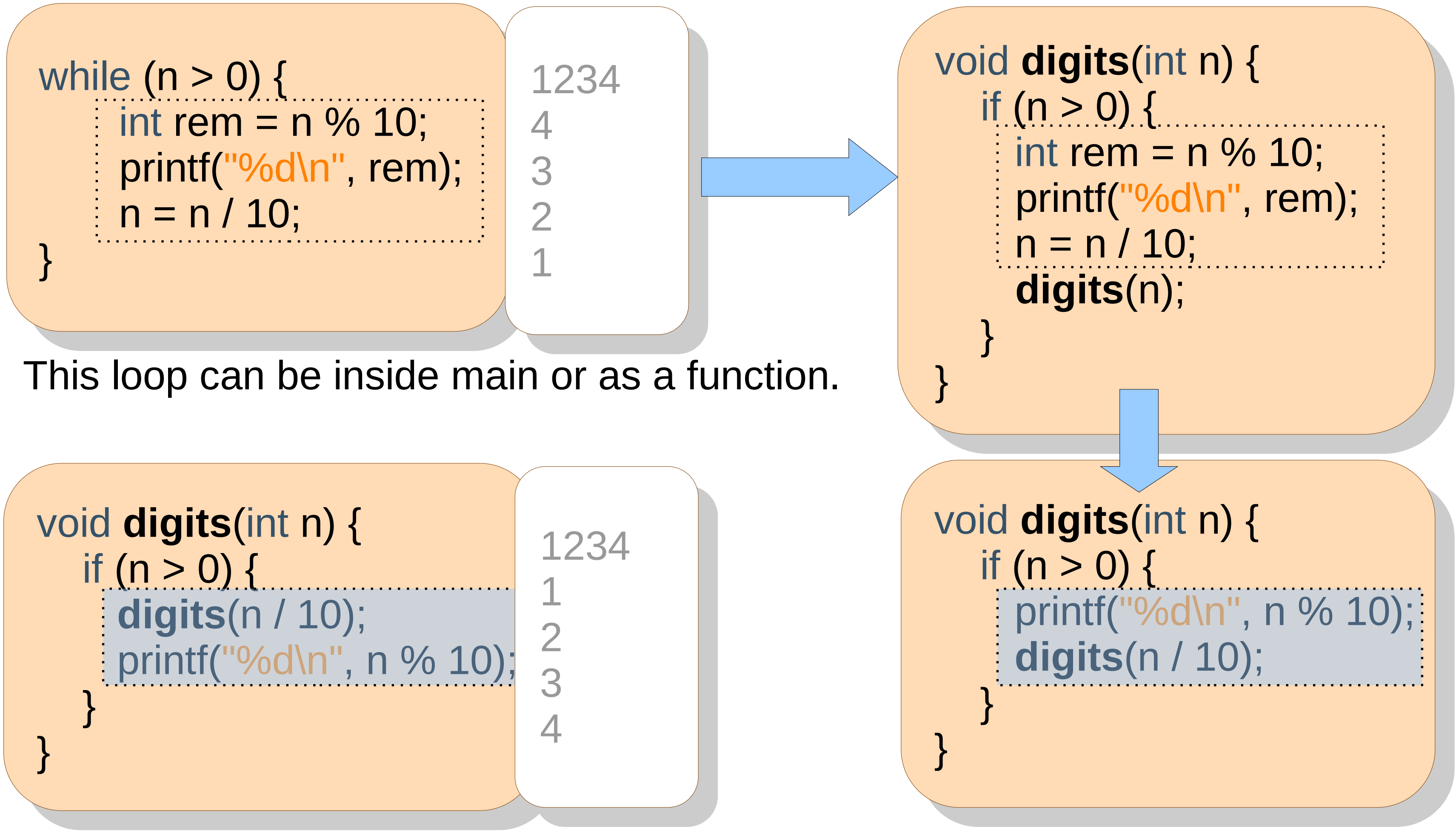
```
void rPrint(int x) {
        if (x > 0) {
                printf(“%d\n“, (11 - x));
                rPrint(x - 1);
        }
}
// call as rPrint(10);
```

```
void rPrint(int x) {
        if (x > 0) {
                printf(“%d\n“, x);
                rPrint(x - 1);
        }
}
// call as rPrint(10);
```

rPrint(1..10) = printf(1); **rPrint**(2..10);

Printing Digits

- Recall printing of digits of a number n.



```
while (n > 0) {
    int rem = n % 10;
    printf("%d\n", rem);
    n = n / 10;
}
```

```
1234
4
3
2
1
```

This loop can be inside main or as a function.

```
void digits(int n) {
    if (n > 0) {
        int rem = n % 10;
        printf("%d\n", rem);
        n = n / 10;
        digits(n);
    }
}
```

```
void digits(int n) {
    if (n > 0) {
        printf("%d\n", n % 10);
        digits(n / 10);
    }
}
```

```
void digits(int n) {
    if (n > 0) {
        digits(n / 10);
        printf("%d\n", n % 10);
    }
}
```

```
1234
1
2
3
4
```

Can we make a slight modification to print the digits in left-to-right order?

Problem: Problems

Write recursive codes for

- **Finding factorial(n)**

  fact(n) = n*fact(n-1),   fact(0) = 1
- **Finding the maximum in an array**

  max(a, 0, N-1) = greater(a[0], max(a, 1, N-1)),

  max(a, k, k) = a[k]
- **Reversing a string (in another string)**

  rev(s, N) = concat(s[N-1], rev(s, N-1))

  rev(s, 0, N-1) = concat(rev(s, 1, N-1), s[0])
- **Binary search**

Hemachandra / Fibonacci Numbers

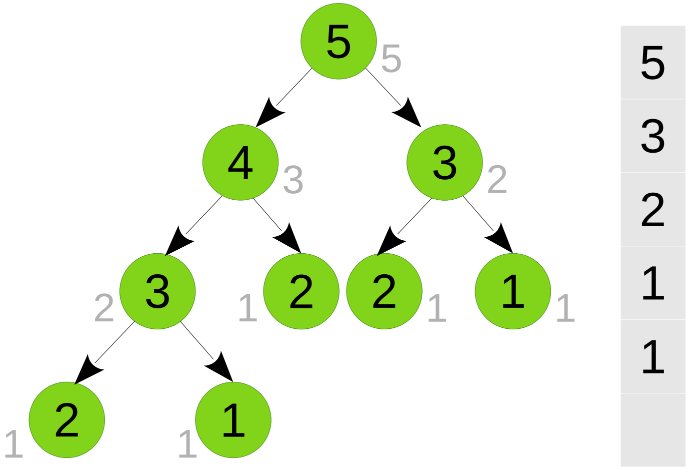
- fib(n) = fib(n – 1) + fib(n – 2)
- fib(1) = 1, fib(2) = 1

Repeated processing
How to reuse the computation?

fib (45): 3139ms
fib2(45): 0ms

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

```
#define INITVAL 0
int fib2recursive(int n, int arr[]) {
        if (arr[n] != INITVAL) return arr[n];
        arr[n] =  fib2recursive(n-1, arr) +
                  fib2recursive(n-2, arr);
        return arr[n];
}
int fib2(int n) {      // only driver, not a recursive fun
        int arr[n+1];  // not using arr[0]
        // init arr
        for (int ii = 1; ii <= n; ++ii)
                arr[ii] = INITVAL;
        arr[1] = arr[2] = 1;        // base case
        // call recursive function
        return fib2recursive(n, arr);
}
```



Problem: ~~Shaving~~ Halting

- A barber shaves all the people in his village who do not shave themselves.

  Does the barber shave himself?
- This statement is wrong.
- The below statement is true.

  The above statement is false.
- These are limitations to our logic – and hence to our computation.
  - We cannot write a program to find out if an arbitrary program is in an infinite loop.

Pass by Value

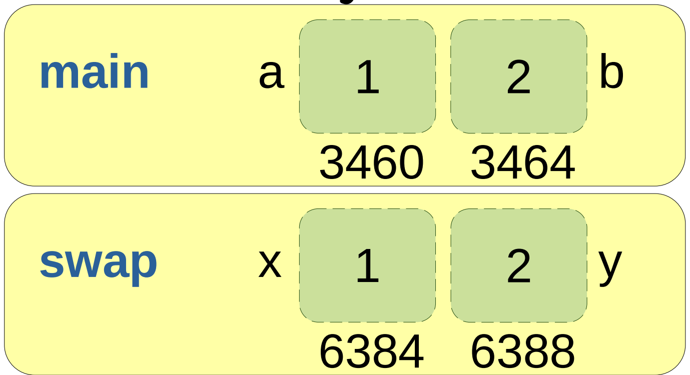
- **printf**("%d", x): pass by value
- **scanf**("%d", &x): pass by?

```
void swap(int x, int y) {
        int temp = x;
        x = y;
        y = temp;
}
// call as swap(a, b);
// does not work.
```

```
void swap(int *x int *y) {
        int temp = *x;
        *x = *y;
        *y = temp;
}
// call as swap(&a, &b);
// works
```

```
void swap(int &x, int &y) {
        int temp = x;
        x = y;
        y = temp;
}
// call as swap(a, b);
// works
```

Pass by value



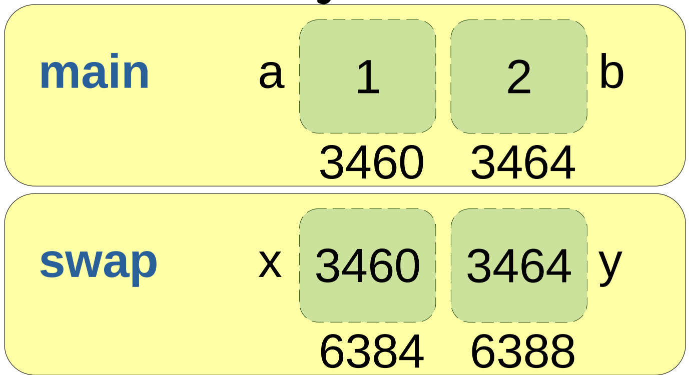
main a 1 2 b

3460 3464

swap x 1 2 y

6384 6388

Pass by value



main a 1 2 b

3460 3464

swap x 3460 3464 y

6384 6388

Pass by reference
Supported in C++

Passing an Array

- We can pass an array as a parameter.
  - fun(arr);
- The array is not copied.
- Array name is treated similar to a pointer.
  - void fun(int a[]) is same as void fun(int *a).
  - fun(arr) == fun(&arr) == fun(&arr[0]) == fun(arr + 0)
- Thus, array elements can be modified in the function.

```
read(arr, N);
int m = max(arr, N);
// works.
```

Arguments to main

- *main* has three forms

```
int main() {
    ...
}
```

```
int main(int a, char *b[]) {
    ...
}
```

Number of command-line arguments

Environment variables

```
int main(int a, char *b[], char *c[]) {

}
```

```
./a.out one 2 three -E
Number of cmdline args = 5
        0: ./a.out
        1: one
        2: 2
        3: three
        4: -E
0: SHELL=/bin/bash
1: SESSION_MANAGER=...
2: QT_ACCESSIBILITY=1
3: COLORTERM=truecolor
...
```

```
int main(int a, char *b[], char *c[]) {
        printf("Number of cmdline args = %d\n", a);
        for (int ii = 0; ii < a; ++ii)
                printf("\t%d: %s\n", ii, b[ii]);
        int jj = 0;
        while (c[jj] != NULL) {
                printf("%d: %s\n", jj, c[jj]);
                ++jj;
        }
}
```

Command-lineofy our mini-calculator.

```
./minic  2  -  5
./minic  5 '*' 7
```

Tip: *atoi* function converts a string into the corresponding integer.

Debugging Tips

- If segfault, guard all array accesses.
  - if (index < N) ... a[index] ...
- If segfault, guard pointer dereferences.
  - if (ptr != NULL) ... *ptr ...
- If infinite loop, add a getchar().
- If infinite loop, check that at least one variable in the condition is getting modified in the loop.
- Compile often, test often, add printfs to check <u>intermediate</u> values.

Tic-Tac-Toe

Design

- 2D array of size 3x3
- Inputs
  - Place of the symbol (1..3, 1..3)
  - ~~Two symbols * and o~~ (two players alternate)
- Output
  - Show the board at every step
  - Declare winner or draw
- Checks
  - Invalid input (e.g., place * at (0, 0) or an occupied cell)
  - Winning placement (vertical, horizontal, diagonal)
  - Optional: Game will be a draw.

All C Keywords

auto	break	case	char
const	continue	default	do
double	else	enum	extern
float	for	goto	if
int	long	register	return
short	signed	sizeof	static
struct	switch	typedef	union
unsigned	void	volatile	while

	Week	Problems	Tools
✓	0	Solve equations, find weighted sum.	Data types, expressions, assignments
✓	1	Find max, convert marks to grade.	Conditionals, logical expressions
✓	2	Find weighted sum for all students.	Loops
✓	3	Encrypt and decrypt a secret message.	Character arrays
✓	4	Our first game: Tic-tac-toe	2D arrays
✓	5	Making game modular, reuse.	Functions
✓	6	Find Hemachandra/Fibonacci numbers.	Recursion
	7	Encrypt and decrypt many messages.	Dynamic memory, pointers
	8	Maintain student records.	Aggregate data types
	9	Search and sort student records.	Searching and sorting algorithms
	A	Reduce memory wastage.	Linked lists
	B	Implement token system in banks.	Queues
	C	IRCTC-like ticket booking system	File handling
	D	Putting it all together	All the above