

Basics

Rupesh Nasre.
rupesh@cse.iitm.ac.in

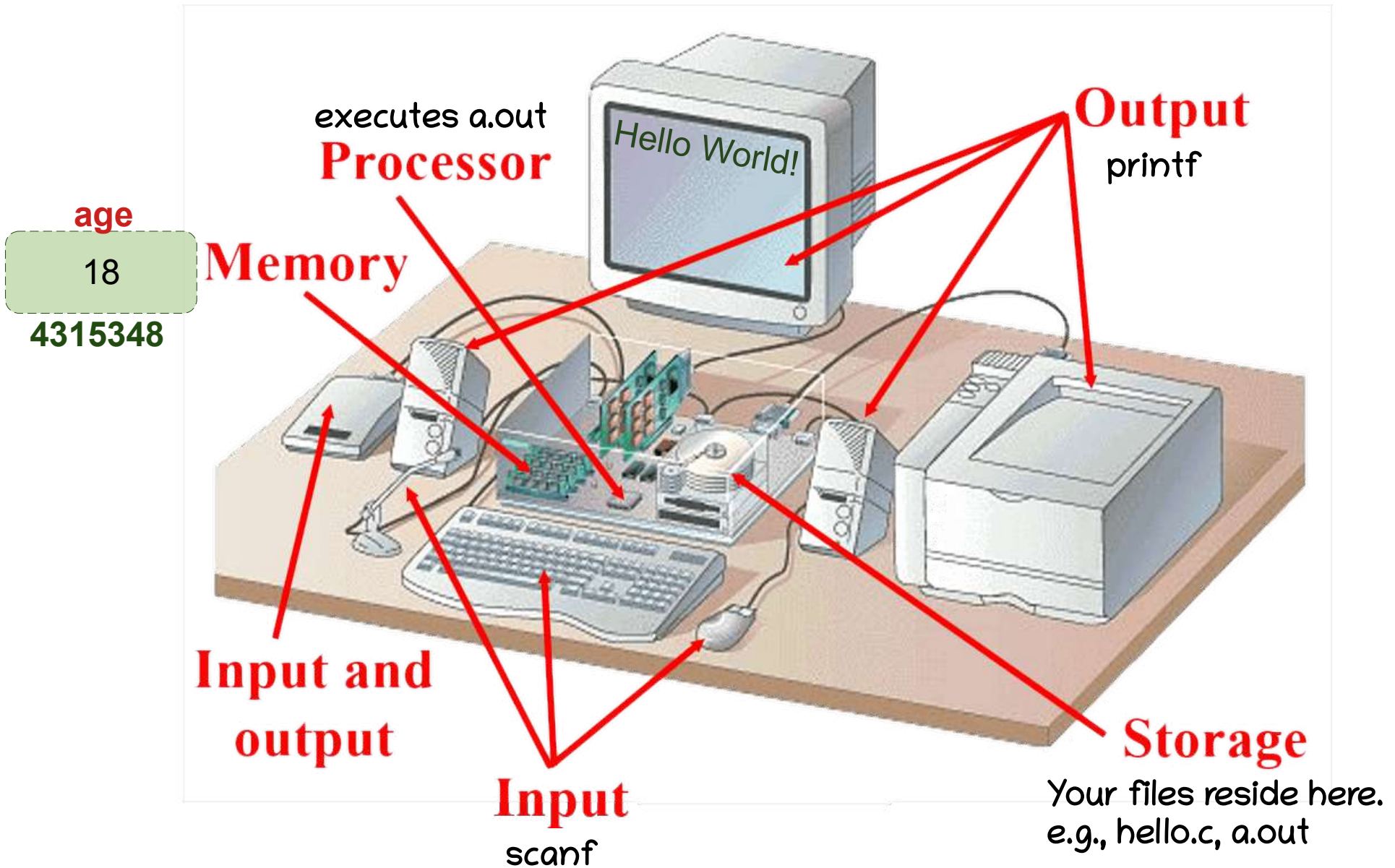
IIT Madras
January 2026

Placement in Computer Science

- CS1111: Problem Solving and Coding
- CS1200: Proofs, Counting
- CS1202: Graphs, Numbers
- **CS1234: Small Applications**
- CS2200: Computation Theory
- CS2300: Overview of Digital World
- CS2600: Hardware
- CS2700: Efficient Implementation
- CS2800: Algorithms
- CS3100: Ways of Programming
- CS3300: Translation (Programmer and Machine)
- CS3500: Resource Management (User and Machine)

CS1234 is foundational to all the future CS subjects.

Computer System



Operating System

- Software which manages hardware
 - I/O device management (`ls cpu`)
 - Process management (`ps`)
 - Memory management (`ls mem`)
 - File system
 - User management and security
- Interface between user and machine
- Examples
 - Linux, Windows, Android, macOS, CentOS, ...

Shell

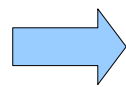
- Shell is also an interface between user and the computer system.
 - Permits text-based interaction
 - Enables automation (shell scripts)
- It is also a piece of software.
- A shell can be developed and deployed by users.
 - You may develop one in the OS course.
 - Your system may have multiple shells (sh, bash, csh, ...)

A terminal window with a black background and green text. The prompt is `#!/bin/bash`. Below it, a command is entered: `>root: env X="() { :; } ; echo shellshock" /bin/sh -c "echo completed"`. The output shows `> shellshock:` followed by `> completed` on the next line.

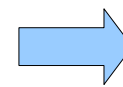
```
#!/bin/bash

>root: env X="() { :; } ; echo shellshock" /bin/sh -c "echo completed"
> shellshock:
> completed
```

c



Compiler

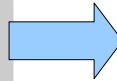


Machine code



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

```
int main() {  
    printf("Hello World!\n");  
}
```

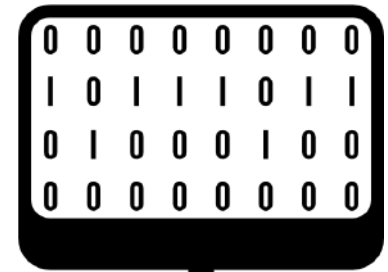


Tamil

Bengali



www.shutterstock.com - 1656635968



```
11000110  
01010011  
01010100  
11101000  
11111110
```

- We will use a translator!
- On your Linux machines, the compiler is **gcc**.

```
$ gcc hello.c
```

```
$ a.out
```

```
Hello World!
```

```
$
```



Command to compile. Translates .c file to a.out.

Run your program (or execute it).

Output of your program.

Command prompt to type the next command.

gcc is also a big program written by many people, **such as you**.

Compiler

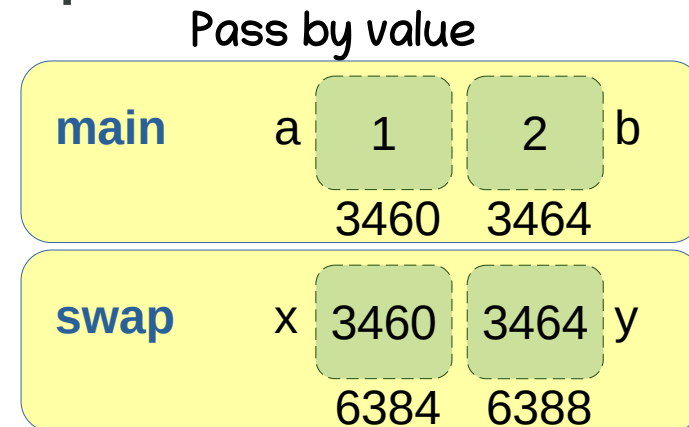
- Compiler checks (primarily) syntax, optimizes the code, and generates machine code.
- Compiler may issue errors / warnings.
- A compiled program does not mean it is correct.
 - A compiler does not check all the semantics (meaning).
 - If I submit working L2 code for L3, the code compiles, but it will not solve the problem.
- A compiler is associated with a language.
 - Same language may have multiple compilers.

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

Pointers

- Regular variables store values. Pointers' values are addresses of other variables.
- Pointers form a new (derived) data type in C.
- Pointers point to variables of a certain type.
- A pointer may contain an address which is either garbage or no longer valid.
- NULL indicates a special pointer, which is 0.
 - Similar to `\0` for strings.



Pointer Dereference

```
int main() {  
    int x = 1, y = 2;  
    int *ptr;  
  
    ptr = &x;  
    *ptr += 10;  
    printf("x=%d, y=%d\n", x, y);  
}
```

Since s is changing,
will str also change?

```
void stringlower(char *s) {  
    while (*s) *s = tolower(*s), ++s;  
}  
  
int main() {  
    char str[] = "Hello World.";  
    char *struptr = str;  
  
    puts(str);  
    while (*struptr != '\0') {  
        *struptr = toupper(*struptr);  
        ++struptr;  
    }  
    puts(str);  
  
    stringlower(str);  
    puts(str);  
}
```

Hello World.
HELLO WORLD.
hello world.

Find the behavior.

```
void fun(int *p, int *xptr) {  
    *p = 10;  
    *xptr = 20;  
}  
int main() {  
    int *p, x;  
    fun(p, &x);  
    printf("*p = %d, x = %d\n", *p, x);  
}
```

```
void fun(int **pptr, int *xptr) {  
    pptr = &xptr;  
    *xptr = 20;  
}  
int main() {  
    int *p, x;  
    fun(&p, &x);  
    printf("*p = %d, x = %d\n", *p, x);  
}
```

```
void fun(int *pptr, int *xptr) {  
    pptr = xptr;  
    *xptr = 20;  
}  
int main() {  
    int *p = NULL, x;  
    fun(p, &x);  
    printf("*p = %d, x = %d\n", *p, x);  
}
```

```
void fun(int **pptr, int *xptr) {  
    *pptr = xptr;  
    *xptr = 20;  
}  
int main() {  
    int *p = NULL, x;  
    fun(&p, &x);  
    printf("*p = %d, x = %d\n", *p, x);  
}
```

Array of Pointers

- Each entry in the array is a pointer.

This leads to a segfault.

`sizeof(arr) == 12 or 24`

`sizeof(arr) == 4 or 8`

```
int main() {  
    int *arr[3], brr[10] = {1, 2, 3};  
    int x, y;  
}
```

```
int **arr; // pointer to a pointer  
  
arr[0] = &x; // type-wise, these are correct  
arr[1] = &y;  
arr[2] = brr;
```

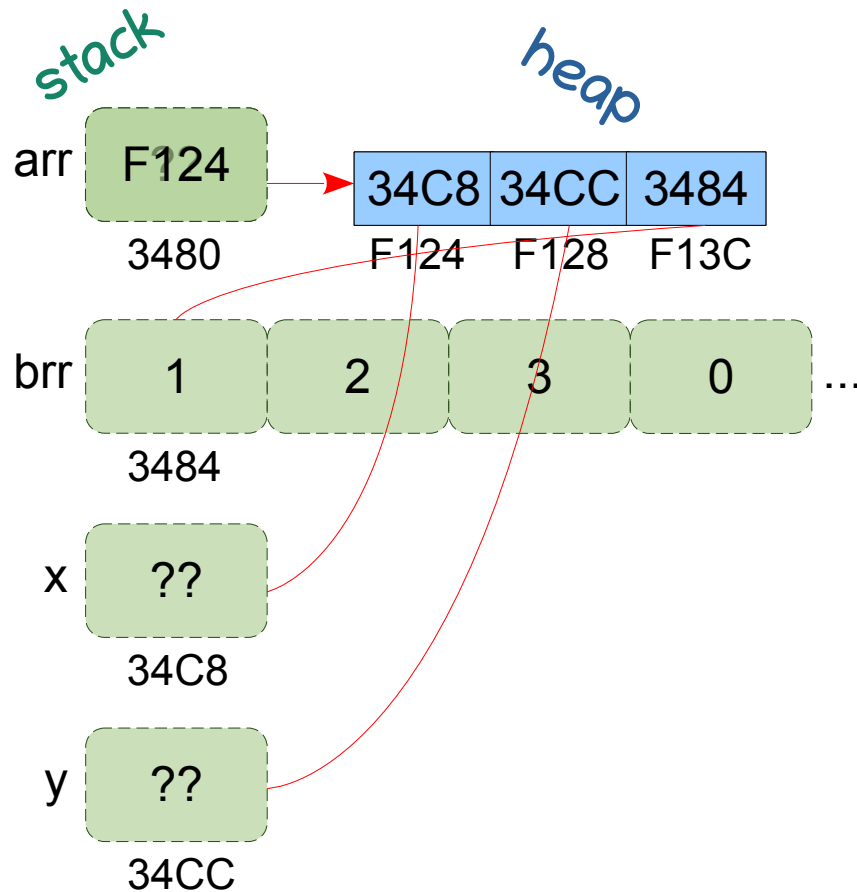
Does this access
resemble a 2D matrix?

1	2	3
4	5	6
7	8	9

`int arr[3][3]`

```
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;  
    }  
}
```

Dynamic Memory Allocation



```
#include <stdlib.h>
```

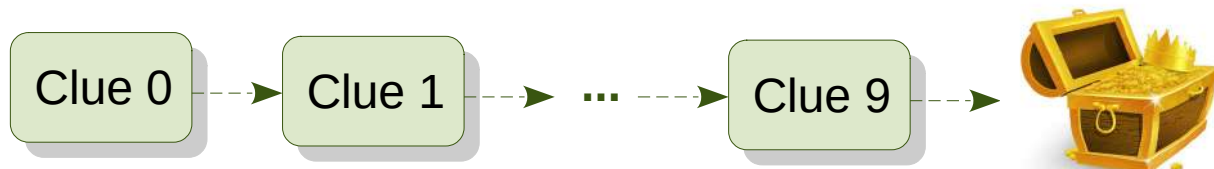
```
int **arr;           // pointer to a pointer
arr = (...)malloc(3 * sizeof(int *));
arr[0] = &x;         // type-wise, these are correct
arr[1] = &y;
arr[2] = brr;
```

Write a program to populate an array that contains pointers to all zeros in another array.

Note: Every malloc should have a corresponding free.
`free(arr);`

Issues with Arrays

- Sometimes, not all the elements are present.
- Sometimes, data is getting streamed.
- Creating `struct student arr[N];` can waste space.
- It would be nice if
 - memory is allocated as and when the datum arrives
 - memory is allocated only for the arrived data
- Need to compromise with contiguousness.



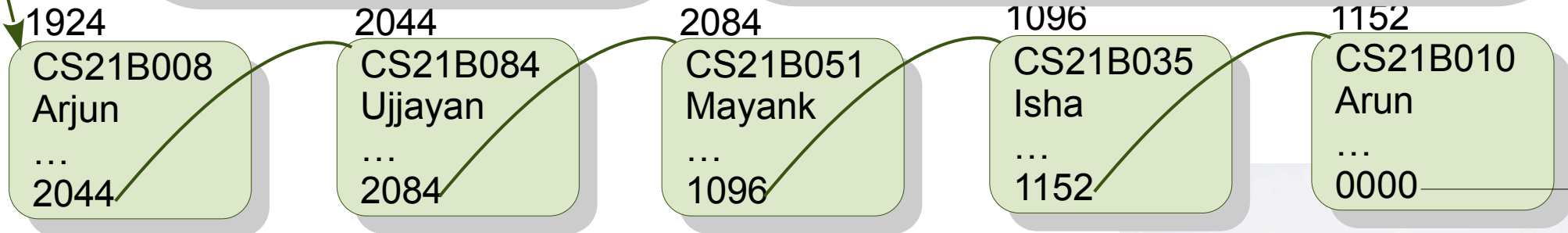
Linked List

```
struct student {  
    char rollno[9];  
    char name[100];  
    char hostel[20];  
    short int room;  
    ...  
    struct student *next;  
};
```

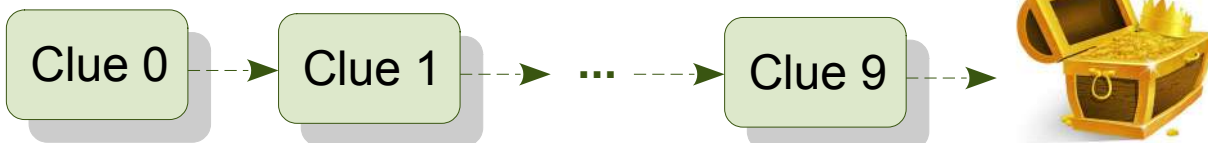
```
void print(struct student *head) {  
    struct student *ptr = head;  
    while (ptr != NULL) {  
        printf("%s %s\n", ptr->rollno, ptr->name);  
        ptr = ptr->next;  
    }  
}
```

(**ptr*).field is so common that C has a shorthand for it: *ptr*→field written as *ptr->field*.

head
1924



If I know the address 1924, I can access all the data in the list.



In this course...

Application	Programming	Tools
Puzzle of grass, goat, lion	Pointers, structures	Editor, visualization
Data maintenance	File I/O	I/O redirection, make
Item grouping	Linked Lists	Debugging
Gene sequencing	Recursion, strings	Memory profiling, ulimit
Closest match	Matrices	Compute profiling, plotting
Keep the recent	Arrays	Version control
Cryptography	Strings, bitwise operators	Library (ncurses)
Unit testing	Command-line	Testing
Web-page creation, analysis	HTML, Strings	wget, shell scripting
Game	Graphics	Library (graphics)
Basics	Python	--

Our goal: *to have fun with automation*