# Arrays

A specific data structure stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of variables with the same data type.

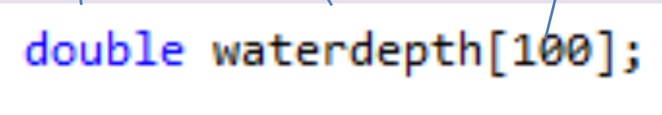
# Motivation

- You may need to define many variables of the same type.
  - Defining so many variables one by one is cumbersome.
- Probably you would like to execute similar statements on these variables.
  - You wouldn't want to write the same statements over and over for each variable.

# Declaration

To declare an array in C++, we should specify the type of the elements and the number of elements required by an array:

type arrayName [ arraySize ];



```
double waterdepth[100];
```

# Example #1

- Write a program that reads the grades of 350 students in a class and finds the average.

# Example #1 *(cont'd)*

- Sol$^{n}$:

```
#include <stdio.h>

int main()
{   int i, sum=0, grade; float avg;

    for (i=0; i<350; i++)
    {   scanf("%d", &grade);
        sum += grade;
    }
    avg = sum/350.0;
    printf("Average = %f\n",avg);
    return 0;
}
```

This was simple.
Since we don't need to store all values, taking the sum is enough. So, we don't need an array.

# Example #2

- Write a program that reads the grades of 350 students in a class and finds those that are below the average.

# Example #2 *(cont'd)*

- Sol$^n$ #1:

```
#include <stdio.h>

int main()
{   int i, sum=0, grade; float avg;

    for (i=0; i<350; i++)
    {   scanf("%d", &grade);
        sum += grade;
    }
    avg = sum/350.0;
    for (i=0; i<350; i++)
        if (grade<avg)
            printf("Below avg: %d\n",grade);
    return 0;
}
```

WRONG!

"grade" contains the score of the last student. You have already lost the previous 349 scores.

# Example #2 *(cont'd)*

- Sol$^n$ #2:

```
#include <stdio.h>

int main()
{   int i,sum,gr0,gr1,gr2,...,gr349;
    float avg;

    scanf("%d", &gr0);
    scanf("%d", &gr1);
    scanf("%d", &gr2);
    ...
    scanf("%d", &gr349);
    sum = gr0+gr1+gr2+...+gr349;
    avg = sum/350.0;
    if (gr0<avg)
       printf("Below avg: %d\n",gr0);
    if (gr1<avg)
       printf("Below avg: %d\n",gr1);
    if (gr2<avg)
       printf("Below avg: %d\n",gr2);
    ...
    if (gr349<avg)
       printf("Below avg: %d\n",gr349);
    return 0;
}
```

You cannot skip these with "..."
You have to repeat each of these statements 350 times.

# Example

- Sol$^n$ #3:

```
#include <stdio.h>

int main()
{   int i, sum=0, grade[350]; float avg;

    for (i=0; i<350; i++)
    {   scanf("%d", &grade[i]);
        sum += grade[i];
    }
    avg = sum/350.0;
    for (i=0; i<350; i++)
        if (grade[i]<avg)
            printf("Below avg: %d\n",grade[i]);
    return 0;
}
```

Defines an array consisting of 350 integer values.
In the definition, the value in the brackets is the number of elements (size).

This means the *i*$^{th}$ element of the array.
Here, the value in the brackets is the index, not the size.

# Arrays

- An array is a variable that is a collection of multiple values of the same type.
- Syntax:

```
type array_name[int_constant_value]={initializer_list};
```

- The size has to be of int type and must be a fixed value (i.e., known at compile time).
- You can define an array of any type (eg: int, float, enum student_type, etc.)
- All elements of the array have the same type.
- You cannot use the {} format for initialization after variable definition,

  ie, `int a[3]={5,8,2}` is correct, but

```
int a[3];
...
a={5,8,2}
```

is wrong.

# Arrays

- The index must of int type.

```
int k[5];
k[k[4]/k[1]]=2; /* Correct as long as k[4]/k[1] is nonnegative*/
k[1.5] = 3;     /* Error since 1.5 is not int */
```

# Arrays

- The lower bound must be nonnegative.

```
float m[8];    int i;


m[-2] = 9.2;  /* Syntax error */


i=-2;
m[i] = 9.2;  /* Run-time error */
```

# Initializing Arrays

- The elements of a local array are arbitrary (as all other local variables).
- The elements of a global array are initialized to zero by default (as all other global variables).

# Initializing Arrays

- You may initialize an array during definition as follows:

```
int array[5] = {10, 8, 36, 9, 13};
```

- However, you cannot perform such an initialization after the definition, i.e.,

```
int array[5];
array = {10, 8, 36, 9, 13};
```

is syntactically wrong.

# Initializing Arrays

- If the number of initializers is less than the size of the array:
  - initialization starts by assigning the first value to the first element and continues as such,
  - remaining elements are initialized to zero (even if the array was local)
- Eg: For the definition

```
int array[5] = {10, 8, 36};
```

the first 3 elements get the values 10, 8, and 36, respectively. array[3] and array[4] become 0.

# Initializing Arrays

- If the number of initializers is more than the size of the array, it is a syntax error.
- It is also possible to skip the size of the array iff the array is explicitly initialized.
  - In this case, the compiler fills in the size to the number of initializers.
  - Eg: For the definition

```
int array[ ] = {5, 9, 16, 3, 5, 2, 4};
```

the compiler acts as if the array was defined as follows:

```
int array[7] = {5, 9, 16, 3, 5, 2, 4};
```

# Example #3

- Read 100 integers and find the unbiased variance.

```
#include<stdio.h>

int main()
{   int X[100], i;
    float avg=0,var=0;

    for (i=0; i<100; i++)
    {   scanf("%d",&X[i]);
        avg += X[i];
    }
    avg /= 100;
    for (i=0; i<100; i++)
        var += (X[i]-avg) * (X[i]-avg);
    var /= 99;
    printf("variance:%f\n", var);
    return 0;
}
```

Unbiased variance of a sample is defined as

$$\frac{{}^{N}_{i=1}\;(X_i \qquad)^2}{N \quad 1}$$

# Example #4

- Find the histogram of the scores in Midterm 1.

```
#include <stdio.h>

int main()
{   int i, hist[101]={0}, score;

    for (i=0; i<350; i++)
    {   scanf("%d", &score);
        hist[score]++;
    }
    for (i=0; i<101; i++)
        printf("%d student(s) got %d\n", hist[i], i);
    return 0;
}
```

# Example #5

- Check if the array in the input is symmetric (eg: 8, 10, 6, 2, 6, 10, 8)

```
#include <stdio.h>
#define SIZE 10

int main()
{   int numbers[SIZE], i;

   for (i=0; i<SIZE; i++)
      scanf("%d",&numbers[i]);
   for (i=0; i<SIZE/2; i++)
      if (numbers[i] != numbers[SIZE-1-i])
         break;
   printf("It is ");
   if (i!=SIZE/2)
      printf("not ");
   printf("symmetric\n");
   return 0;
}
```

# Arrays Have Fixed Size!

- The size of an array must be stated at compile time.
- This means you cannot define the size when you run the program. You should fix it while writing the program.
- This is a very serious limitation for arrays. Arrays are not fit for dynamic programming.
  - You should use pointers for this purpose.

# Arrays Have Fixed Size!

- What you can do is to define very large arrays, making the size practically infinite → Wastes too much memory.
- Your program may exceed the maximum memory limit for the process.

# Arrays as Parameters

- Although you write like a value parameter, an array is always passed by reference (variable parameter).
  - Therefore, when you make a change in an element of an array in the function, the change is visible from the caller.

# Example #6

- Fill in an array of integer from input.

```
#include <stdio.h>

void read_array(int ar[10])
{   int i;
    for (i=0; i<10; i++)
        scanf("%d", &ar[i]);
}
int main()
{   int a[10], i;
    read_array(a);
    for (i=0; i<10; i++)
        printf("%d ",a[i]);
    return 0;
}
```

# Arrays as Parameters

- The size you specify in the function header is not important; you may even skip it.
- Eg:

```
void func(int arr[5])
{   int i;
    for (i=0; i<10; i++)
        arr[i]=i;
}
int main()
{   int a[10], i;
    func(a);
    for (i=0; i<10; i++)
        printf("%d ",a[i]);
    return 0;
}
```

- This will work without any problems though the function header is misleading.

# Example #7

- Fill in an array of integer from input.

```
#include <stdio.h>

void read_array(int ar[10])
{   int i;
    for (i=0; i<10; i++)
        scanf("%d", &ar[i]);
}
int main()
{   int a[10], i;
    read_array(a);
    for (i=0; i<10; i++)
        printf("%d ",a[i]);
    return 0;
}
```

# Example #8

- Write a function that inverts its array parameter.

```
void invert(int ar[10])
{   int i, temp;
    for (i=0; i<10; i++)
    {   temp=ar[i];
        ar[i] = ar[9-i];
        ar[9-i] = temp;
    }
}
```

What is wrong here?

This function changes nothing

# Example #9: Bubble Sort

- Sort the values in an array in ascending order.

```
#include <stdio.h>
void read_array(int ar[], int size)
{   int i;
    for (i=0; i<size; i++)
        scanf("%d", &ar[i]);
}

void print_array(int ar[], int size)
{   int i;
    for (i=0; i<size; i++)
        printf("%3d", ar[i]);
    printf("\n");
}

void swap(int *a, int *b)
{   int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
```

# Example #9: Bubble Sort *(cont'd)*

```
void bubble_sort(int ar[], int size)
{  int i, j;
   for (i = 0; i < size; i++)
      for (j = i + 1; j < size; j++)
         if (ar[i] > ar[j])
            swap(&ar[i],&ar[j]);
}

int main()
{  int ar[10];
   read_array(ar,10);
   bubble_sort(ar,10);
   print_array(ar,10);
   return 0;
}
```

# Example #10: Insertion Sort

```
void insertion_sort(int ar[], int size)
{
   int value, i, j;
   for (i=1; i<size; i++)
   {
      value = ar[i];
      j = i-1;
      while ((j>=0) && (ar[j]>value))
      {
         ar[j+1] = ar[j];
         j--;
      }
      ar[j+1] = value;
   }
}
```

# Example #11: Binary Search

- Given a sorted array, search for a specific value and return its index.

```
int binary_search(int A[], int number, int N)
{   int low = 0, high = N - 1, mid;

   while (low <= high)
   {   mid = (low + high) / 2;
      if (A[mid] == number)
         return mid;
      if (A[mid] < number)
         low = mid + 1;
      else
         high = mid - 1;
   }
   return -1;
}
```