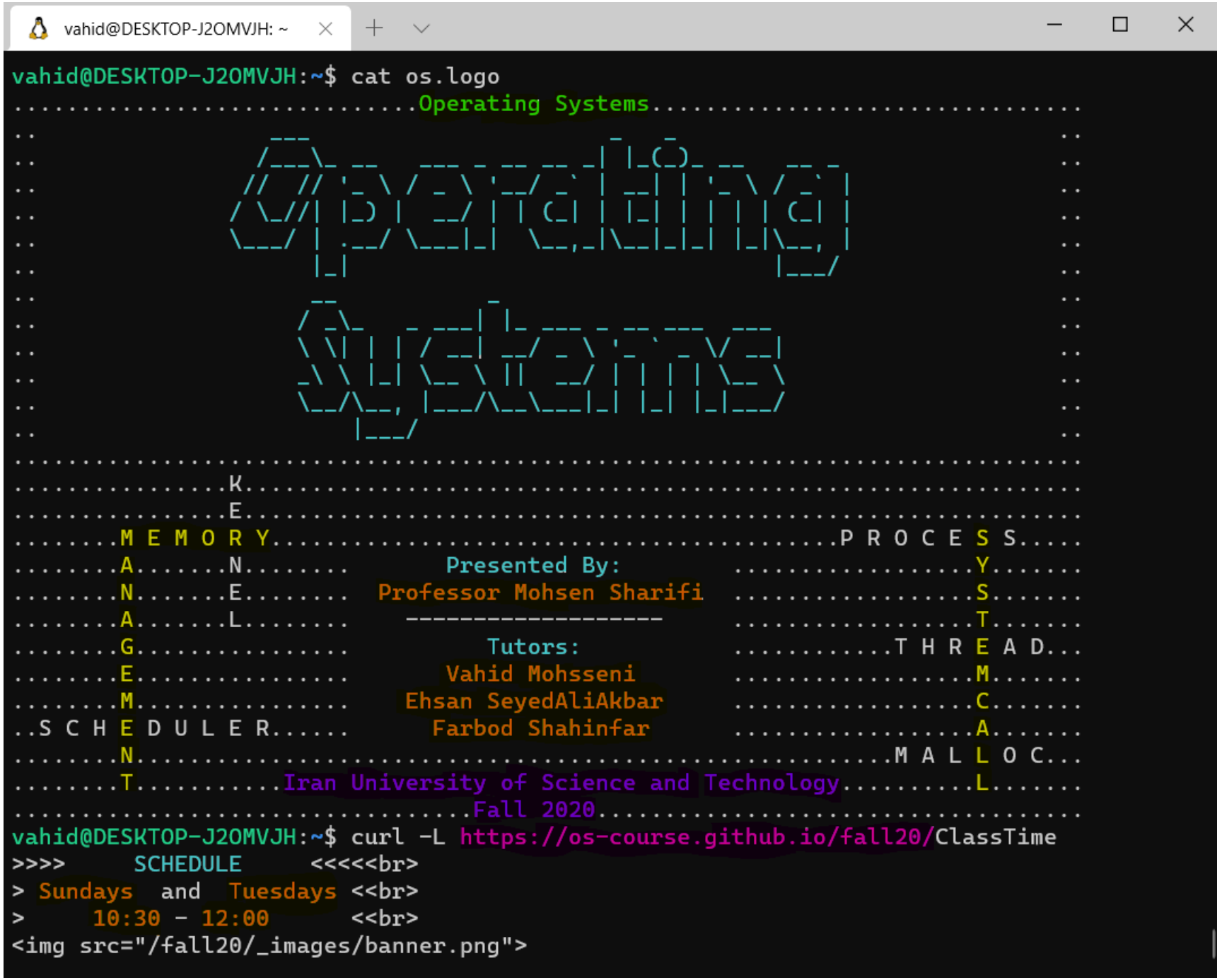
Operating Systems

Quick Introduction to C Programming Language

Fall 2020


vahid@DESKTOP-J2OMVJH: ~
vahid@DESKTOP-J2OMVJH:~$ cat os.logo
Operating Systems
KERNEL
MEMORY MANAGEMENT
SCHEDULER
PROCESSYS
THREAD
MALLOC
Presented By:
Professor Mohsen Sharifi
Tutors:
Vahid Mohsseni
Ehsan SeyedAliAkbar
Farbod Shahinfar
Iran University of Science and Technology
Fall 2020
vahid@DESKTOP-J2OMVJH:~$ curl -L https://os-course.github.io/fall20/ClassTime
>>>> SCHEDULE <<<<
> Sundays and Tuesdays <
> 10:30 - 12:00 <
<img src="/fall20/_images/banner.png">

Agenda

- Brief History
- Structure of a C Program
- Data Types
- Operators
- Array
- Flow Control

Brief History of C Programming Language

History

- The C programming language was devised in the early 1970s as a system implementation language for the nascent Unix operating system.
- The C programming language was created with the purpose of writing an operating system with a high level language.

History

- The C programming language was devised in the early 1970s as a system implementation language for the nascent Unix operating system.
- The C programming language was created with the purpose of writing an operating system with a high level language.

History

- C was influenced by B programming language.
- B programming language was the developed by Ken Thompson

Ken Thompson

Code written in B

```
main() {
    extrn putchar, n, v;
    auto i, c, col, a;

    i = col = 0;
    while(i<n)
        v[i++] = 1;
    while(col<2*n) {
        a = n+1 ;
        c = i = 0;
        while (i<n) {
            c =+ v[i] *10;
            v[i++]  = c%a;
            c =/ a--;
        }

        putchar(c+'0');
        if(!(++col%5))
            putchar(col%50?' ': '*n');
    }
    putchar('*n*n');
}
v[2000];
n 2000;
```

History: PDP-7

- Unix was developed for PDP-7 written in assembly by Ritchie and Thompson.

History: PDP-11

- Ritchie and Thompson decided to port UNIX on PDP-11
- UNIX for PDP-11 was also developed in assembly
- There was need for a programming language for developing utilities on the new platform

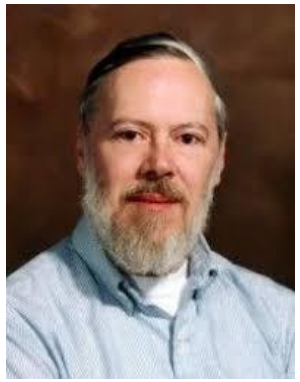
History: PDP-11

- Try to implement Fortran compiler
- Try to use BCPL which resulted in B
  - B was slow and not taking advantage of hardware capabilities.
- Dennis Ritchie created C (1972)

History

- Unix v2, had C compiler and related utility
- Unix v4 was re-implemented with C.
- Unix was one of the first operating system kernels to be implemented in a language other than assembly.

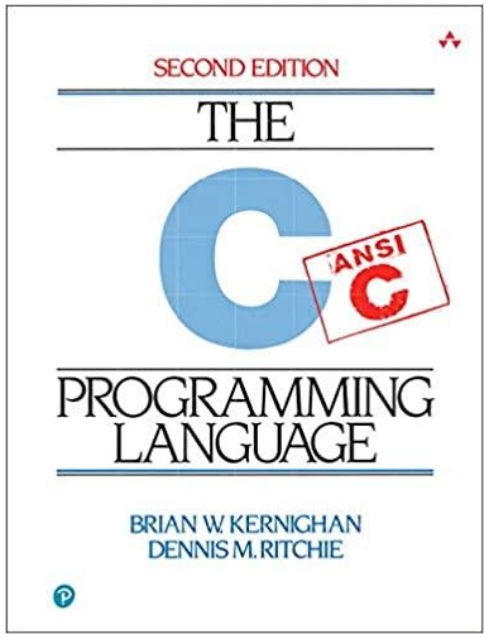
History: CD&K

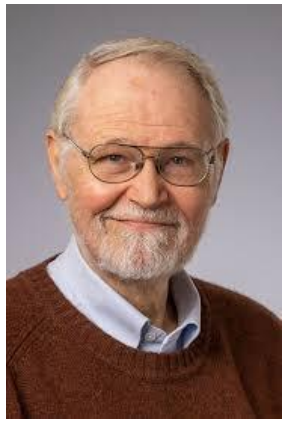


Dennis Ritchie

1941 – 2011

Known for:
ALTRAN
B
BCPL
C
Multics
Unix





Brian W. Kernighan

1942 – present

Known for:
Unix
AWK
Kernighan–Lin algorithm
The C programming book

History

- During 1970 and 1980 versions of C was implemented for different types of computers so there was a need for a standard definition.
- Since then ANSI and then ISO have voted on different C standards including: C89, C99, C11, C18

Compiled or Interpreted

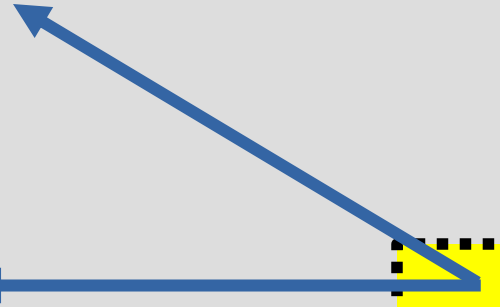
Structure of a C Program

Structure of a C Program

```c
/* adding standard input output header file to the
 * source code.
 * */
#include <stdio.h>
int main(int argc, char *argv[])
{
    // defining some variables
    int a;
    int b, c;
    a = 10;
    b = 20;
    c = a + b;

    // writing to stdout
    printf("hello world\n");
    printf("a + b = %d\n", c);

    return 0;
}
```

Structure of a C Program

```c
/* adding standard input output header file to the
 * source code.
 * */
#include <stdio.h>
int main(int argc, char *argv[])
{
    // defining some variables
    int a;
    int b, c;
    a = 10;
    b = 20;
    c = a + b;

    // writing to stdout
    printf("hello world\n");
    printf("a + b = %d\n", c);

    return 0;
}
```



Commenting, multiline and single line

Structure of a C Program

```
/* adding standard input output header file to the
 * source code.
 * */
#include <stdio.h>
int main(int argc, char *argv[])
{
    // defining some variables
    int a;
    int b, c;
    a = 10;
    b = 20;
    c = a + b;

    // writing to stdout
    printf("hello world\n");
    printf("a + b = %d\n", c);

    return 0;
}
```

Adding header file to the source code.

#include <….> // search in the systems directories
#include "……" // can have relative path

(more on the topic of header file in future.)

Structure of a C Program

```c
/* adding standard input output header file to the
 * source code.
 * */
#include <stdio.h>
int main(int argc, char *argv[])
{
    // defining some variables
    int a;
    int b, c;
    a = 10;
    b = 20;
    c = a + b;

    // writing to stdout
    printf("hello world\n");
    printf("a + b = %d\n", c);

    return 0;
}
```

By convention the program starts from the main function.

The main function can have two variables
Int argc and char *argv[].

With help of these variable you can access the parameters passed to the program with they are called.

Structure of a C Program

```c
/* adding standard input output header file to the
 * source code.
 * */
#include <stdio.h>
int main(int argc, char *argv[])
{
    // defining some variables
    int a;
    int b, c;
    a = 10;
    b = 20;
    c = a + b;

    // writing to stdout
    printf("hello world\n");
    printf("a + b = %d\n", c);

    return 0;
}
```

A block of code is defined by { } in the C programming language.

This block determines the scope of function, variables and other statements.

Structure of a C Program

```c
/* adding standard input output header file to the
 * source code.
 * */
#include <stdio.h>
int main(int argc, char *argv[])
{
    // defining some variables
    int a;
    int b, c;
    a = 10;
    b = 20;
    c = a + b;

    // writing to stdout
    printf("hello world\n");
    printf("a + b = %d\n", c);

    return 0;
}
```

Variable definition

```
<type> <variable name> [,<variable name>];
```

Data Types

Data Types

Name	Size (bytes)
[unsigned] char	1
[unsigned] short	2
[unsigned] int	4
[unsigned] long	8
[unsigned] long long	8
[unsigned] float	4
[unsigned] double	8

- Size of data types may vary depending on compiler and its configurations.
- No boolean type but:

  #include <stdboo.h>

  defines bool, true, and false.

Data Types

- Notice, in <stdint.h> there are some useful type definitions.
  - int8_t, int16_t, int32_t, int64_t
  - uint8_t, uint16_t, uint32_t, uin64_t
  - Link: https://www.gnu.org/software/libc/manual/html_node/Integers.html

Data Types

Signed

bit: 3 2 1 0

[0][0][0][0] = 0
[0][0][0][1] = 1
[0][0][1][0] = 2
[0][0][1][1] = 3
[0][1][0][0] = 4
.
.
.
[0][1][1][1] = 7
[1][0][0][0] = -8
[1][0][0][1] = -7
[1][0][1][0] = -6
[1][0][1][1] = -5
[1][1][0][0] = -4
.
.
.

Unsigned

bit: 3 2 1 0

[0][0][0][0] = 0
[0][0][0][1] = 1
[0][0][1][0] = 2
[0][0][1][1] = 3
[0][1][0][0] = 4
.
.
.
[0][1][1][1] = 7
[1][0][0][0] = 8
[1][0][0][1] = 9
[1][0][1][0] = 10
[1][0][1][1] = 11
[1][1][0][0] = 12
.
.
.

Operators

Operators

Type	Operators
Arithmetic	*, /, +, -, %, ++, --
Relational	==, !=, >, <, >=, <=
Logical	&&, ||, !
Bitwise	&, |, ^, ~, <<, >>
Assign	=, <arithmetic op>=, <bitwise op>=
Others	sizeof(), &, *, (condition) ? <value> : <value>

- Sizeof returns the number of bytes a data type requires.

Array

Array

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    int values[] = {1,2,3,4};
    // int values[10] = {1,2,3,4}; // what is the difference?
    printf("values[2]: %d\n", values[2]);

    Int arr2d[4][5] = {
        {1,2,3,4},
        {5,6,7,8},
        {9,0,1,2}
    };
      return 0;
}
```

- You can access values in the array by using its index counting from 0.
- Initialization will start from index 0 and assigns values.
- Unspecified indexes are set to 0.

Array

- Sizeof an array variable evaluates to the amount of memory array has acquired.

Array: Declaring an Array

```
{
    int i, j, intArray[ 10 ], number;
    float floatArray[ 1000 ];
    int tableArray[ 3 ][ 5 ];  /* 3 rows by 5 columns */

    const int NROWS = 100;
    const int NCOLS = 200;
    float matrix[ NROWS ][ NCOLS ];
}
```

Array: Initializing Array

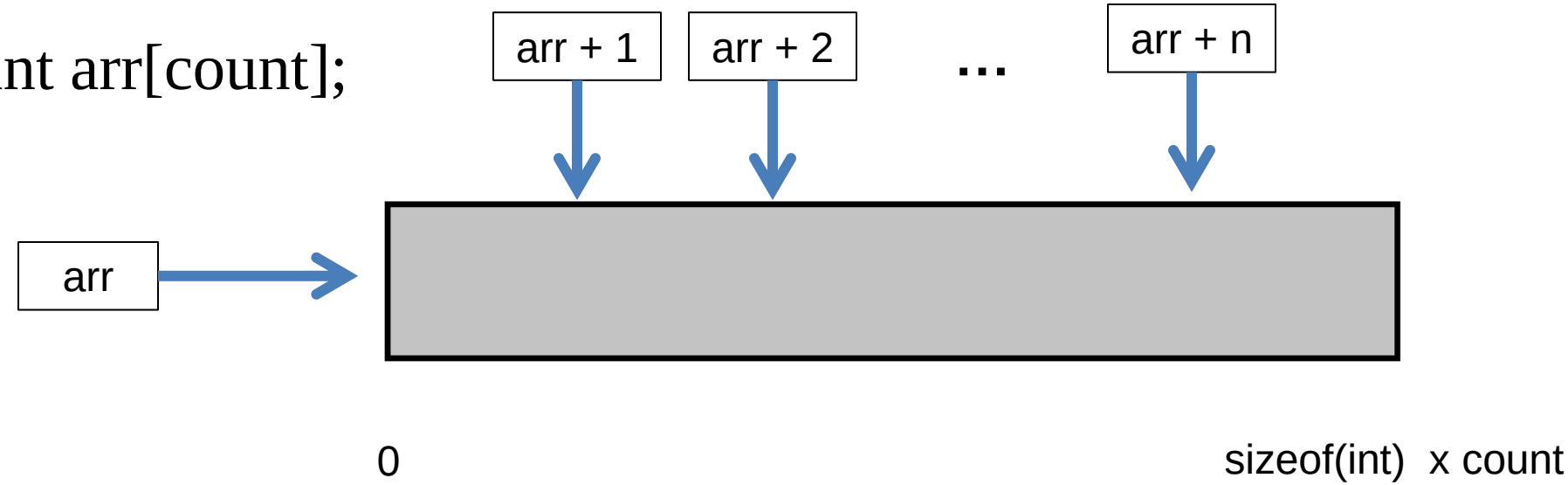
```
{
    int i =  5, intArray[ 6 ] = { 1, 2, 3, 4, 5, 6 }, k;
    float sum  = 0.0f,
                floatArray[ 100 ] = { 1.0f, 5.0f, 20.0f };
    double  piFractions[ ] = {3.141592654,
                                1.570796327, 0.785398163};

    int numbers[100] = {1, 2, 3, [10] = 10, 11, 12,
                                [60] = 50, [42] = 420 };
};
```

Array

int arr[count];

Array

int arr[count];

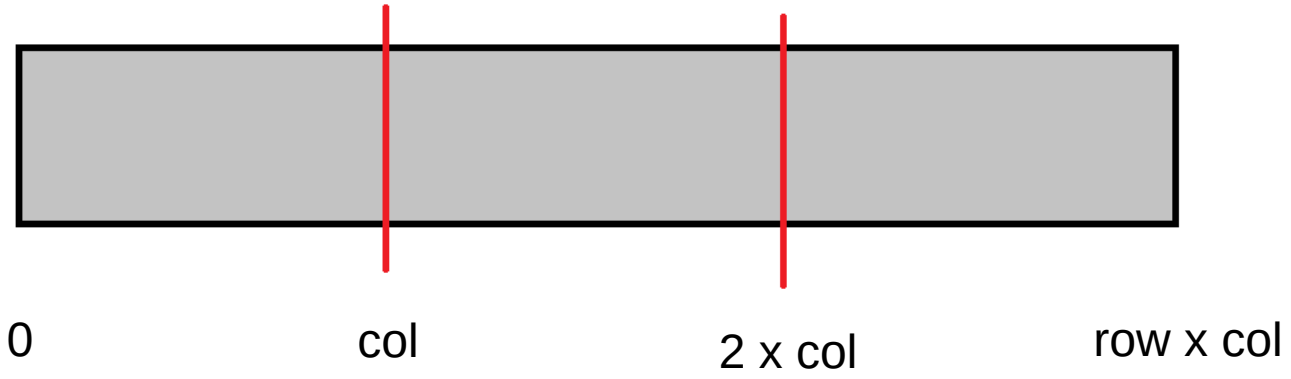


Array: 2d Array memory layout

char mat[row][col]; // row = 3, col = 5

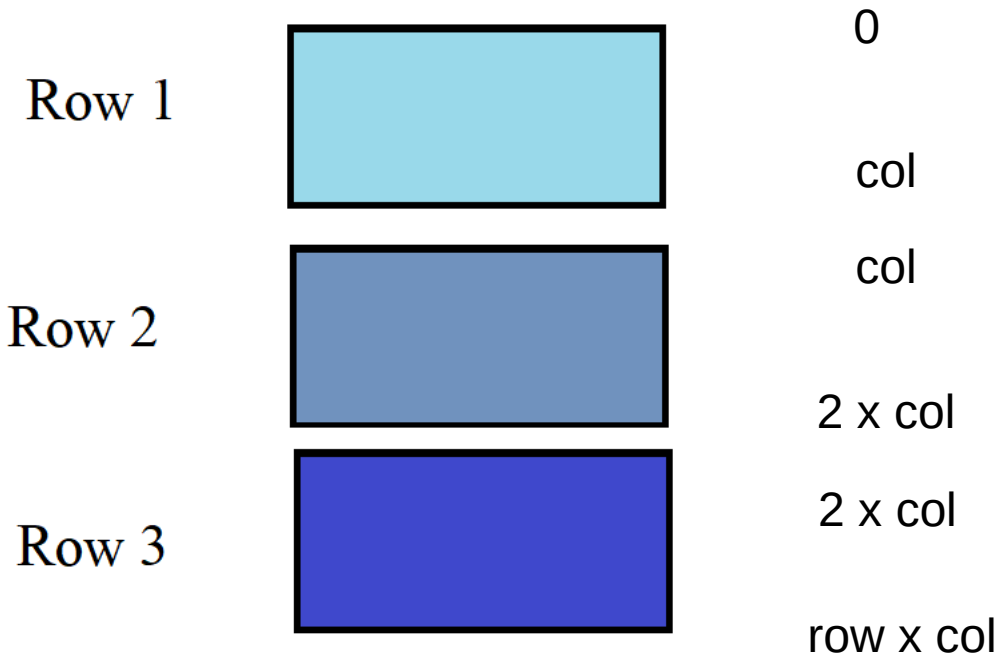
Array: 2d Array memory layout

char mat[row][col]; // row = 3, col = 5



Array: 2d Array memory layout

char mat[row][col]; // row = 3, col = 5



Array: 2d Array memory layout

char mat[row][col]; // row = 3, col = 5

```
row 0: 0x...020 0x...021 0x...022 0x...023 0x...024
row 1: 0x...025 0x...026 0x...027 0x...028 0x...029
row 2: 0x...02a 0x...02b 0x...02c 0x...02d 0x...02e
```

Flow Control

Flow Control: If Statements

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    int temperature;
    scanf("%d\n", &temperature);
    if (temperature < 23) {
        turn_on_heater();
    } else if (temperature < 26) {
        turn_off_heater();
        turn_off_cooler();
    } else if (temperature < 40) {
        if (is_heater_enable()) {
            turn_off_heater();
        }
        turn_on_cooler();
    }
    return 0;
}
```

Flow Control: While Statements

```
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[])
{
      unsigned char condition = 1;
      char cmd[32];
      while (condition) {
            fgets(cmd, 32, stdin);
            if (strcmp(cmd, "quit\n") == 0) {
                  condition = 0;
            }
            /* execute the command and perform
             * related operations.
             * ...
             */
      }
      return 0;
}
```

Flow Control: Do-While Statements

```c
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[])
{
     unsigned char condition = 1;
     do {
          /* do operations and related logic
           * ...
           */
     } while (condition);
     return 0;
}
```

Flow Control: For loop

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    int count = 8;

    for (int i = 0; i < count; i++) {
        // ...
    }
    return 0;
}
```

Flow Control: For loop equivalent While loop

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
   /*for (int i = 0; i < count; i++) {

   }*/

    int count = 8;
    int i = 0;
    while (i < count) {
        // …
        // last instruction
        i++;
    }
    return 0;
}
```

Flow Control: For loop, Fibonacci Sequence

```c
/* Program to calculate the first 20 Fibonacci numbers. */
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
    int i, fibonacci[ 20 ];
    fibonacci[ 0 ] = 0; fibonacci[ 1 ] = 1;
    for( i = 2; i < 20; i++ )
        fibonacci[ i ] = fibonacci[ i - 2 ] + fibonacci[ i - 1 ];
    for( i = 0; i < 20; i++ )
        printf( "Fibonacci[ %d ] = %f\n", i, fibonacci[ i ] );
}
```

Flow Control: Break and Continue

```
#include <stdio.h>
int main(int argc, char *argv[]) {
   unsigned char condition = 1;
   int array[] = {5, 6, 2, 8, 12, 19, 20, 13};
   int count = 8;
   int key = 19;
   int index = -1;
   int count_odd = 0;
   for (int i = 0; i < count; i++) {
        if (array[i] == key) {
            index = i;
            break;
        }
        if (array[i] % 2 == 0)
            continue;
        count_odd++;
   }
     return 0;
}
```

Flow Control: Switch-Case

```
#include <stdio.h>

int main(int argc, char *argv[])
{
      unsigned char condition = 1;
      switch (value) {
        case 1:
            // ….
            break;
        case 2:
        case 3:
            // ….
            break;
        default:
            //….
   }
      return 0;
}
```

Questions?

?