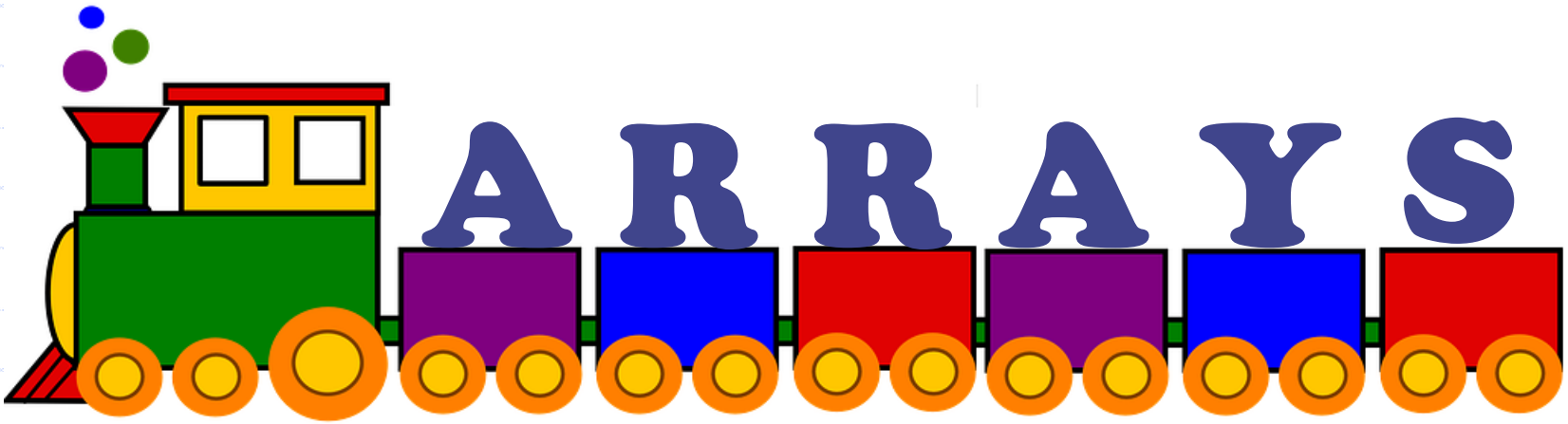
ESC101: Introduction to Computing



Defining arrays

Dictionary meaning of the word array

arr-ay: noun

1. a large and impressive grouping or organization of things: He couldn't dismiss the array of facts.
2. regular order or arrangement; series: an array of figures.

Arrays in C

An array in C is defined similar to defining a variable.

```
int a[5];
```

The square parenthesis [5] indicates that a is not a single integer but an array, that is a ***consecutively allocated*** group, of 5 integers.

It creates five integer boxes or variables

a[0] a[1] a[2] a[3] a[4]

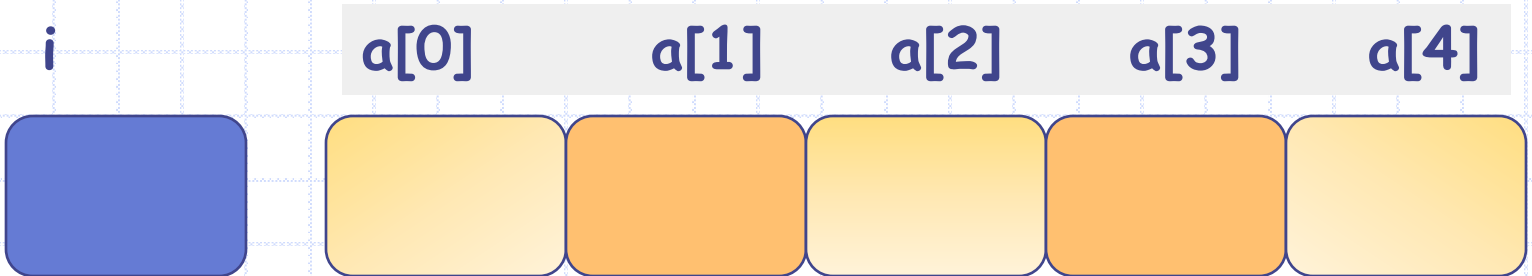
The boxes are addressed as a[0], a[1], a[2], a[3] and a[4]. These are called the **elements** of the array.

```
include <stdio.h>
int main () {
      int i;
      int a[5];

       for (i=0; i < 5; i= i+1) {
            a[i] = i+1;

            printf("%d", a[i] );
       }
       return 0;
}
```

The program defines an integer variable called i and an integer array with name a of size 5

This is the notation used to address the elements of the array.

- The variable i is being used as an "index" for a.
- Similar to the math notation a_i



```
include <stdio.h>
int main () {
      int a[5];
      int i;
       for (i=0; i < 5; i= i+1) {
            a[i] = i+1;
        }
        return 0;
}
```

Let us trace through the execution of the program.

Fact : Array elements are consecutively allocated in memory.

i 5

a[0]	a[1]	a[2]	a[3]	a[4]
1	2	3	4	5

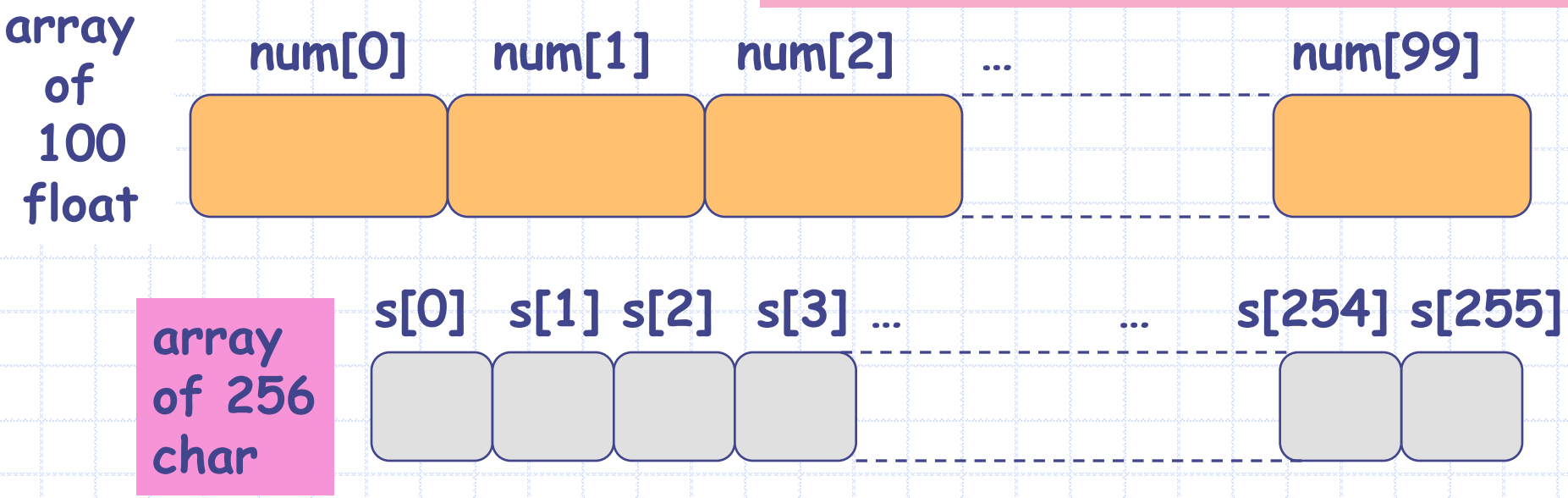
Statement becomes a[0] =0+1;
Statement becomes a[1] =1+1;
Statement becomes a[2] =2+1;
Statement becomes a[3] = 3+1;
Statement becomes a[4] = 4+1;

One can define an array of float or an array of char, or array of any data type of C. For example

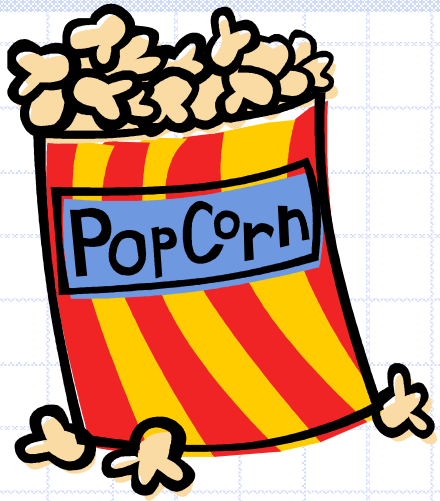
```
int main() {
        float num[100];

        char s[256];

 /* some code here */
}
```

This defines an array called num of 100 floating point numbers indexed from 0 to 99 and named num[0]… num[99]

This defines an array called s of 256 characters indexed from 0 to 255 and named s[0]…s[255].



Mind the size(of array)

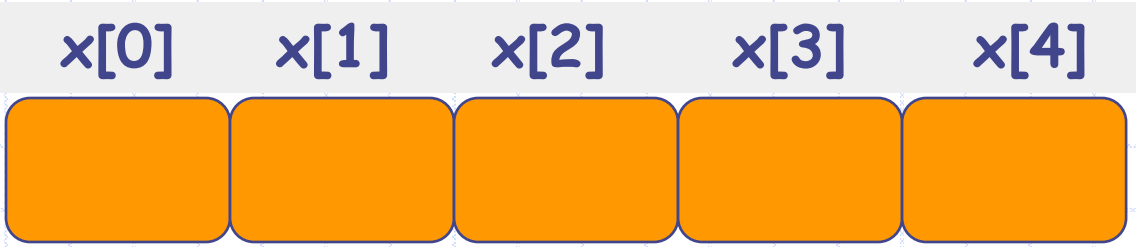


Consider program fragment:

```
int f() {
    int x[5];
        …
}
```

This defines an integer array named x of size 5.

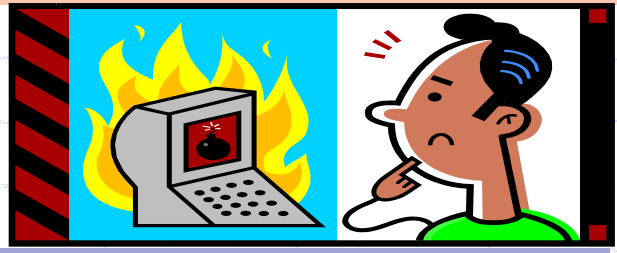
Five integer variables named x[0] x[1] … x[4] are allocated.



The variables x[0],x[1] … x[4] are integers, and can be assigned and operated upon like integers! OK, so far so good!

But what about x[5], x[6], … x[55]?
Can I assign to x[5], increment it, etc.?

NO! Program may crash!



Why?

x[5], x[6], and so on are undefined. These are names but no storage has been allocated. Shouldn't access them!

Q: Shouldn't I or couldn't I access array elements outside of the array range declared?

Ans: You can but shouldn't. Program may crash.


```
int f() {
      int x[5];
      x[0] =0;
      x[1] =1;
        ...
      x[4] =4;

      x[5] = 5;
      x[6] = 6;
}
```

All good

Both these statements are not recommended.

Will it compile? Yes, it will compile. C compiler may give a warning.

But, upon execution, the program may give "segmentation fault: core dumped" error or it may also run correctly and without error.

Reading directly into array

Read N numbers from user directly into an array

```
#include <stdio.h>
int main() {
   int num[10];
   for (i=0; i<10; i=i+1) {
        scanf("%d", &num[i]);
   }
   return 0;
}
```

scanf can be used to read directly into array by treating an array element like any other variable of the same data type.

1. For integers, read as scanf("%d", &num[i]);
2. For reading elements of a char array s[], use scanf("%c", &s[j]).

In the previous slide, we had the statement:

scanf("%d", &num[i]);

What does &num[i] mean?

&num[i] is made of two operators & and []. & num [i] gives the address of the array element num[i].

& is the "address-of" operator.

1. It can be applied to any defined variable.
2. It returns the location (i.e., address) of this variable in the program's memory.

[ ] is the array indexing operator, e.g, num[i].

&num[i] is evaluated as &(num[i]).

& is applied to the result of applying the indexing operator [i] to num.

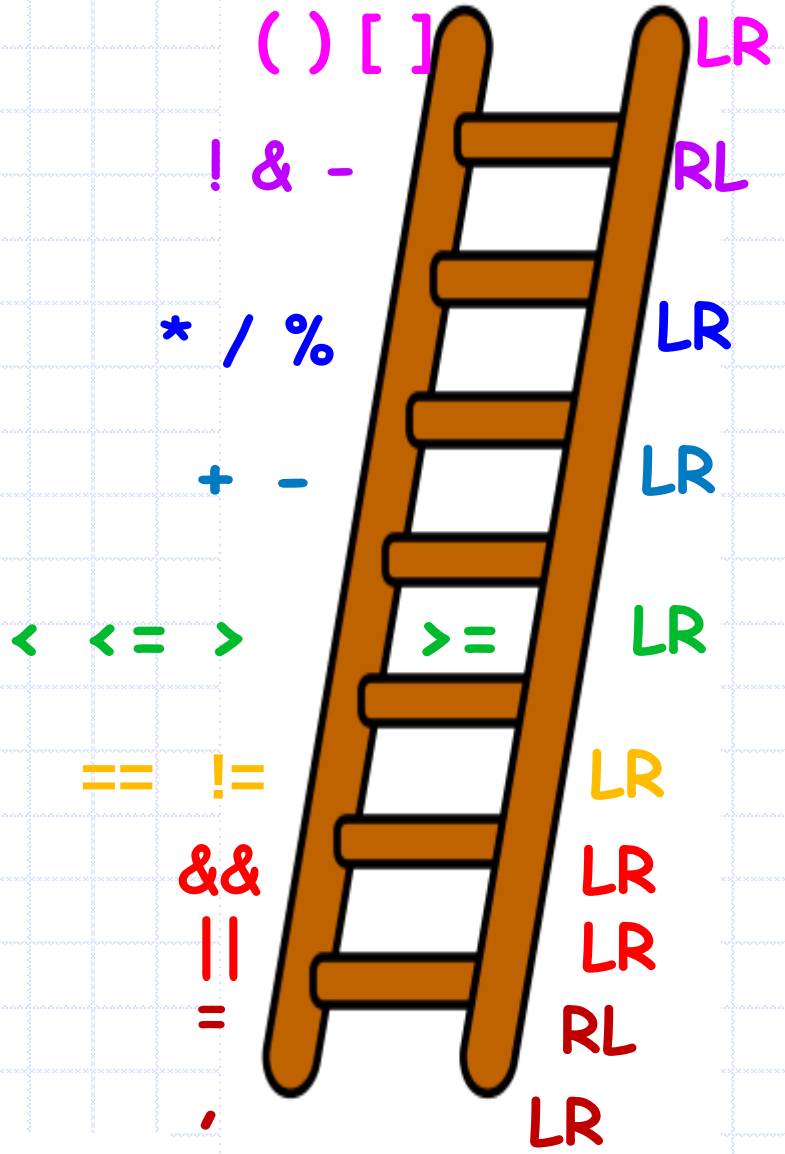
NOT as

(&num)[i] which would mean that first & is applied to num and [] operator is applied to &num

We have seen that &num[i] is evaluated by applying the indexing operator first and the address-of operator second.

More formally, the precedence of the operators in C reflects this.

1. The array indexing operator [] is given higher precedence than the address-of operator &.
2. So &num[i] is evaluated by applying the array operator first and the address-of operator next.



Legend LR: Left-to-Right associativity
RL: Right –to-Left associativity

Array Example: Print backwards

Problem:

Define a character array of size 100 (upper limit) and read the input character by character and store in the array until either

- 100 characters are read or
- EOF (End Of File) is encountered

Now print the characters backwards from the array.

Example Input 1

```
Me or
Moo
```

Output 1

```
ooM
ro eM
```

Example Input 2

```
Eena Meena Dika
```

Output 2

```
akiD aneeM aneE
```

Read and print in reverse

1. We will design the program in a top down fashion, using just main() function.
2. There will be two parts to main: read_into_array and print_reverse.
3. read_into_array will read the input character-by-character up to 100 characters or until a end of input.
4. print_reverse will print the characters in reverse.

Overall design

```
int main() {
        char s[100]; /* to hold the input */
        /* read_into_array */
        /* print_reverse */
        return 0;
}
```

Let us design the program fragment read_into_array.

Keep the following variables:

1. int count to count the number of characters read so far.
2. int ch to read the next character using getchar().

Note that getchar() has prototype **int getchar()** since getchar() returns all the 256 characters and the **integer EOF**

```
int ch;
int count = 0;
read the next character into ch using getchar();
while (ch is not EOF AND count < 100) {
        s[count] = ch;
        count = count + 1;
        read the next character into ch using getchar();
}
```

An initial design (pseudo-code)

```
int ch;
int count = 0;
read the next character into ch using getchar();
while (ch is not EOF AND count < 100) {
        s[count] = ch;
        count = count + 1;
        read the next character into ch using getchar();
}
```

initial design pseudo-code

```
int ch;
int count = 0;
ch = getchar();
while ( ch != EOF && count < 100) {
        s[count] = ch;
        count = count + 1;
        ch = getchar();
}
```

Translating the read_into_array pseudo-code into code.

Overall design

```
int main() {
    char s[100];
 /* read_into_array */
 /* print_reverse */
    return 0;
}
```

What is the value of count at the end of read_into_array?