CME 112- Programming Languages II

Week 2

Memory Layout of C Programs and Recursive Functions

Assist. Prof. Dr. Caner Özcan

We are here to make our own journey; It is not easy to find the courage to be our own on this journey, but without ourselves, nothing in our lives can be understood. ~D. Cüceloğlu

Storage Classes

► Four storage classes are automatics, external, register, and static with corresponding.

- **auto:** Variables declared within function are automatic by default. These variables can be used in scope of the function. auto double x, y;

  They're stored in the **Stack**.

  Global variables and parameter variables cannot take auto property.

- **extern:** One methods of transmitting information across blocks and functions is to use external variables.

  If the variable defined by extern is not given the initial value, the compiler does not allocate space in memory.

Examples: auto & extern

- This use of extern is used to tell the compiler to ‘’look for it elsewhere’’ either in this file or in some other file.

example1.c

```
#include <stdio.h>

int a = 1, b = 2;
c = 3;
int f(void);
int main(void) {

        printf("%3d\n", f());
        printf("%3d%3d%3d\n", a, b, c);

        return 0;
}
```

file2.c

```
int f(void) {
        extern a;
        int b, c;
        b = c = a;
        return (a + b + c);
}
```

Storage Classes

- **register:** Storage class register tells the compiler that the association variables should be stored in high-speed memory registers.

  Specifies that the variable is kept in the registers of the CPU, not in memory.

- **static:** Local variables defined in functions.

  After the function ends, the variable value is stored.

  Only valid in the function they are defined.

  Static local and global variables are kept in the data segment region.

Storage Classes

Storage classes in C

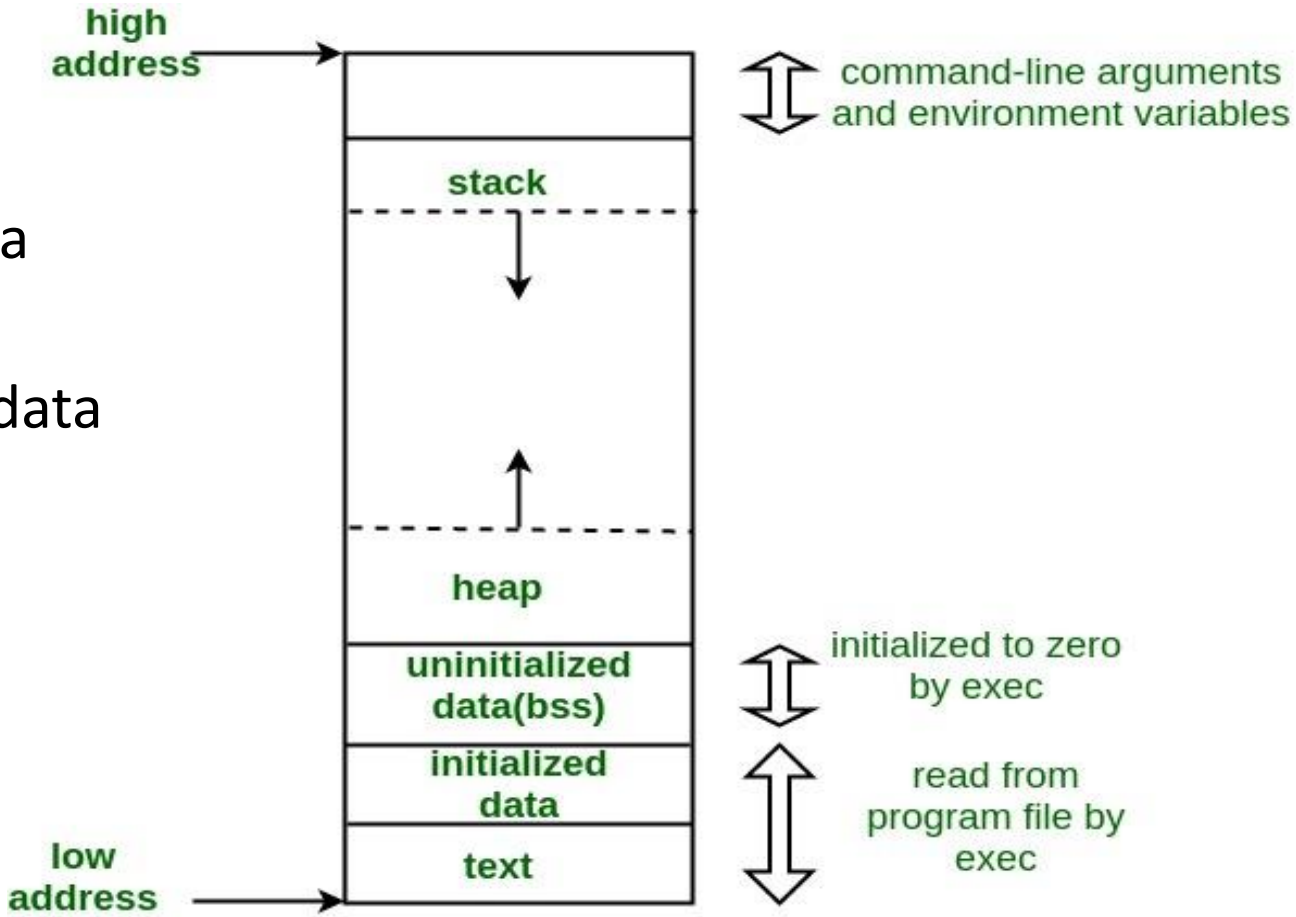
Storage Specifier	Storage	Initial value	Scope	Life
auto	stack	Garbage	Within block	End of block
extern	Data segment	Zero	global Multiple files	Till end of program
static	Data segment	Zero	Within block	Till end of program
register	CPU Register	Garbage	Within block	End of block

https://www.geeksforgeeks.org/storage-classes-in-c/

Memory Layout of C Programs

- A typical memory representation of C program consists of following sections.

1. Text segment
2. Initialized data segment
3. Uninitialized data segment
4. Stack
5. Heap



Memory Layout of C Programs

1. Text Segment:

- A text segment , also known as a code segment or simply as text, is one of the sections of a program in an object file or in memory, which contains executable instructions.
- Usually, the text segment is sharable so that only a single copy needs to be in memory for frequently executed programs, such as text editors, the C compiler, the shells, and so on.
- Also, the text segment is often read-only, to prevent a program from accidentally modifying its instructions.

Memory Layout of C Programs

2. Initialized Data Segment :

- A data segment is a portion of virtual address space of a program, which contains the global variables and static variables that are initialized by the programmer.

3. Uninitialized Data Segment:

- Data in this segment is initialized by the kernel to arithmetic ‘0’ (zero) before the program starts executing uninitialized data starts at the end of the data segment and contains all global variables and static variables that are initialized to zero or do not have explicit initialization in source code.

Memory Layout of C Programs

4. Stack:

- Stack, where automatic variables are stored, along with information that is saved each time a function is called.
- Each time a function is called, the address of where to return to and certain information about the caller's environment are saved on the stack.
- The newly called function then allocates room on the stack for its automatic and temporary variables.
- This is how recursive functions in C can work.
- Each time a recursive function calls itself, a new stack frame is used, so one set of variables doesn't interfere with the variables from another instance of the function.

Memory Layout of C Programs

5. Heap :

- Heap is the segment where dynamic memory allocation usually takes place.
- Heap area is managed by malloc, realloc, and free.

```c
#include <stdio.h>

int global; /* Uninitialized variable stored in bss*/
int main(void)
{
    int *ptr_one;
    ptr_one = (int *)malloc(sizeof(int)); /* memory allocating in heap segment */
    int c;//local variable stored in stack
    static int i = 100; /* Initialized static variable stored in DS*/
    static int k; /* Initialized static variable stored in bss*/
    return 0;
}
```

Creating Large Programs

- Typically, a large program is written in a separate directory as a collection of .h and .c file, with each .c file contains one or more functions definition
- When the preprocessor program receives the #include <"filename"> directive, it searches for the file in the same folder or in the system-dependent places.
- If it cannot be found, preprocessor issues an error message and compilations stops.
- Files with .h extension can include #include, #define directives, struct structures, function prototypes.

Creating Large Programs


```
#include "pgm.h"

int main(void)
{
        int i;
        for (i = 0; i < N;i++)
                f2();

        return 0;
}
```

program.c

```
#include <stdio.h>
#define N 5

void f2(void)
{
        printf("Hello from f2()\n");
}
```

pgm.h

Recursion

- Self-calling functions.
- If a function is called with a base case it returns the result.
- If a function is called with a more complex problem, the function divides the problem into two conceptual pieces;
  - First: a piece that function know how to do
  - Second: a piece that function does not know how to do
    - The second part must resemble the original problem
    - The function launches a new copy of itself (recursion step) to solve what it does not know how to do
- Eventually base case gets solved.

Recursion

- Program that prints numbers from 1 to N on the screen with recursion.

```c
#include <stdio.h>
int f(int n)
{
    if (n == 0)
        return 0;
    f(n - 1);
    printf("%d\n", n);
}
int main(void)
{
    int num = 10;
    f(num);
    return 0;
}
```

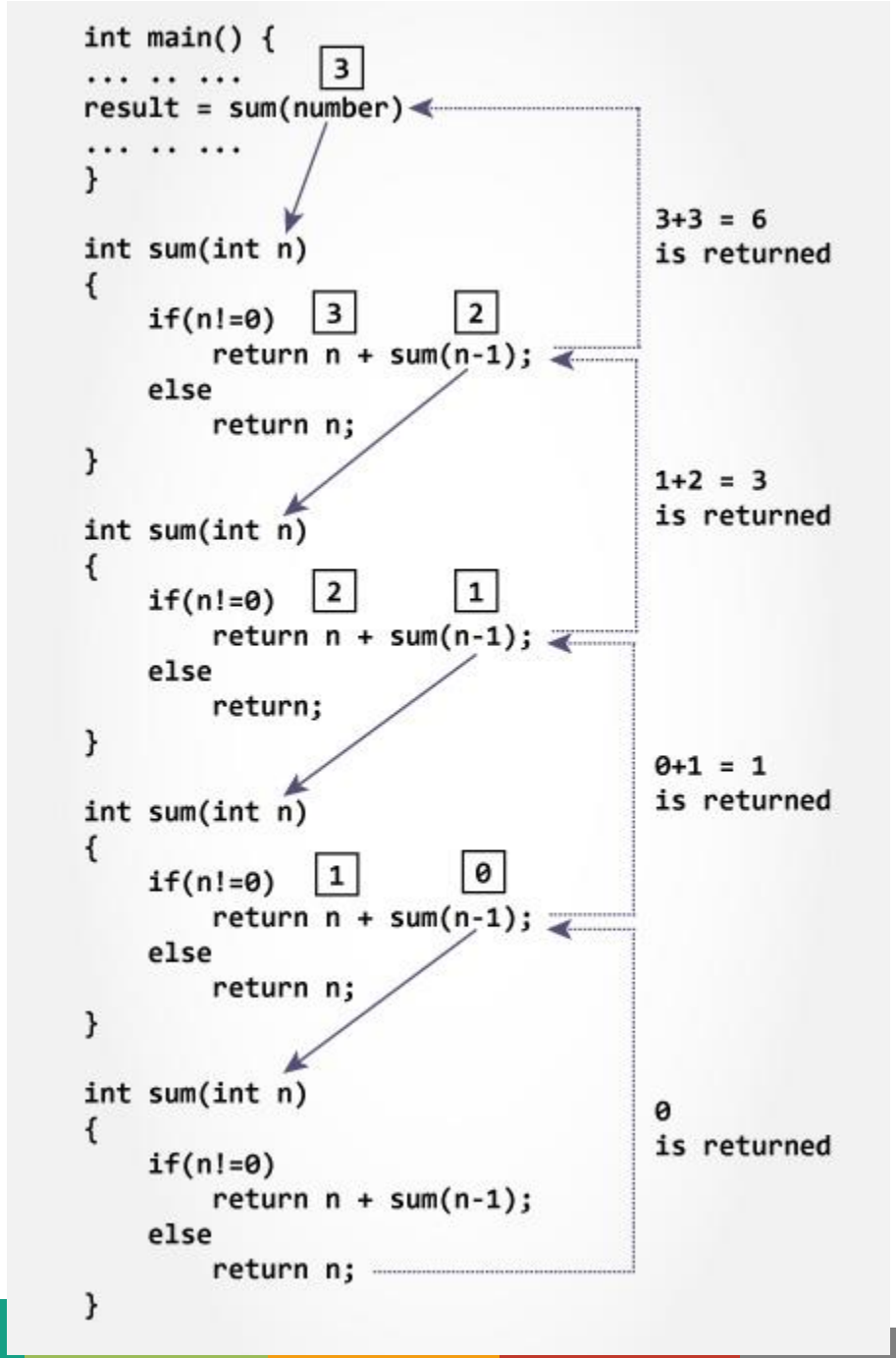
Recursion

- A recursive function that finds the sum of the numbers from 1 to N.

```
#include <stdio.h>
int sum(int n)
{
   if (n == 1)
      return n;
   else
      return (n + sum(n - 1));
}

int main(void)
{
   int num = 10;
   printf("Result = %d", sum(num));
   return 0;
}
```

Recursion


int main() {
... .
result = sum(number)
... .
}
3
int sum(int n)
{
if(n!=0)
return n + sum(n-1);
else
return n;
}
3
2
int sum(int n)
{
if(n!=0)
return n + sum(n-1);
else
return;
}
2
1
int sum(int n)
{
if(n!=0)
return n + sum(n-1);
else
return n;
}
1
0
int sum(int n)
{
if(n!=0)
return n + sum(n-1);
else
return n;
}
3+3 = 6
is returned
1+2 = 3
is returned
0+1 = 1
is returned
0
is returned

Recursion

- Program that prints the multiplication table as a recursive.

```c
#include <stdio.h>
int table(int x) {
   int i;
   if (x <= 10) {
      for (i = 1; i<11; i++)
         printf("%-3d", x*i);
      printf("\n");
      return table(x + 1);
   }
   else return 1;
}

int main(void){
   int x = 1;
   table(x);
   return 0;
}
```

Recursion

- A recursive definition of the factorial function is arrived at by observing the following definition.

  n! = n. (n-1)!

- Example: factorial
  - 5! = 5. 4. 3. 2. 1
  - Notice that

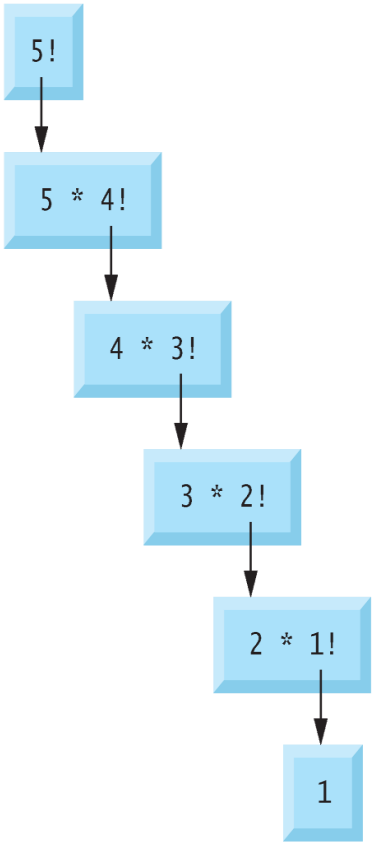
    5! = 5. 4!

    4! = 4. 3!...
  - Can compute factorials recursively
  - Solve base case (1! = 0! =1) then

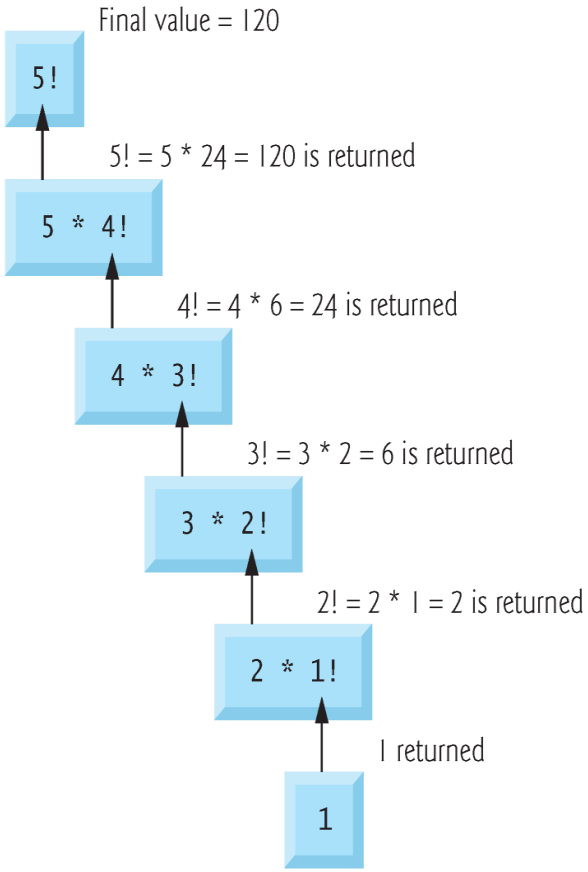
    2! = 2. 1! = 2*1 = 2

    3! = 3. 2! = 3*2 = 6

Recursion



(a) Sequence of recursive calls.



(b) Values returned from each recursive call.

Recursion

- Recursion program to calculate and print the factorials of integers 0-10.

```c
#include <stdio.h>
long faktorial(long n){
   if (n <= 1)
      return 1;
   else
      return (n*faktorial(n - 1));
}
int main(void){
   int i;
   for (i = 0; i <= 10; i++) {
      printf("%d! = %d\n", i, faktorial(i));
   }
   return 0;
}
```

Recursion Fibonacci Numbers

- Fibonacci series: 0, 1, 1, 2, 3, 5, 8...
- Each number is the sum of the previous two.
- Base case:

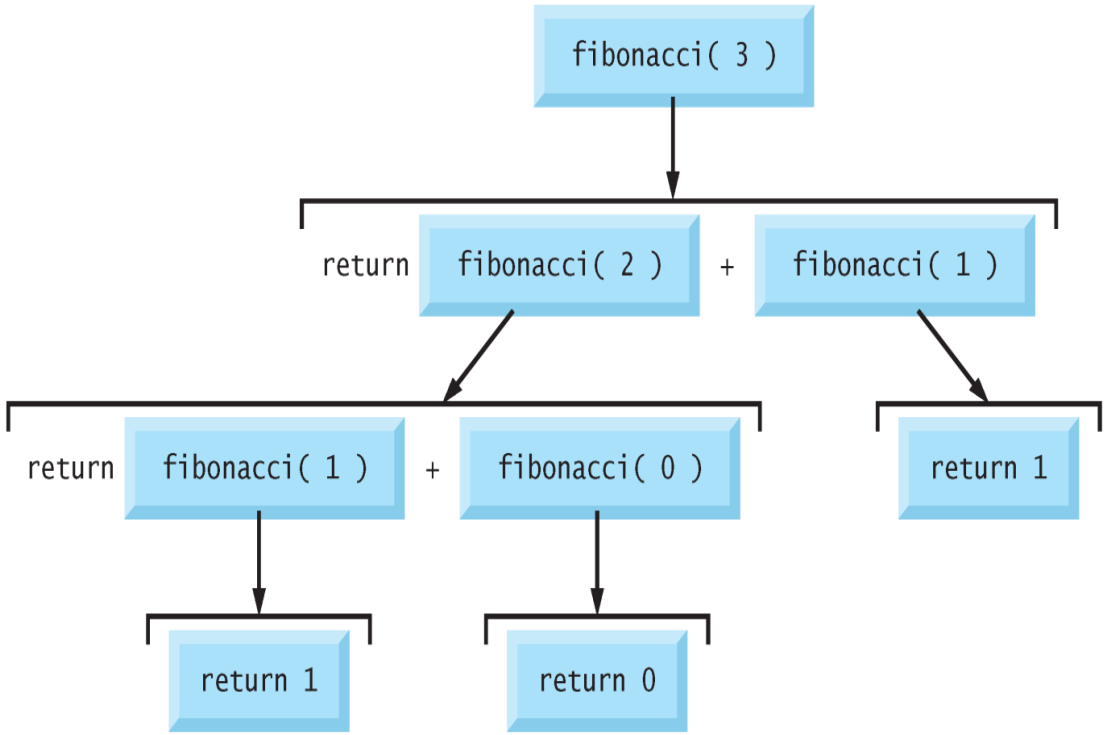
  Fib(0) = 0
  Fib(1) = 1

- Can be solved recursively:

  Fib (n) = Fib(n-1) + Fib(n-2)

Recursion Fibonacci Numbers

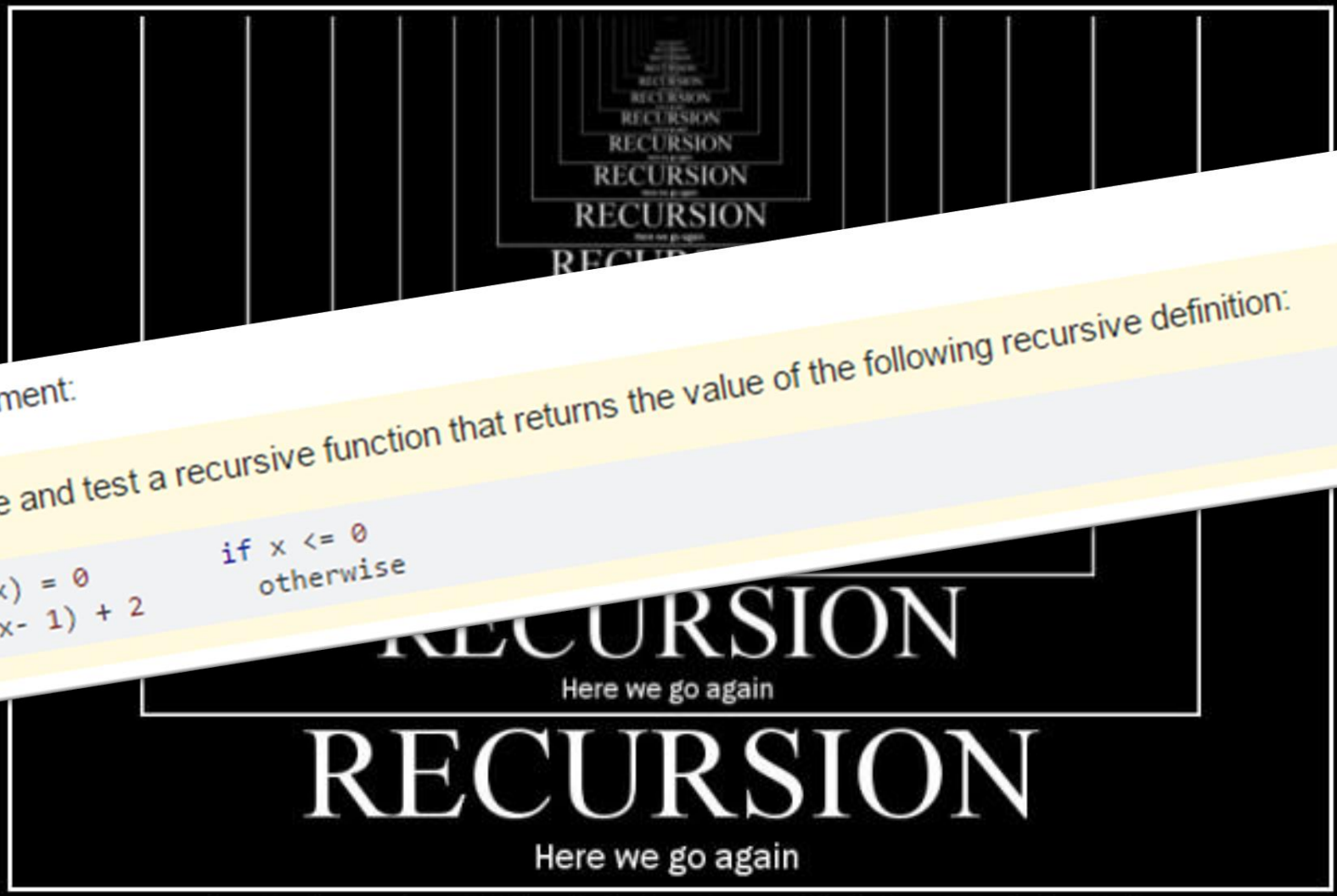
- Figure shows how function Fibonacci would evaluate fibonacci(3)



Recursion Fibonacci Numbers

- Calculating n^{th} Fibonacci number recursively.

```
#include <stdio.h>
long fibonacci(long n){
   if (n == 0 || n == 1)
      return n;
   else
      return fibonacci(n - 1) + fibonacci(n - 2);
}
int main(void){
   long i, n;
   printf("How many fibonacci numbers?:");
   scanf("%d", &n);
   for (i = 1; i <= n; i++){
      printf("Number %d: %ld\n", i, fibonacci(i));
   }
   return 0;
}
```



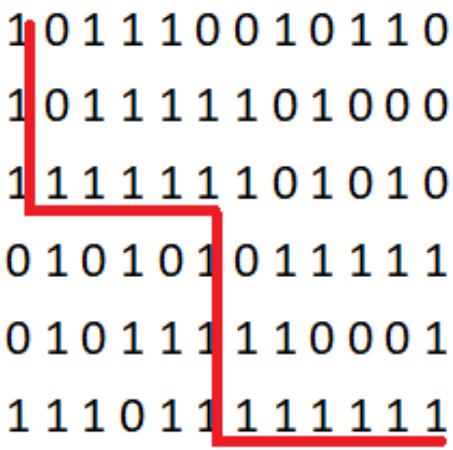
Assignment:

Write and test a recursive function that returns the value of the following recursive definition:

```
f(x) = 0          if x <= 0
f(x- 1) + 2         otherwise
```

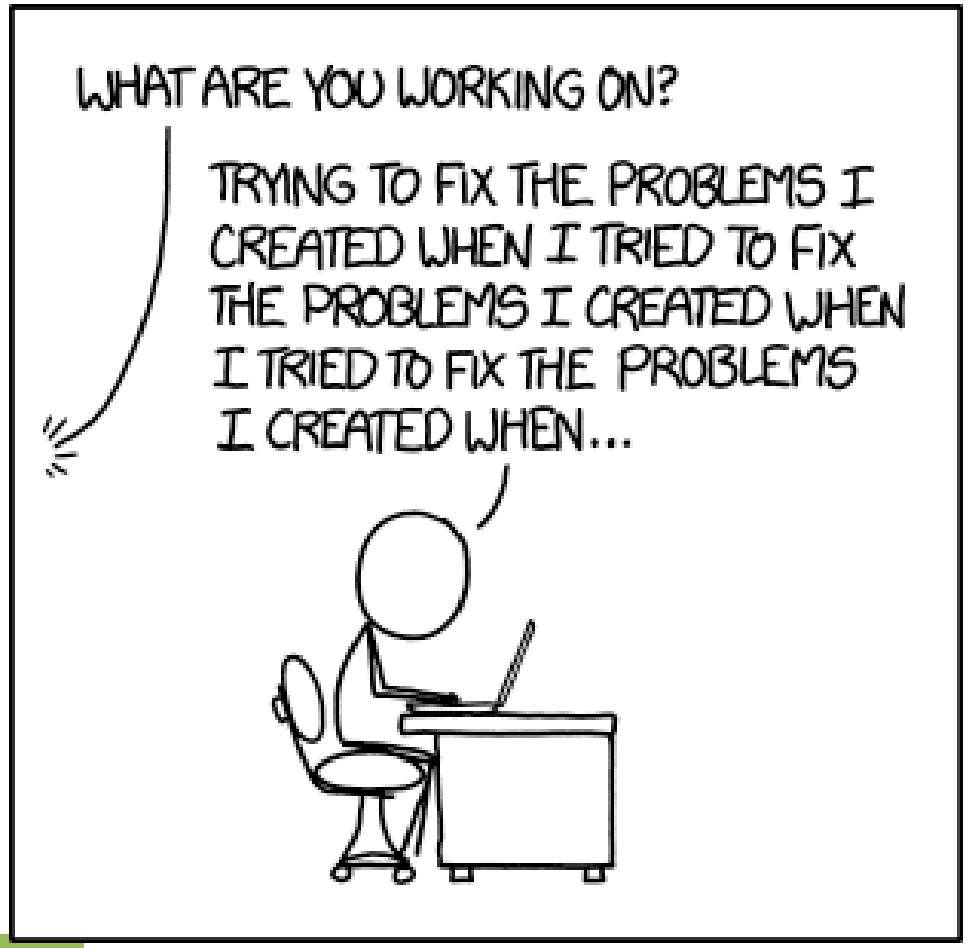
Recursive Shortest Path Finding

- A matrix of size mxn consists of values 1 and 0. Write the C program that finds the shortest path from 0x0 to mxn point. One (1) values in matrix means path, zero (0) means wall. So you cannot go to the coordinate or position with zero value. You can move in all four directions (up, down, left and right). An example path is given below.



Next Week

- Examples with recursive functions



References

- Doç. Dr. Fahri Vatansever, “Algoritma Geliştirme ve Programlamaya Giriş”, Seçkin Yayıncılık, 12. Baskı, 2015.
- Kaan Aslan, “A’dan Z’ye C Klavuzu 8. Basım”, Pusula Yayıncılık, 2002.
- Paul J. Deitel, “C How to Program”, Harvey Deitel.
- “A book on C”, All Kelley, İra Pohl

Thanks for listening

CANER ÖZCAN
canerozcan@karabuk.edu.tr

Footnote..

It is difficult to find a black cat in a dark room, especially if there are no cats in the room. This phrase describes the progress of science. It says that science is much different from what is described in TV, newspapers, internet news and high school curriculum.