



Arrays (some stuff)

In general

`elemtype aname[alength];`

- Block of memory of length $S * alength$ is allocated,
 S = size in bytes of a single `elemtype` variable.
- `aname` = starting address of zeroth element = address of allocated block.
- Value of `aname` cannot be changed.
- Type of `aname`: `elemtype *`
- Type of `aname[i]` : `elemtype`

Example

- What does the following code do?

```
int q[4];
int *r;
r = q;
r[3] = 5;
cout << q[3] << endl;
cout << r[3] << endl;
```

- The array `q` is allocated in memory.
- Variable `r` is created.
- `q` = address of the zeroth element of the array is placed in `r`.
- Because `r`, `q` have the same value, `r[3]`, `q[3]` also denote the same variable.

Character arrays with initialization

```
char n1[20]="Ajanta", n2[]="Ellora";
```

n1 will be created with length 20, and initialized as follows.

- 'A', 'j', 'a', 'n', 't', 'a' will get stored in n1[0] through n1[5].
- Finally '\0' will get stored in n1[6].
- The elements n1[7] through n1[19] will not be initialized.

n2 will be created of length 1 + the length of the string "Ellora".

- It will also be initialized to the string "Ellora" followed by '\0'.

A screenshot of a Linux desktop environment with a purple and red background. A text editor window titled "Mr.Checka.cpp" is open, showing C++ code. The code includes `<iostream>`, uses the `std` namespace, and defines a `main` function. Inside `main`, an array `b` of size 10 is initialized with the values {1, 2, 3, 5}. The code then prints the array's content and a pointer to its first element. The status bar at the bottom of the editor indicates "C++", "Tab Width: 8", "Ln 6, Col 17", and "INS".

```
1 #include<iostream>
2
3 using namespace std;
4
5 int main(){
6     int b[10]={1, 2, 3, 5};
7     cout<<"b"<<endl;
8     int q=12;
9     int *p=&q;
10    cout<<p[0]<<endl;
11 }
```

Here you can clearly see 2 things-

1. You can denote `*p` by `p[0]` for any general pointer (Need not even be an array)
BTW `p[1]` will give you the int that occupies the space just after `*p` or `p[0]` ends as computer will read `p[n]` as `*(p+sizeof(datatype) × n)`
2. `int b[10]={1,2}` is as valid as `int b[]={1,2}` but-
`int b[];`
isn't valid

BTW the `'\0'` at the end of a string/ character array is called "sentinel"

Also `cout<<char_array` prints the string instead of the address as you would normally assume

Also also you can do

```
char arr[50]="Yes";
```

but you can't do

```
char arr[50]="Yes";  
arr="No"; //Thus this means that you can only equate char_array to a string during Declaration  
//But never otherwise i.e. aisehi assignment not allowed
```

Character string constant

- Character string constant: text enclosed within double quotes
- Examples: "India", "C++ is nice", "Please give your name"
- How C++ implements these:
 - The string is stored in some place in memory.
 - A null terminator is used.
 - It resembles a char array!
 - C++ denotes a character string constant by the address where it starts
- When you pass a character string constant, the address where it is stored in memory is passed.
 - Just like any array!
 - Its type is `const char *`



While passing functions if use const char, then the string becomes const and we cannot change it at all-

```
void f(int arr[], char *muslin, const char nope){ //We could also have used char muslin[]
    muslin[0]='N';
    muslin[1]='o';
    muslin[2]='\0';
    //nope[0]='J' will give errors as we have defined the string as const
    return;
}

int main(){
    int arr[50]={1,2,3,4};
    char nope="Papaji";
    //btw char muslin[50]=nope is illegal
    char muslin[20]="Yes daadi";
    cout<<muslin<<endl;
    f(arr, muslin);
    cout<<muslin;
    return 0;
}
```

In 2D arrays (or multidimensional arrays in general, you have to hardcode all but one... woh re while passing. Like dekh le

```
//void death(int arr[][], char mamu[][][]) gives error
void death(int arr[][10], char mamu[][12][20]){
    ...
}
```

Compiler allots memory to 2D arrays row-wise i.e.

```
int arr[3][4];
```

This will be stored as arr[0][0], arr[0][1]...arr[0][3], arr[1][0]....arr[1][3].....arr[2][3]

1 / 1 points

1 / 1 attempts

Consider a two-dimensional array `int pqr[5][5]` and the element `pqr[0][0]` is stored at the address `0x52A12F0`. What is the address of the element `pqr[2][3]` in the above arrangement?

Consider `int` to be size 4 bytes.

- ☐ `0x52A12F0`
- ☐ `0x52A1324`
- ☐ `0x52A1358`
- ☐ `0x52A138C`

Submit

Hint

Answer

`0x52A1324`

You can hard-code initialise 2D arrays as such-

```
int arr[2][3]={{1,2,3},{4,5,6}}
//You can even write it as {1,2,3,4,5,6} as it dgaf
```

```
char countries[3][20] = {"India", "Nepal", "China"};
```

- Creates array `countries` with 3 rows each of 20 characters.
- 2d array = collection of 1d arrays.
- Thus `countries[i]` = array consisting of `countries[i][0]...countries[i][19]`
- Row `i` of array is initialized using `i`th character string in rhs.
- As usual, each character string implicitly includes a `'\0'` following it.

```
//In above code if we did this-
cout<<countries[1];
//Output- Nepal
```