COMP1511: Iterations, Collection - Arrays

Session 2, 2018

Iteration: Sum

- Read numbers until end of input (or a non-number) is reached then print the **sum** of the numbers

version -01

```c
#include <stdio.h>

int main(void) {
    int sum, x, noRead;

    sum = 0;

    printf("Enter number: ");
    noRead = scanf("%d", &x);
    while (noRead == 1) {
        sum = sum + x;
        printf("Enter number: ");
        noRead = scanf("%d", &x);
    }

    printf("Sum of the numbers is %d\n", sum);
    return 0;
}
```

scanf

version -02

```c
#include <stdio.h>

int main(void) {
    int sum, x;

    sum = 0;

    printf("Enter number: ");
    while (scanf("%d", &x) == 1) {
        sum = sum + x;
        printf("Enter number: ");
    }

    printf("Sum of the numbers is %d\n", sum);
    return 0;
}
```

scanf

Iteration: Max

- Read numbers until end of input (or a non-number) is reached then print the **maximum** of the numbers

*Read **first value** and consider it as maximum so far*

If required, update current maximum value (maximum value so far)

Final Maximum value

```c
#include <stdio.h>

int main(void) {
    int x, noRead, currentMax;

    printf("Enter number: ");
    noRead = scanf("%d", &x);
    if(noRead != 1){
        printf("No number entered!\n");
        return 0;
    }
    currentMax = x;

    printf("Enter number: ");
    noRead = scanf("%d", &x);
    while (noRead == 1) {

        if(x > currentMax) {
            currentMax = x;
        }

        printf("Enter number: ");
        noRead = scanf("%d", &x);
    }

    printf("Max of the numbers is %d\n", currentMax);

    return 0;
}
```

Iteration: “Previous’ pattern

A simple program which reads integers and prints **snap** and exits if the same number is **read twice in a row.**

- Note for simplicity we are assuming scanf succeeds in reading an integer.
- A robust program would check that scanf returns 1 to indicate an integer read.

```
int main(void) {
    int currentN, previousN;

    printf("Enter a number: ");
    scanf("%d", &previousN);

    printf("Enter a number: ");
    scanf("%d", &currentN);

    while (currentN != previousN) {
        previousN = currentN;
        printf("Enter a number: ");
        scanf("%d", &currentN);
    }

    printf("Snap!\n");
    return 0;
}
```

Assign current number to previousN for next iteration

Iteration Demos

- fibonacci.c
- calculate_pi.c
- calculate_e.c
- Etc. ….

Collection - Arrays

Suppose I need to compute statistics on class marks?

```
int mark_student0, mark_student1, mark_student2,
mark_student0 = 73;
mark_student1 = 42;
mark_student2 = 99;
...
```

- cumbersome, need hundreds of individual variables
- can't write while loop which executes for each student
- becomes unfeasible if dealing with a lot of values

Solution use an array

```
int mark[930];
mark[0] = 73;
mark[1] = 42;
mark[2] = 99;
...
```

C Array

- C array is a **collection** of variables called **array elements**.
- All array elements must be the **same type**.
- Array elements don't have a name
- Array elements accessed by a number called the **array index**.
- Valid array indices for array with `n` elements are `0 to n - 1`
- Array can have millions/billions of elements.
- Array elements must be **initialized**.
- **Can't** assign scanf/printf **whole** arrays.
- Can assign scanf/printf array elements.

Array

```
// Declare an array with 10 elements
// and initialises all elements to 0
int myArray[10] = {0};
```

	myArray
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0

Array

```
// Declare an array with 10 elements
// and initialises all elements to 0
int myArray[10] = {0};

// Put some values into the array.
myArray[0] = 3;
```

	myArray
0	3
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0

Array

```
// Declare an array with 10 elements
// and initialises all elements to 0
int myArray[10] = {0};

// Put some values into the array.
myArray[0] = 3;
myArray[5] = 17;
```

	myArray
0	3
1	0
2	0
3	0
4	0
5	17
6	0
7	0
8	0
9	0

Array

```
// Declare an array with 10 elements
// and initialises all elements to 0
int myArray[10] = {0};

// Put some values into the array.
myArray[0] = 3;
myArray[5] = 17;
myArray[10] = 42; // <-- Error
```

	myArray
0	3
1	0
2	0
3	0
4	0
5	17
6	0
7	0
8	0
9	0

Reading Arrays

Scanf can't read an entire array. This will read only 1 number:

```
#define ARRAY_SIZE 42
...
int array[ARRAY_SIZE];
scanf("%d", &array);
```

Instead you must read the elements one by one:

```
i = 0;
while (i < SIZE) {
    scanf("%d", &array[i]);
    i = i + 1;
}
```

Printing Arrays

printf can't print an entire array. This won't compile:

```
#define ARRAY_SIZE 42
...
int array[ARRAY_SIZE];
printf("%d", array);
```

Instead must print the elements one by one:

```
i = 0;
while (i < ARRAY_SIZE) {
    printf("%d\n", array[i]);
    i = i + 1;
}
```

Copying Arrays

Suppose we have the following:

```
int array1[5] = {1, 2, 3, 4, 5};
int array2[5];
```

Array assignment not allowed in C. This won't compile:

```
array2 = array1;
```

Instead must must copy the elements one by one:

```
i = 0;
while (i < 5) {
    array2[i] = array1[i];
    i = i + 1;
}
```

Array: Example - Reverse Order (simple)

- Read **5 numbers** and **print** them in **reverse order**

- Note for simplicity we are assuming scanf succeeds in reading an integer.
- A robust program would check that scanf returns 1 to indicate an integer read.
- The **constants** 4 & 5 below would be better replaced with a **#define**

```
int main(void) {
    int x[5], i, j;
    printf("Enter 5 numbers: ");
    i = 0;
    while (i < 5) {
        scanf("%d", &x[i]);
        i = i + 1;
    }
    printf("Numbers reversed are:\n");
    j = 4;
    while (j >= 0) {
        printf("%d\n", x[j]);
        j = j - 1;
    }
    return 0;
}
```

Array: Example - snap (simple)

- A simple program which reads integers and prints **snap** and exits if the **same number is read twice**

- Note the use of **return** to leave the main function and hence finish program execution

```c
#define MAX_NUMBERS 100000

int main(void) {
    int numbers[MAX_NUMBERS];
    int nNumbersRead, i;

    nNumbersRead = 0;
    while (nNumbersRead < MAX_NUMBERS) {
        printf("Enter a number: ");
        if (scanf("%d", &numbers[nNumbersRead]) != 1) {
            return 0;
        }
        i = 0;
        while (i < nNumbersRead) {
            if (numbers[i] == numbers[nNumbersRead]) {
                printf("Snap!\n");
                return 0;
            }
            i = i + 1;
        }
        nNumbersRead = nNumbersRead + 1;
    }
    printf("Sorry my array is full I have to stop!\n");
    return 0;
}
```

Array: Example - snap (version-02)

- A simple program which reads **integers** in the **range 1..99** and prints **snap** and exits when the same number is **read twice**

- Note for simplicity we are assuming scanf succeeds in reading an integer.
- A robust program would check that scanf returns 1 to indicate an integer read.

```c
#include <stdio.h>

#define LARGEST_NUMBER 99

int main(void) {
    int i, n, snap;
    int numberCounts[LARGEST_NUMBER + 1];
    i = 0;
    while (i < LARGEST_NUMBER) {
       numberCounts[i] = 0;
       i = i + 1;
    }
    snap = 0;
    while (snap == 0) {
        printf("Enter a number: ");
        scanf("%d", &n);
        if (n < 0 || n > LARGEST_NUMBER) {
            printf("number has to be between 0 and 99 inclusive\n");
        } else {
            numberCounts[n] = numberCounts[n] + 1;
            if (numberCounts[n] > 1) {
                printf("Snap!\n");
                snap = 42;
            }
        }
    }

    return 0;
}
```

Array: Example - frequency

```c
#define LARGEST_INTEGER 99

int main(void) {
    // the array element at index i
    // contains a count of how many times integer i has been seen
    int integer_counts[LARGEST_INTEGER + 1];

    // initialise all array elements to zero
    // this could also be done by changing the declaration to
    // int integer_counts[LARGEST_INTEGER + 1] = {0};

    int i = 0;
    while (i < LARGEST_INTEGER) {
       integer_counts[i] = 0;
       i = i + 1;
    }

    while (1) {
        int n;
        printf("Enter a number: ");
        if (scanf("%d", &n) != 1) {
            return 0;
        }

        if (n < 0 || n > LARGEST_INTEGER) {
            printf("number has to be between 0 and %d inclusive\n", LARGEST_INTEGER);
        } else {
            integer_counts[n] = integer_counts[n] + 1;
            printf("You have entered %d %d times\n", n, integer_counts[n]);
        }
    }

}
```

A simple program which reads integers in the range 1..99 and prints how many time each integer has been read.

Array: Example using function

- Find Max using a function,
  - array is passed to the function as a parameter

```c
#include <stdio.h>

int findMax( int a[], int size){

    int curMax = a[0];
    int j = 1;
    while(j < size){

        if( a[j] > curMax){
            curMax = a[j];
        }
        j++ ;   // j = j + 1
    }
    return curMax;
}

int main(void) {
    int a[5], i;
    printf("Enter 5 numbers: ");
    i = 0;
    while (i < 5) {
        scanf("%d", &a[i]);
        i = i + 1;
    }

    int maxValue = findMax(a, 5);

    printf("Max is %d \n ", maxValue);

    return 0;
}
```

Arrays of Arrays

- C supports arrays of arrays.
- Useful for multi-dimensional data.

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

printf("%d\n", matrix[1][1]);
```

Read a Two-dimensional Array

```
#define SIZE 42
...
int matrix[SIZE][SIZE];
int i, j;

i = 0
while (i < SIZE) {
        j = 0;
  while (j < SIZE) {
    scanf("%d", &matrix[i][j]);
    j = j + 1;
  }
  i = i + 1;
}
```

Print a Two-dimensional Array

```
...

while (i < SIZE) {
        j = 0;
  while (j < SIZE) {
    print("%d", &matrix[i][j]);
    j = j + 1;
  }
  printf("\n");
  i = i + 1;
}
```