

CS1101: Foundations of Programming

1D Arrays



Dept. of Computer Science & Engineering
Indian Institute of Technology Patna

Array

- Many applications require multiple data items that have common characteristics
 - In mathematics, we often express such groups of data items in indexed form:
 - $x_1, x_2, x_3, \dots, x_n$
- Array is a data structure which can represent a collection of data items which have the same data type (float/int/char/...)

Example: Printing number in reverse

3 numbers

```
int  a,b,c;  
scanf("%d",&a);  
scanf("%d",&b);  
scanf("%d",&c);  
printf("%d ",c);  
printf("%d ",b);  
printf("%d \n",a);
```

4 numbers

```
int  a,b,c,d;  
scanf("%d",&a);  
scanf("%d",&b);  
scanf("%d",&c);  
scanf("%d",&d);  
printf("%d ",d);  
printf("%d ",c);  
printf("%d ",b);  
printf("%d \n",a);
```

The problem

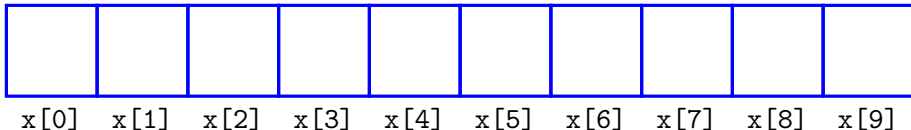
- Suppose we have 10 numbers to handle
 - Or 20
 - Or 100
- Where do we store the numbers? Use 100 variables?
- How to tackle this problem?
- Solution: Use arrays

Using arrays

- All the data items constituting the group share the same name

```
int x[10];
```

- Individual elements are accessed by specifying the index



Declaring arrays

- Like variables, the arrays used in a program must be declared before they are used
- **General syntax:** `type array-name[size];`
 - `type` specifies the type of element that will be contained in the array (`int`, `float`, `char`, etc.)
 - `size` is an integer constant which indicates the maximum number of elements that can be stored inside the array `int marks[5];`
 - `marks` is an array that can store a maximum of 5 integers

Array declaration

- **Examples:**

```
int x[10];  
char line[100];  
float points[90];  
char names[35];
```

- **If we are not sure of the exact size of the array, we can define an array of a large size**

```
int marks[50];
```

though in a particular run we may only be using, say, 10 elements

- **Array size should be constant**

Accessing array elements

- A particular element of the array can be accessed by specifying two things:
 - Name of the array
 - Index (relative position) of the element in the array
- In C, the index of an array starts from zero
- Example:
 - An array is defined as `int x[10];`
 - The first element of the array `x` can be accessed as `x[0]`, fourth element as `x[3]`, tenth element as `x[9]`, etc.
- The array index must evaluate to an integer between 0 and $n-1$ where n is the maximum number of elements possible in the array
$$a[x+2] = 25; \quad b[3*x-y] = a[10-x] + 5;$$
- Remember that each array element is a variable in itself, and can be used anywhere a variable can be used (in expressions, assignments, conditions,...)

Simple example

```
int main(){
    int data[10],i;
    for(i=0; i<10; i++)
        data[i] = i;
    i=0;
    while(i < 10){
        printf("data[%d]=%d\n", i, data[i]);
        i++;
    }
    return 0;
}
```

Storage for array in memory

- Starting from a given memory location, the successive array elements are allocated space in consecutive memory locations



- x — starting address of the array in memory
- k — number of bytes allocated per array element
- $x[i]$ — is allocated memory location at address $x + i * k$

Special operator: AddressOf (&)

- Remember that each variable is stored at a memory location with a unique address.
- Putting & before a variable name gives the starting address of the variable in the memory (where it is stored, not the value)
- Can be put before any variable (with no blank in between)

```
int a = 10;
```

```
printf("Value of a: %d, address of a: %u\n", a, &a);
```

Storage

```
int main(){
    int data[10], i;
    for(i=0; i<10; i++)
        printf("&data[%d]=%u\n",i,&data[i]);
    return 0;
}
```

&data[0]=227818948

&data[1]=227818952

&data[3]=227818956

...

Note: Memory addresses are being printed as unsigned integers using %u in printf. Address values can be negative if printed using %d.

Typically variables are allocated memory locations whose addresses are multiple of 4.

Reading the elements into an array?

- By reading them one element at a time

```
for(j=0; j<25; j++)  
    scanf("%d",&a[j]);
```

- The ampersand (&) is necessary
- The elements can be entered all in one line or in different lines

Example

```
int main(){
    const int MAX_SZ=100;
    int a[MAX_SZ], i, sum=0, size;
    scanf("%d",&size);
    for(i=0; i < size; i++){
        scanf("%d",&a[i]);
        sum += a[i];
    }
    printf("Sum: %d, Avg: %f", sum, ((float) sum)/size);
    return 0;
}
```

Array index

- The array index needs to be an integer and lies between 0 and $n - 1$ where n is the declared size of the array

```
a[x+2] = 54; // x int
```

```
a[3*x-2] = a[x+2] * a[x+1] ;
```

```
a[x] = a[a[x+2]] ;
```

- Each element of an array is a variable and can be used in expressions, assignments, conditions, etc.

Initialization of arrays

- **General form:** `type array_name[size] = { list of values };`

- **Examples:**

```
int marks[5] = {72, 83, 65, 80, 76};
```

```
char name[4] = {'A', 'm', 'i', 't'};
```

- **The size may be omitted. In such cases the compiler automatically allocates enough space for all initialized elements**

```
int flag[] = { 1, 1, 1, 0 };
```

```
char name[] = { 'A', 'm', 'i', 't' };
```

Warning

- In C, while accessing array elements, array bounds are not checked

- **Example:**

```
int marks[10];  
  
:  
  
marks[14] = -3;
```

- The above assignment would not necessarily cause an error
- Rather, it may result in unpredictable program results

How to print the elements of an array?

- By printing them one element at a time, one per line

```
for(j=0; j<25; j++)  
    printf("\n %d",a[j]);
```

- The elements are printed all in one line

```
for(j=0; j<25; j++)  
    printf(" %d",a[j]);
```

How to copy the elements of an array to another?

- By copying individual elements

```
for(j=0; j<25; j++)
```

```
    a[j]=b[j];
```

- The elements assignments will follow the rules of assignments expressions
- Destination array must have sufficient size

Note

- Let `int a[10], b[10];`
- You cannot use `=` to assign one array to another
 - `a = b; // not a valid one`
 - Also, `a, b` cannot be an l-value in any assignment
- You **cannot** use `==` to compare two arrays
 - `a == b; // valid syntax`
 - Though valid syntax but does not compare element by element
- You cannot directly `scanf` or `printf`: `printf("...",a);`

Example: Find the minimum of 10 numbers

Example: Find the minimum of 10 numbers

```
int main(){
    int i, min, a[10];
    for(i=0; i<10; i++)
        scanf("%d",&a[i]);
    min=a[0];
    for(i=1;i<10;i++){
        if(a[i] < min)
            min=a[i];
    }
    printf("Min=%d\n",min);
    return 0;
}
```

Alternate version - 1

```
const int size=10;
int main(){
    int i, min, a[size];
    for(i=0; i<size; i++)
        scanf("%d",&a[i]);
    min = a[0];
    for(i=1; i<size; i++){
        if(a[i] < min)
            min = a[i];
    }
    printf("Min=%d\n",min);
    return 0;
}
```

**Change only one line to
change the problem size**

Alternate version - 2

```
#define size 10
int main(){
    int i, min, a[size];
    for(i=0; i<size; i++)
        scanf("%d",&a[i]);
    min=a[0];
    for(i=1; i<size; i++){
        if(a[i] < min)
            min = a[i];
    }
    printf("Min=%d\n",min);
    return 0;
}
```

**Change only one line to
change the problem size**

Used define macro

#define macro

- `#define X Y`
- **Preprocessor directive**
- Compiler will first replace all occurrences of string X with string Y in the program, then compile the program
- **Similar effect as read-only variables (`const`), but no storage allocated**
- We prefer you use `const` instead of `#define`

Alternate version - 3

```
int main(){
    int i, min, a[100], n;
    scanf("%d",&n); /* no of elements */
    for(i=0; i<n; i++)
        scanf("%d",&a[i]);
    min=a[0];
    for(i=1; i<n; i++){
        if(a[i] < min)
            min = a[i];
    }
    printf("Min=%d\n",min);
    return 0;
}
```

Define an array of large size and use only the required number of elements

Computing CPI

```
const int nsub=6;
int main(){
    int grade[nsub],credit[nsub],i,gpsum=0,creditsum=0;
    double cpi;
    scanf("%d",&n); /* no of elements */
    for(i=0; i<n; i++)
        scanf("%d%d",&grade[i],&credit[i]);
    for(i=0; i<n; i++){
        gpsum += grade[i]*credit[i];
        creditsum += credit[i];
    }
    cpi=((double)gpsum)/creditsum;
    printf("CPI=%lf\n",cpi);
    return 0;
}
```

$$cpi = \frac{\sum_{i=1}^N c_i \times g_i}{\sum_{i=1}^N c_i}$$

Handling two arrays

Practice problems

- Write a C program that reads an integer n and stores the first n Fibonacci numbers in an array.

$$F_n = F_{n-1} + F_{n-2}, F_0 = 0, F_1 = 1$$

- Write a C program that reads an integer n and uses an array to efficiently find out the first n prime numbers.
- Read in an integer n , read in n integers and print the integer with the highest frequency.
- Read in an integer n , read in n numbers and find out the mean, median and mode.
- Read in two integers, m and n . Read m integers in array X and n integers in B . Find out the common elements in arrays X & Y and store it those another array Z .
- Read in two integers, m and n . Read m integers in array X and n integers in B . Find out the elements that are either in X or Y and store those in another array Z .