Chapter Eleven

Arrays

1

A Motivating Example

Input 5 integers and print them in reverse order

```
main( ) {
   int  n0, n1, n2, n3, n4;
   scanf("%d",  &n0);
   scanf("%d",  &n1);
   scanf("%d",  &n2);
   scanf("%d",  &n3);
   scanf("%d",  &n4);
   printf("n%d = %d\n", n4);
   printf("n%d = %d\n", n3);
   printf("n%d = %d\n", n2);
   printf("n%d = %d\n", n1);
   printf("n%d = %d\n", n0);
}
```

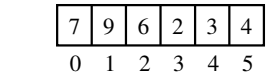
2

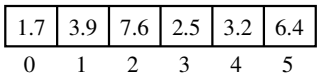
Arrays

- An *array* is a *consecutive* group of memory locations with two characteristics
- An array is *homogeneous*: all memory locations in the array store data of the *same type*
- An array is *ordered*: memory locations in the array are named in *ordered integer index* beginning at zero

3

Examples





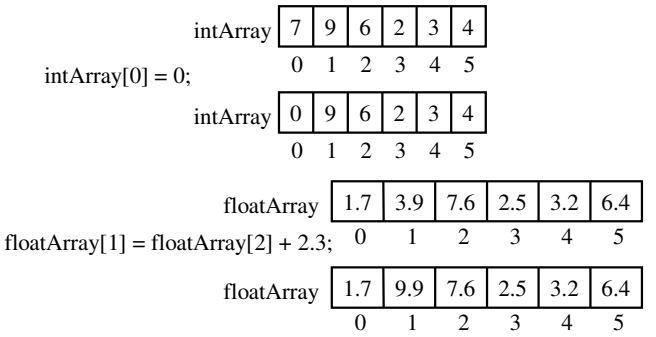
4

Array Declaration & Access

- Arrays are declared as
  element-type array-name [*array-size*];
  int intArray[6];
  float floatArray[6];
- Array elements are accessed as
  intArray[0] = 0;
  floatArray[1] = floatArray[2] + 2.3;

5

Examples



6

An Example

```
#define SIZE 5
main( )
{
   int i, n[SIZE];

   for ( i = 0; i < SIZE; i++ ) {
      scanf("%d", &n[i]);
   }
   for ( i = SIZE - 1; i >= 0; i-- ) {
      printf("n[%d] = %d\n", i, n[i]);
   }
}
```

7

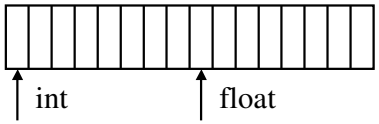
An Example

```
#define SIZE 5
main( )
{
   int i, sum, n[SIZE];
   sum = 0;
   for ( i = 0; i < SIZE; i++ )
      scanf("%d", &n[i]);
   for ( i = 0; i < SIZE; i++ )
      sum += n[i];
   printf("sum = %d\n", sum);
}
```

8

Address of Variables

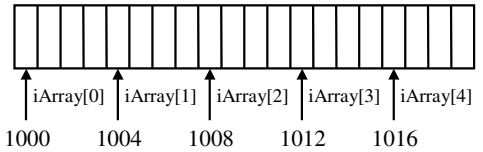
- In memory, every *byte* is identified by an address
- Data values requiring multiple bytes are identified by the address of the *first byte*



9

Address of Array Elements

int iArray[5]; number of bytes = 4 * 5 = 20



address of iArray[i] = 1000 + 4 * i
base address offset

10

Common Pitfalls

- Whenever you use arrays in your programs, make sure that the *index values* used to select elements from the array *remain within the array bounds*
- On most computers, referencing elements that are outside the array bounds *is not detected as an error* but will *certainly lead to unpredictable results*

11

Passing Arrays as Parameters

```
#define  SIZE  5
main( )
{
   int  n[SIZE];

   inputArray(n); /* use 0 as sentinel value */
   reverseArray(n);
   printArray(n);
}
```

12

Two Issues

- The required size of the array n is unknown
- The array n passed to the two functions inputArray and reverseArray should be changed by these two functions

13

Generalizing the Size of Arrays

- The usual strategy is to declare an array that is larger than you need and use only part of it
- The number of elements declared is called the *allocated size* of the array
- The number of elements actually in use is called the *effective size* of the array

14

Generalizing the Size of Arrays

```
int  n[MAXSIZE];

void printArray(int n[MAXSIZE], int size);
void printArray(int n[], int size);

void reverseArray(int n[], int size);

int inputArray(int n[], int maxsize);
int inputArray(int n[], int maxsize, int sentinel);
```

15

Generalizing the Size of Arrays

```
#define  MAX  100
main( )
{
   int  n[MAX], size;

   size = inputArray(n, MAX, 0);
   reverseArray(n, size);
   printArray(n, size);
}
```

16

Passing Array Arguments

- When an array is passed to a function, instead of copying the entire array to the function, only *the base address* of the array is passed to the function
- The *array parameter* is thus a *synonym* of the *array argument*. Changing the elements of the array parameter is the same as changing the elements of the array arguments

17

printArray

```
static void printArray(int  array[], int size)
{
   int  i;

   for (i = 0; i < size; i++) {
      printf("%d\n", array[i]);
   }
}
```

18

inputArray

```
static int inputArray(int  array[], int max, int sentinel){
   int  n, value;
   n = 0;
   while (TRUE) {
      printf("?");
      scanf("%d", &value);
      if (value == sentinel)  break;
      if (n == max) {printf("Error: array full"); exit(1); }
      array[n++] = value;
   }
   return n;
}
```

19

reverseArray

```
static void reverseArray(int  array[], int size)
{
   int  i;

   for (i = 0; i < size / 2; i++) {
      /* swap(array[i], array[size – i –1]); */
      swap(array, i, size – i –1);
   }
}
```

20

swap

```
static void swap(int  array[], int p1, int p2)
{
   int  tmp;

   tmp = array[p1];
   array[p1] = array[p2];
   array[p2] = tmp;
}
```

21

An Example

```
<<This program counts letter frequencies>>
Peter Piker picked a peck
Of pickled peppers.

A   1
C   3
D   2
E   8
F   1
I   3
…
```

22

An Example

```
int nA, nB, nC, …, nZ;

int  letterCounts[26];

int  letterIndex(char ch)
{
   if (isalpha(ch)) {
      return toupper(ch) – ‘A’;
   } else {
      return –1;
   }
}
```

23

An Example

```
void recordLetter(char ch, int letterCounts[])
{
   int  index;

   index = letterIndex(ch);
   if (index != -1) letterCounts[index]++;
}
```

24

An Example

```
void clearIntArray(int array[], int n)
{
   int  i;

   for (i = 0; i < n; i++) {
      array[i] = 0;
   }
}
```

25

An Example

```
void displayLetterCounts(int letterCounts[])
{
   char  ch;
   int  num;

   for (ch = 'A'; ch <= 'Z'; ch++) {
      num = letterCounts[letterIndex(ch)];
      if (num != 0) printf("%c %4d\n", ch, num);
   }
}
```

26

Static Initialization of Arrays

```
int  digits[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
int  digits[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

string bigCities[] = {
   "New York", "Los Angeles", "Chicago",
   "Houston", "Philadelphia", "San Diego",
   "Detroit", "Dallas",
}

int nBigCities = sizeof bigCities / sizeof bigCities[0];
```

27

Scalar-Type Array Index

```
string booleanName[2] = {"FALSE", "TRUE"};

typedef  enum  {FALSE, TRUE}  bool;

printf("flag = %s\n", booleanName[flag]);
```

28

Multidimensional Arrays

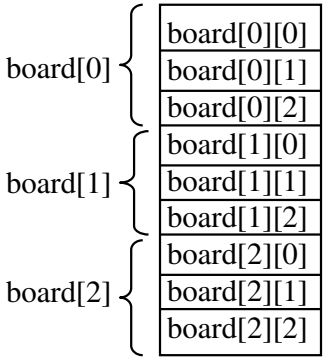
- Arrays of arrays are called *multidimensional* arrays

  char board[3][3];

board[0][0]	board[0][1]	board[0][2]
board[1][0]	board[1][1]	board[1][2]
board[2][0]	board[2][1]	board[2][2]

29

Multidimensional Arrays



30

Passing Multidimensional Arrays

```
void displayBorad(char board[3][3])
{
   int  row, column;
   for (row = 0; row < 3; row++) {
      if (row != 0) printf("---+---+---\n");
      for (column != 0; column < 3; column++) {
         if (column != 0) printf("|");
         printf(" %c ", borad[row][column]);
      }
      printf("\n");
   }
}
```

31

Initializing Multidimensional Arrays

```
double identityMatrix[3][3] = {
   {1.0, 0.0, 0.0},
   {0.0, 1.0, 0.0},
   {0.0, 0.0, 1.0}
};
```

32