Arrays (1.6)

EECS 2031

Arrays

- Grouping of data **of the same type.**
- Loops commonly used for manipulation.
- Programmers set array sizes explicitly.

Arrays: Example

- Syntax

```
type name[size];
```

- Examples

```
int bigArray[10];
double a[3];
char grade[10];
```

3

Arrays: Definition and Access

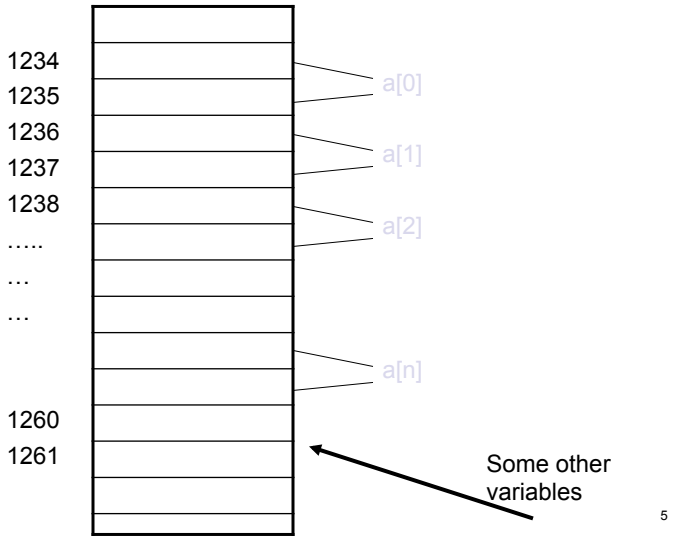
- Defining an array: allocates memory

```
int score[5];
```

  - allocates an array of 5 integers named "score"
- Individual parts can be called:
  - indexed or subscripted variables
  - "elements" of the array
- Value in brackets called index or subscript
  - numbered from 0 to (size – 1)

4

Arrays Stored in Memory



5

Initialization

- In declarations enclosed in curly braces

int a[5] = {11,22}; Declares array a and initializes first two elements and all remaining set to zero

int b[ ] = {1,2,8,9,5}; Declares array b and initializes all elements and sets the length of the array to 5

6

Array Access

```
x = ar[2];
ar[3] = 2.7;
```

- What is the difference between
  `ar[i]++, ar[i++], ar[++i]` ?

7

Strings

- No **`string`** type in C
- String = array of char
- char gretings[ ] = “Hello”

H	e	l	l	o	\0

8

Control Flow (Chapter 3)

EECS 2031

16 September 2017

Statements and Blocks (3.1)

- Statement: followed by a semicolon.
- Block
  - enclosed between { and }
  - syntactically equivalent to a single statement
  - no semicolon after the right brace
- Variables can be declared inside *any* block.

Control Flow Statements

- Similar to Java
- **`if – else`**
- **`else – if`**
- **`switch`**
- **`while`**
- **`for`**
- **`do – while`**
- **`break`**
- **`break`**
- **`continue`**
- **`goto`**
- labels

11

if – else

```
if (n > 0)
  if (a > b)
     z = a;
  else
     z = b;
```

```
if (n > 0) {
  if (a > b)
     z = a;
}
else
  z = b;
```

12

if – else – if

```
int binary_search( int x, int v[], int n ) {
   int low, high, mid;
   low = 0;
   high = n - 1;
   while (low <= high) {
        mid = (low + high)/2;
        if (x < v[mid])
               high = mid + 1;
        else if (x > v[mid])
               low = mid + 1;
        else /* found match */
               return mid;
   }
   return -1; /* no match */
}
```

13

switch

```
while ((c = getchar()) != EOF) {
   switch (c) {
   case '0': case '1': case '2': case '3': case '4':
   case '5': case '6': case '7': case '8': case '9':
        ndigit[c-'0']++;
        break;
   case ' ':
   case '\n':
   case '\t':
        nwhite++;
        break;
   default:
        nother++;
        break;
   }
}
```

14

`while` and `for` Loops

```
while ((c = getchar()) == ' ' || c == '\n'
            || c = '\t')
  ; /* skip white space characters */


for (i = 0; i < n; i++)
  ...
```

15

do – while

```
do {
  s[i++] = n % 10 + '0';
} while ((n /= 10) > 0);
```

Note: the above curly brackets are not necessary. They just make the code more readable.

16

continue

Skip negative elements; increment non-negative elements.

```
for (i = 0; i < n; i++) {
  if (a[i] < 0)      /* skip negative */
     continue;
  a[i]++;  /* increment non-negative */
}
```

17

break

Return the index of the first negative element.

```
...
for (i = 0; i < n; i++)
  if (a[i] < 0) /* 1st negative element */
     break;
if (i < n)
  return i;
...
```

18

`goto` and Labels

Determine whether arrays a and b have an element in common.

```
  for (i = 0; i < n; i++)
     for (j = 0; j < m; j++)
          if (a[i] == b[j])
               goto found;
  /* didn't find any common element */
  ...
found:
  /* got one: a[i] == b[j] */
  ...
```

19

Notes

- Code that relies on `goto` statements is generally harder to understand and to maintain. So `goto` statements should be used rarely, if at all.
- `break` and `continue` should be used only when necessary.

20

Next lecture...

- Types, operators and expressions (chapter 2, C book)

21