

ARRAYS

An array is a variable that can store multiple values with the same data type. For example, if you want to store 10 integers, you can create an array for it. This is how to do it.

```
int num[10];
```

To illustrate:

0	1	2	3	4	5	6	7	8	9
4	6	12	8	3	4	2	7	8	6

Create an Array

How to declare an array?

In general, to declare an array

```
datatype arrayName[arraySize];
```

For example,

```
int num[10];
```

Where: num is the name of the array,

10 is the maximum number of integers that can be stored,

int is the data type of those values.

To illustrate:

0	1	2	3	4	5	6	7	8	9

Ten memory locations are allocated for array num. Num can only accept integer values.

Another example,

```
char name[30];
```

Where name is an array of thirty characters.

You can declare an array and initialize.

Here are few examples,

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

Each value is assigned to each location. Assignment is done by position which means, 1 is assigned to index 0, 2 to index 1, 3 to index 0 and so on.

0	1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9	10

```
int num[10] = {0};
```

0 is assigned to all memory locations in the array. 0 is assigned in index 0. The rest is automatically initialized to 0.

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

```
int num[10] = {1,2,3};
```

The above example is similar to the previous one. 1 is assigned to index 0, 2 to index 1 and 3 to index 0. The rest is initialized to 0. *In C, if the initializer list has a lesser number of elements to its size, the rest is initialized to 0.*

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

You can also declare and initialize without the size.

For example,

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

Since size is not defined, the array will be created based on the number of initializers.

How to access array elements?

You can use array subscript (or index) to access any element stored in array. The subscript should only be an integer or an integer expression. Subscript starts with 0, which means `num[0]` represents the first element in the array `num`. In general, `num[n-1]` can be used to access `n`th element of an array, where `n` is any integer number.

For example,

```
int num[10];

num[0]    // first element of num array

num[9]    //last element of num array
```

To access an element in array, just specify its name and the subscript.

For example, if you want to assign 5 to index 2, then you do it like this,

```
num[2]=5;
```

To display element 3, then you do it like this,

```
printf("%d", num[3]);
```

Input and Output an Array

Here is how to input elements to the entire array: use the for loop, starting from index 0 to `size-1` and input using `scanf` if it is an array of numbers.

For example,

```
for(i=0;i<size;i++)  
    scanf("%d",&num[i]);
```

Here is how to display the elements of the array: use the for loop, starting from index 0 to `size-1` and display using `printf`.

For example,

```
for(i=0;i<size;i++)  
    printf("%d",num[i]);
```

Take note that when accessing elements of an array using a loop, the index is always the counter variable.

Traverse an Array

To traverse an array, use the for loop, starting from index 0 to `size-1` and do the task asked.

For example: To compute the sum of the elements of the array,

```
sum = 0;  
for(i=0;i<size;i++)  
    sum += num[i];
```

Here is an example of a program using arrays:

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

int main(int argc, char *argv[]) {
    int num[SIZE];
    int i, sum;
    //input
    for(i=0; i<SIZE; i++)
        scanf("%d", &num[i]);
    //output
    for(i=0; i<SIZE; i++)
        printf("%d", num[i]);
    //traverse
    sum=0;
    for(i=0; i<SIZE; i++)
        sum+=num[i];
    printf("Sum = %d", sum);
    return 0;
}
```

Memory Representation of an Array

Here is the memory representation of an array:

num	num[0]	num[1]	num[2]	num[3]	num[4]	num[5]	num[6]	num[7]	num[8]	num[9]
88824	1	2	3	4	5	6	7	8	9	10
88820	88824	88828	8882C	88830	88834	88838	8883C	88840	88844	88848

As shown on the illustration, elements are in the indexed memory locations. However, there is the first value in the array name that is the address of its first element. Thus the array name is a pointer. Since it holds the address of its first element, it constantly points to its first element.

How to pass arrays to functions?

Arrays are passed to functions by reference. What is actually passed is the address of its first element by passing the name of the array that contains the address of its first element as discussed on the topic “Memory Representation of an Array”. By convention, the array size is also passed by value.

Here is an example:

```
main.c  array.h  array.c
1  void input(int num[], int size);
2  void display(int num[], int size);
3  int computeSum(int num[], int size);
4
```

```
main.c  array.h  array.c
1  #include<stdio.h>
2
3  void input(int num[], int size){
4      int i;
5      for(i=0;i<size;i++)
6          scanf("%d",&num[i]);
7  }
8
9  void display(int num[], int size){
10     int i;
11     for(i=0;i<size;i++)
12         printf("%d",num[i]);
13 }
14
15 int computeSum(int num[], int size){
16     int i, sum;
17     sum=0;
18     for(i=0;i<size;i++)
19         sum+=num[i];
20     return sum;
21 }
22
```

```
main.c  array.h  array.c
1  #include <stdio.h>
2  #include <stdlib.h>
3  #define SIZE 10
4
5  int main(int argc, char *argv[]) {
6      int num[SIZE];
7      //input
8      input(num, SIZE);
9      //output
10     display(num, SIZE);
11     //traverse
12     printf(" = %d\n",computeSum(num, SIZE));
13     return 0;
14 }
```

To illustrate:

In the main program, this is num:

0	1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9	10

In input(), display() and sum, this is num:

Num

Parameter num is just a reference to num in main.

More on Array Traversal

Let us define more examples for array traversal. The following are count functions.

```
int countPos(int num[], int size); //counts the number of positive integers
int countNeg(int num[], int size); //counts the number of negative integers
int countEven(int num[], int size); //counts the number of even integers
int countOdd(int num[], int size); //counts the number of odd integers
```

```

int countPos(int num[], int size){
    int i, count=0;
    for(i=0;i<size;i++){
        if(num[i]>=0)
            count++;
    }
    return count;
}

int countNeg(int num[], int size){
    int i, count=0;
    for(i=0;i<size;i++){
        if(num[i]<0)
            count++;
    }
    return count;
}

int countEven(int num[], int size){
    int i, count=0;
    for(i=0;i<size;i++){
        if(num[i]%2==0)
            count++;
    }
    return count;
}

int countOdd(int num[], int size){
    int i, count=0;
    for(i=0;i<size;i++){
        if(num[i]%2!=0)
            count++;
    }
    return count;
}

printf(" = %d\n",countPos(num, SIZE));
printf(" = %d\n",countNeg(num, SIZE));
printf(" = %d\n",countEven(num, SIZE));
printf(" = %d\n",countOdd(num, SIZE));

```

Let's have another set of examples. They are search functions.

```

int findItem(int num[], int size, int item); //returns 1 if item is in the array; 0 if otherwise
int findItemAt(int num[], int size, int item); //returns the index location of item; -1 if item does not exist
                                           //assuming elements are distinct

```

```

int findItem(int num[], int size, int item){
    int i, ans = 0;
    for(i=0;i<size;i++){
        if(num[i]==item){
            ans=1;
            break;
        }
    }
    return ans;
}

int findItemAt(int num[], int size, int item){
    int i, ans = -1;
    for(i=0;i<size;i++){
        if(num[i]==item){
            ans=i;
            break;
        }
    }
    return ans;
}

printf(" = %d\n",findItem(num, SIZE,5));
printf(" = %d\n",findItemAt(num, SIZE,5));

```

Practice Exercise (Ungraded)

Define the following functions:

```

int findMin(int num[], int size); //finds and returns the smallest element
int findMax(int num[], int size); //finds and returns the largest element

```

Insert an Element in an Array

Initially, an array is always empty. To fill it out, you add elements one by one. For the following examples, size will no longer be passed as a parameter but the current number of its elements. We will name the variable `count`. Variable `count` will be initialized to 0 in main.

Inserting an element in an array depends on the location where to insert.

1. Insert at the end

Function `add` inserts `item` at the end of the list. After `item` is inserted, `count` is incremented by 1. Thus, `count` is passed by reference.

```

void add(int num[], int *count, int item); //inserts item at the end

void add(int num[], int *count, int item){
    num[(*count)++] = item;
}

```



```

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

int main(int argc, char *argv[]) {
    int num[SIZE];
    int count = 0;
    add(num, &count, 5);
    add(num, &count, 10);
    add(num, &count, 15);
    display(num, count);
    return 0;
}

```

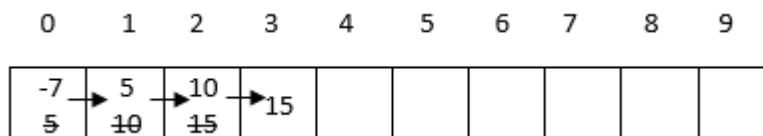
```
5 10 15
```

2. Insert at index 0

Function `addFront` inserts `item` as the first element in the list. After `item` is inserted, `count` is incremented by 1. Thus, `count` is passed by reference.

To insert an element at the front, repeatedly move the elements starting from the last one until index 0.

To illustrate, assuming to insert -7:



Two versions can be derived from this algorithm.

```

void addFrontV1(int num[], int *count, int item); //inserts item at index 0
void addFrontV2(int num[], int *count, int item); //inserts item at index 0

```

Algorithm for version 1:

Using variable `i` as the counter variable,

1. Initialize `i` with the value of `count` which is currently 3.
2. Repeatedly assign the value of `num[i-1]` to `num[i]`.
3. Decrement the value of `i`.
4. `i` has to stop at index 1.
5. Insert `item` to index 0.
6. Increment `count` by 1.

Definition:

```

void addFrontV1(int num[], int *count, int item){
    int i;
    for(i=*count;i>0;i--)
        num[i] = num[i-1];
    num[0] = item;
    (*count)++;
}

```

Algorithm for version 2:

Using variable `i` as the counter variable,

1. Initialize `i` with the value of `count-1` which is currently 2.
2. Repeatedly assign the value of `num[i]` to `num[i+1]`.
3. Decrement the value of `i`.
4. `i` has to stop at index 0.
5. Insert `item` to index 0.
6. Increment `count` by 1.

Definition:

```

void addFrontV2(int num[], int *count, int item){
    int i;
    for(i=(*count)-1;i>=0;i--)
        num[i+1] = num[i];
    num[0] = item;
    (*count)++;
}

```

3. Insert at whatever the value of `pos`

The only difference of the algorithm of `addAt` from `addFront` is the position where to insert. For `addFront`, the location is specifically index 0, while for `addAt`, the location is whatever the value of `pos`. `addAt` can insert a value anywhere as long as it is within the bounds of the current elements in the array. Thus, `addAt` is the generic algorithm for insert.

Practice Exercise (Ungraded)

Define the two versions of `addAt`:

```

void addAtV1(int num[], int *count, int item, int pos); //inserts item at pos
void addAtV2(int num[], int *count, int item, int pos); //inserts item at pos

```

Delete an Element in the Array

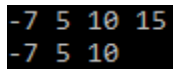
Similar to the algorithm for insert, deleting an element in an array depends on the location where to remove the data.

When deleting an element, note that the last element remains. So to not include the last element after moving the data, simply decrement the value of `count` by 1.

1. Delete the end element

To delete the last element, simply not count it by decrementing the value of `count` by 1.

```
void deleteEnd(int *count);  
  
void deleteEnd(int *count){  
    (*count)--;  
}
```



-7 5 10 15
-7 5 10

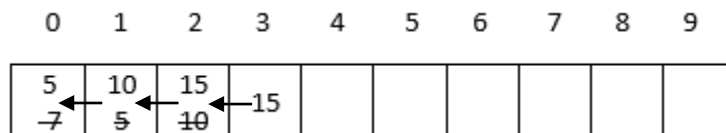
As shown on the above figure, 15 is deleted from the list.

2. Delete the first element

Function `deleteFront` deletes the first element in the list. After first element is deleted, `count` is decremented by 1. Thus, `count` is passed by reference.

To delete the first element, repeatedly move the elements starting from the first element adjacent to the element to be deleted until the last one.

To illustrate:



Two versions can be derived from the algorithm.

```
void deleteFrontV1(int num[], int *count);  
void deleteFrontV2(int num[], int *count);
```

Algorithm for version 1: Using variable `i` as the counter variable,

1. Initialize `i` with the value of the first index that is 0.
2. Repeatedly assign the value of `num[i+1]` to `num[i]`.
3. Increment the value of `i`.
4. `i` has to stop at `count-2`.
5. Decrement `count` by 1.

Definition:

```
void deleteFrontV1(int num[], int *count){  
    int i;  
    for(i=0; i<(*count)-1; i++)  
        num[i]=num[i+1];  
    (*count)--;  
}
```

Algorithm for version 2:

Using variable `i` as the counter variable,

1. Initialize `i` to 1.
2. Repeatedly assign the value of `num[i]` to `num[i-1]`.
3. Increment the value of `i`.
4. `i` has to stop at `count-1`.
5. Decrement `count` by 1.

Definition:

```
void deleteFrontV2(int num[], int *count){
    int i;
    for(i=1; i<*count; i++)
        num[i-1]=num[i];
    (*count)--;
}
```

3. Delete the element at pos

The only difference of the algorithm of `deleteAt` from `deleteFront` is the position where to delete. For `deleteFront`, the location is specifically index 0, while for `deleteAt`, the location is whatever the value of `pos`. `deleteAt` can delete a value anywhere as long as it is within the bounds of the current elements in the array. Thus, `deleteAt` is the generic algorithm for delete.

Practice Exercise (Ungraded)

Define the two versions of `deleteAt`:

```
void deleteAtV1(int num[], int *count, int pos);
void deleteAtV2(int num[], int *count, int pos);
```

GRADED EXERCISE

- Open the Airline Reservation System file