6. More on Pointers

14th September

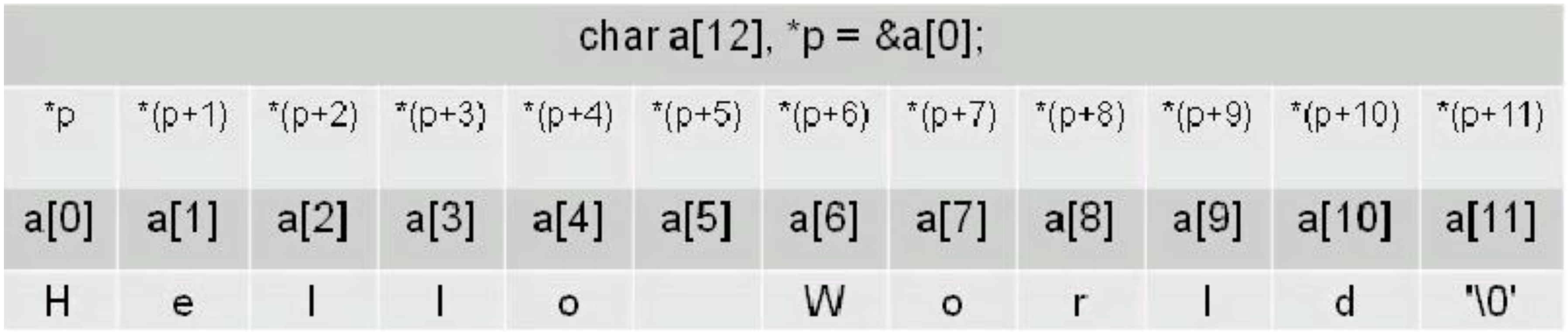
IIT Kanpur

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 arrays contd..

- Name of the array is synonymous with the address of the first element of the array.

```
int *p;
int sample[10];
p = sample;           // same as p = &sample[0];
```

```
int *p;
int sample[10];
p = sample;
p[5] = 100;           // Both these statements
*(p+5) = 100;         // do the same thing
```

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.

Passing single dimension arrays to functions

- In C, you cannot pass the entire data of the array as an argument to a function.
  - How to pass array then?
    - Pass a pointer to the array.

```
int main() {
		int sample[10];
		func1(sample);
		…
}
void func1(int *x) {
		…
}
void func1(int x[10]) {
		…
}
void func1(int x[]) {
		…
}
```

2-Dimensional Arrays (Array of arrays)

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

Access the point 1, 2 of the array:

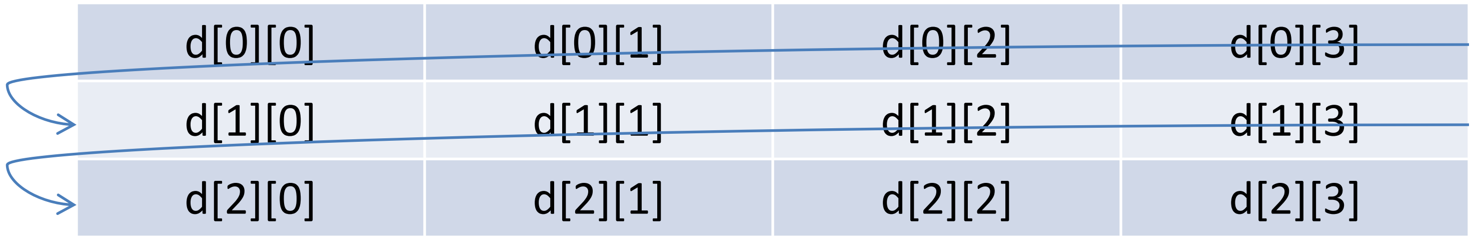
d[1][2]

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]

Multidimensional Arrays

- Syntax

type array_name[size1][size2]…[sizeN];

e.g

```
int a[3][6][4][8];
```

size of array = 3 x 6 x 4 x 8 x 4 bytes

Arrays of Pointers

```
int *x[10];
```

Declares an array of int pointers. Array has 10 pointers.

Assign address to a pointer in array

```
x[2] = &var;
```

To find the value of var,

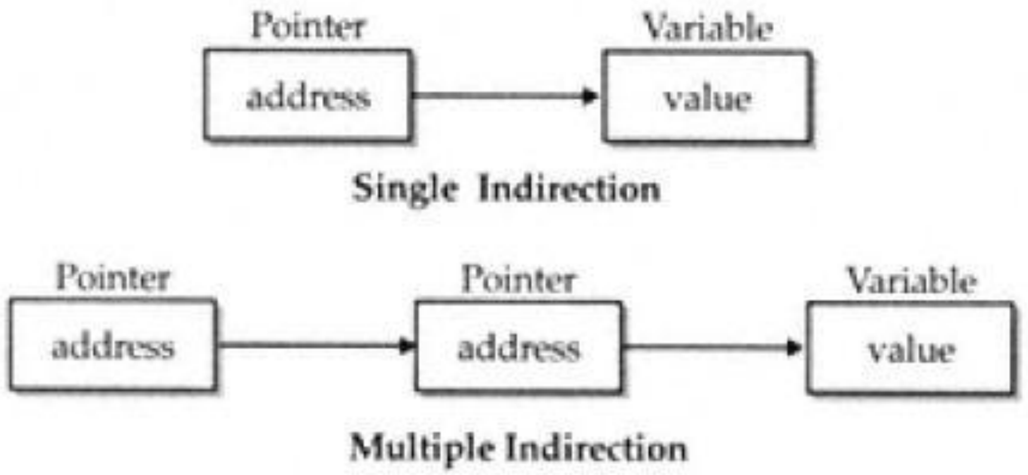
```
int i =*x[2];
```

Pointer to Pointer

- Declaration
  - Place an additional asterisk

```
double **newbalance;
```

newbalance is a pointer to a double pointer.



Pointer to Pointer contd..

```
#include <stdio.h>

int main() {
		int x, *p, **q;
		x = 10;
		p = &x;
		q = &p;

		printf(“%d %d %d\n”, x, *p, **q);
		return 0;
}
```

{program: pointers.c}

Dynamic Memory Allocation

- To allocate memory at run time.
- malloc(), calloc()
  - both return a void*
    - you'll need to typecast each time.

```
char *p;
p = (char *)malloc(1000); /*get 1000 byte space */
```

```
int *i;
i = (int *)malloc(1000*sizeof(int));
```

Dynamic Memory Allocation contd..

- To free memory
- free()
  - free(ptr) frees the space allocated to the pointer ptr

```
int *i;
i = (int *)malloc(1000*sizeof(int));
.
.
.
free(i);
```

Pointers to functions

- A function pointer stores the address of the function.
- Function pointers allow:
  - call the function using a pointer
  - functions to be passed as arguments to other functions

return_type (*function_name)(type arg1, type arg2…)

{program: function_pointer.c}