5. Arrays, Pointers and Strings

7th September

IIT Kanpur

Arrays

- An Array is a collection of variables of the same type that are referred to through a common name.
- Declaration

  type var_name[size]

  e.g

```
int A[6];
double d[15];
```

Array Initialization

After declaration, array contains some garbage value.

Static initialization

```
int month_days[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

Run time initialization

```
int i;
int A[6];
for(i = 0; i < 6; i++)
            A[i] = 6 - i;
```

Memory addresses

- Memory is divided up into one byte pieces individually addressed.

  - minimum data you can request from the memory is 1 byte

- Each byte has an address.

  for a 32 bit processor, addressable memory is 2^{32} bytes. To uniquely identify each of the accessible byte you need $\log_2 2^{32}$ = 32 bits

0A	0x00001234
23	0x00001235
6C	0x00001236
1D	0x00001237
‘W’	0x00001238
‘o’	0x00001239
‘w’	0x0000123A
‘\0’	0x0000123B
.	.
.	.
.	.
	0x24680975
	0x24680976
	0x24680977
	0x24680978

Array - Accessing an element

int A[6];

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]
0x1000	0x1004	0x1008	0x1012	0x1016	0x1020
6	5	4	3	2	1

6 elements of 4 bytes each,
total size = 6 x 4 bytes = 24 bytes

Read an element

```
int tmp = A[2];
```

Write to an element

```
A[3] = 5;
```

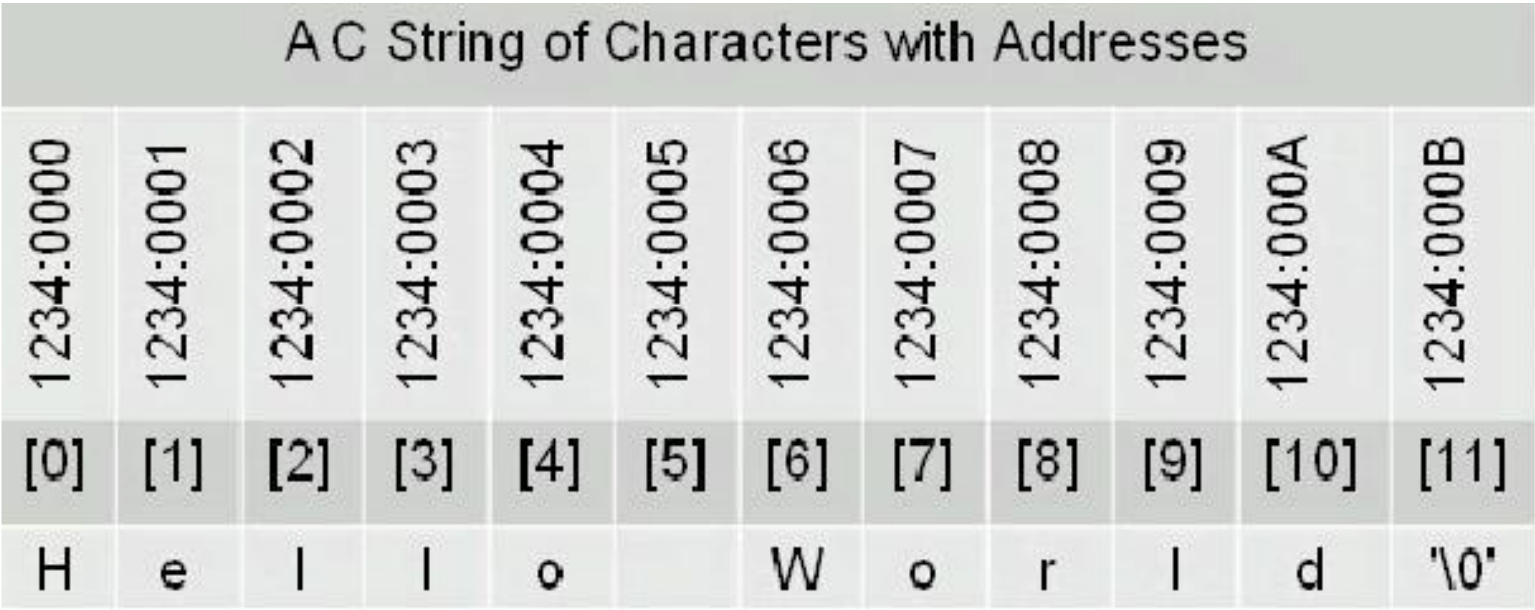
{program: array_average.c}

Strings in C

- No “Strings” keyword
- A string is an array of characters.

```
char string[] = “hello world”;
char *string = “hello world”;
```

OR



A C String of Characters with Addresses

1234:0000	1234:0001	1234:0002	1234:0003	1234:0004	1234:0005	1234:0006	1234:0007	1234:0008	1234:0009	1234:000A	1234:000B
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
H	e	l	l	o		W	o	r	l	d	'\0'

Significance of NULL character ‘\0’

```
char string[] = “hello world”;
printf(“%s”, string);
```

- Compiler has to know where the string ends
- ‘\0’ denotes the end of string

{program: hello.c}

Some more characters (do $man ascii):

‘\n’ = new line, ‘\t’ = horizontal tab, ‘\v’ = vertical tab, ‘\r’ = carriage return
‘A’ = 0x41, ‘a’ = 0x61, ‘\0’ = 0x00

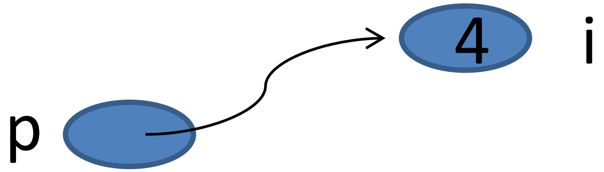
Pointers in C

- A char pointer points to a single byte.
- An int pointer points to first of the four bytes.
- A pointer itself has an address where it is stored in the memory. Pointers are usually four bytes.

```
int *p; ⬄ int* p;
```

- * is called the dereference operator
- *p gives the value pointed by p

```
int i = 4;
p = &i;
```



- & (ampersand) is called the reference operator
- &i returns the address of variable i

More about pointers

```
int x = 1, y = 2, z[10];
int *ip;            /* A pointer to an int */

ip = &x;            /* Address of x */
y = *ip;            /* Content of ip */
*ip = 0;            /* Clear where ip points */
ip = &z[0];         /* Address of first element
                           of z */
```

{program: pointer.c}

Pointer Arithmetic

- A 32-bit system has 32 bit address space.
- To store any address, 32 bits are required.

- Pointer arithmetic : p+1 gives the next memory location assuming cells are of the same type as the base type of p.

Pointer arithmetic: Valid operations

- pointer +/- integer → pointer
- pointer - pointer → integer

- pointer <any operator> pointer → invalid
  - pointer +/- pointer → invalid

Pointer Arithmetic: Example

```
int *p, x = 20;
p = &x;
printf("p    = %p\n", p);
printf("p+1 = %p\n", (int*)p+1);
printf("p+1 = %p\n", (char*)p+1);
printf("p+1 = %p\n", (float*)p+1);
printf("p+1 = %p\n", (double*)p+1);
```

Sample output:

```
p    = 0022FF70
p+1 = 0022FF74
p+1 = 0022FF71
p+1 = 0022FF74
p+1 = 0022FF78
```

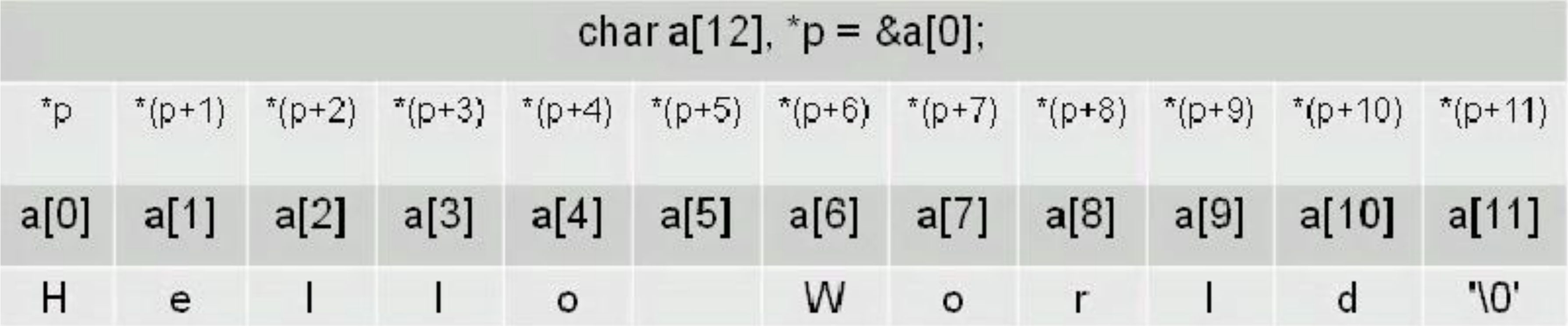
{program: pointer_arithmetic.c}

Pointers and arrays

- Pointers and arrays are tightly coupled.

char a[] = “Hello World”;

char *p = &a[0];



char a[12], *p = &a[0];											
*p	*(p+1)	*(p+2)	*(p+3)	*(p+4)	*(p+5)	*(p+6)	*(p+7)	*(p+8)	*(p+9)	*(p+10)	*(p+11)
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]
H	e	l	l	o		W	o	r	l	d	'\0'

Pointers and function arguments

- Functions only receive copies of the variables passed to them.

{program: swap_attempt_1.c}

- A function needs to know the address of a variable if it is to affect the original variable

{program: swap_attempt_2.c}

- Large items like strings or arrays cannot be passed to functions either.

```
printf(“hello world\n”);
```

- What is passed is the address of “hello world\n” in the memory.

2-Dimensional Arrays (Array of arrays)

```
int d[3][2];
```

Access the point 1, 2 of the array:

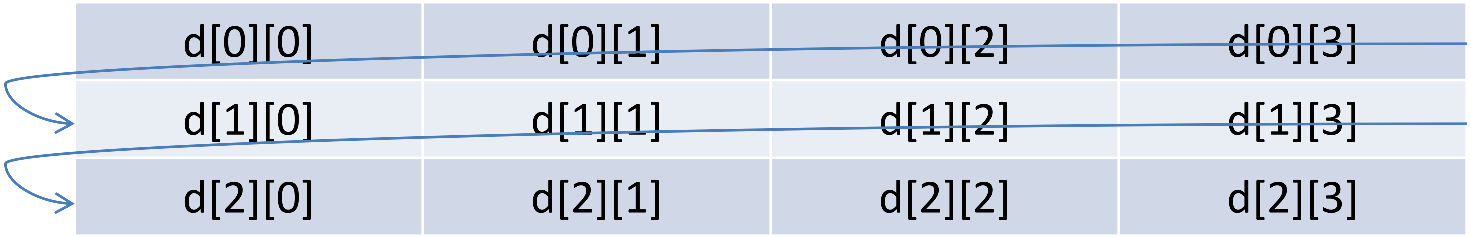
```
d[1][2]
```

Initialize (without loops):

```
int d[3][2] = {{1, 2}, {4, 5}, {7, 8}};
```

More about 2-Dimensional arrays

A Multidimensional array is stored in a row major format.
A two dimensional case:
➔ next memory element to d[0][3] is d[1][0]



d[0][0]	d[0][1]	d[0][2]	d[0][3]
d[1][0]	d[1][1]	d[1][2]	d[1][3]
d[2][0]	d[2][1]	d[2][2]	d[2][3]

What about memory addresses sequence of a three dimensional array?
➔ next memory element to t[0][0][0] is t[0][0][1]