Day:

Date: / /

Arrays:

We have worked with variables but does can only store one value so to store more than one value we have Arrays.

Delaring an array:

- An array is declared similarly to a variable but size of the array is given.

```
Datatype  array-name [ size ];
```

eg

```
int  day [6];
```

now this is an array which can hold 6 values.

Different ways to declare:

1) The simple:

```
int  day [6];
```

Day: Date: / /

2) User input:

```
int size;
cin >> size

int arr[size];
```

3) declaring size before:

```
int size = 5;
int arr[size];
```

(2) and (3) are correct but are not preferrable bcz:

size is a variable and can be changed anywhere else in the code, but this would confuse us as we would though size of array changed.

Solution:

(2)

```
int temp
cin >> temp;
const int size = temp;
int arr [size];
```

(3)

```
const int size = 5;
int arr [size];
```

Yousuf

Day: Date: / /

4) Using const for arrays

```
const int num[2];   ✗
```
This will give error.

- When we use const to make an unmodifiable array it needs to be initialized.

```
const int num[2] = {1, 2};   ✓
```
allowed.

- This array cannot be modified.

```
const int num[2] = {1,};   ✓
```
This is also allowed, but the second element will be 0.

5) No Size

```
const int num[];   ✗ not allowed
const int num[] = {0, 1, 2};   ✓ allowed
```

- This will consider 3 as size.

Day: Date: / /

Important points related to initializing:

1) Only declaring.

int arr[5]; ✓

This will store 0,0,0,0,0. Most compilers store 0 if value not initialized.

2) initializing but will more element than size.

int arr[6]={1,2,3,4,5,6,7} ✗

error
Not allowed.

REMEMBER: This is not allowed but if we give value e.g for arr[7] even though size is 6 it will be stored .. learn Later in bound check.

* element are written in {} curly brackets.

Yousaf

Day: Date: / /

Bound Checking:

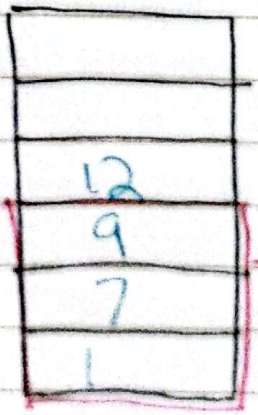
- C++ is a very efficient language and it avoids processes like bound checking e.g if we declare an array with size 5 and then we cin 6 items into it it will be allowed. BUT HOW? and what is the problem?

- For this we will go back to how an array is made:

Array:

In ram, when we form an array, a certain place according to size is given to the array eg

eg int arr[3];



This area is now assigned to the array. These 3 blocks are of array now.

But e.g if we cin 4 items: 1, 7, 9, 12

Day: Date: / /

- Now the 3 items do belong in the array and are protected but the 4th one (12) is accessible but not protected.

Not protected?

- By not protected I mean that the memory address of the 4th one might be assigned to any other array or even variables so it might be changed even though we did not change it.

Remember indexes start with 0 not 1 and end at (size-1).

Day: Date: / /

Datatypes:

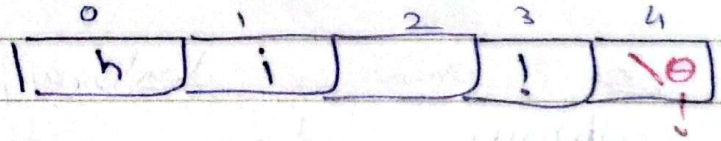
- like variables, array are also of different datatype.

1) char datatype:

This works the same way as int datatype.

```
char arr[5] = {'h', 'i', ' ', '!', '\0'}
```

* In char always remember to leave one last element space - Why.



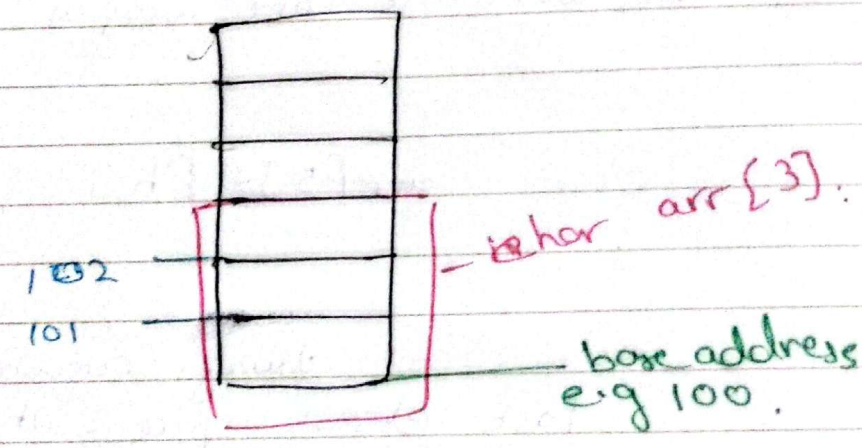
At the end of every char array there should be a null character the notify that the array has ended.

- It will work fine even without it but it should be added for good measure.

Day: Date: / /

- In char each element or character gets ONE BYTE i.e one memory address.

How does an array work:



When we declare an array it always consider a base address and with indexing it moves up for.
example for [0] = 'base address = 100
for [1] = '100+1 = 101

- Now char array only takes one byte so when we say size is 3 we mean that the bytes occupied are 3. very important

Page: Date: / /

Important points in char arrays:

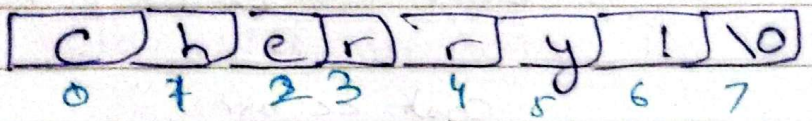
1) We can initialize char by a string!!

~~except~~ char arr[] = "Cherry!"; allowed.

what will happen?

curly bracket not necessary

only ~~this~~ for this



as string is made from characters it is allowed and will be stored like above.

2) what if we don't ~~write~~ leave space for null character:

- ~~b~~ - char array are the only arrays that can be printed by cout.!!!

~~- To print arrays~~

- char can be cout simply e.g.

char arr[7] = "cherry!";

cout<<arr; // only for char.

Journal

Day: Date:

This will give the elements in char

* But if there is no space for '\0' then a few extra characters are printed. Because array doesn't know when to stop.

eg

```
char arr[2] = {'A', 'B'};

cout << arr;
```

output: AB]K 123 :- (unknown)

* Curly bracket will always be used but not required ONLY if we add a string to char array. ie.

char arr[] = "Cherry";

* char arr[6] = "Cherry"; ✗ error as no place for \0 (string always end with \0.)

char arr[6] = {'C', 'h', 'e', 'r', 'y'} ✓ Allowed but will print garbage at end.

But if we use something like arr[size], size declared before. There will be no error. as we can add as many as we want, even greater than size

Yousuf

Day: Date: / /

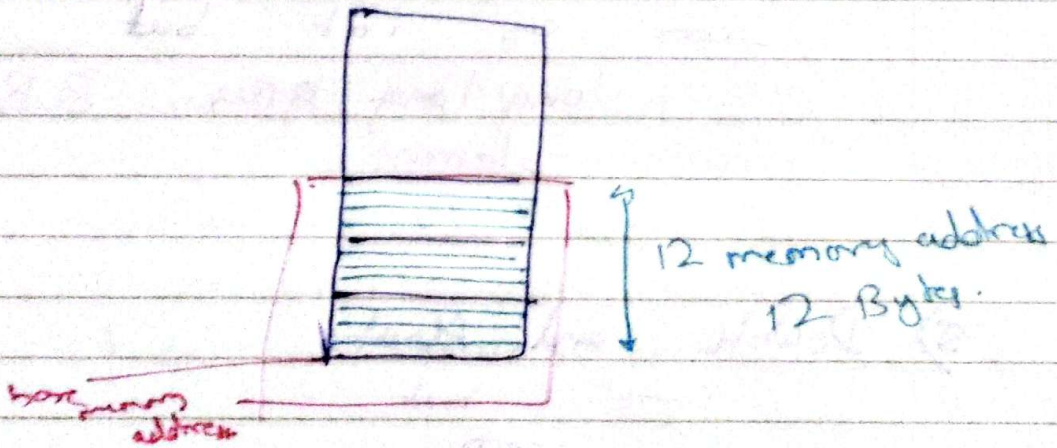
1) Int Datatype:

- Stores integers.

- Remember each int has 4 BYTES. so in an int array when we write

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

The size in bytes is 3 x 4 = 12 Bytes.
How?



- Doesn't have any null character

- Cannot be cout simply ie cout << arr;
This will output the base memory address.

Yousaf

Day: Date: / /

3) String arrays:

- For now just know that we can store strings in these arrays.

string arr[] = { "Haider", " ", "Ramzan" }

4) long int and long long int:

Same as int but

long long gives 8B to any element.

5) Double and float:

same but

Double → 8B.
float → 4B

6) short int:

2B.

Yousaf

Day: Date: / /

Accessing array element:

- Now we have stored values in array but how to access them.

AlWays remember:

Arrays start with index = 0
and then move forward 0, 1, 2, 3, 4 ... (x-1)

Accessing:

~~char~~

```
char arr [3] = {'H', 'A', 'D'};
                 0    1    2

cout << arr [0]; // output H
cout << arr [1]; // output A
cout << arr [2]; // output D.
```

output:

HAD

~~Header~~

* Even cout << arr [200] is allowed.
But will give some random answer from the RAM.

Yousaf

Printing arrays:

array cannot be simply outputted with cout. except char.

so we use loop:

```
1   int size = 5;
2   int arr[size] = {12, 13, 27, 28, 19};
3
4   for(int i = 0; i < size; i++) {
5       cout << arr[i];
6   }
```

- This code is used to print the array and we can also use it for inputs also. by replacing cout with cin.

* always start loop with i=0 as index starts from 0.

Day: Date: / /

Increment and decrement:

- we can use increment (++) or decrement (--) with arrays too BUT:

* They will not work for whole array (ERROR). e.g ++arr;
* They will work for particular element e.g ++arr[2];

Example of one D arrays usage:

1) Linear search:

- In we have an array of e.g ID's, but want to check if a particular ID is in there or not we can use:

```
int IDs[10] = { ..... };
int id;
cin >> id; // id to check

for (int i=0; i<10; i++) {
    if (id == IDs[i]) {
        cout << "ID is There" << endl;
    }
}
```

Yousaf

Day: Date: / /

Q Finding maximum number in an array:

```
int main(){

    const int size = 5;
    int arr[size] = {1, 3, 7, 6, 4};

    int max = arr[0]; // Starting with the first number as maximum.

    for(int i=0; i<size; i++){
        if (arr[i] > max){
            max = arr[i]
        }
    }

    Cout << max << endl;

return 0;
}
```

can be any number but must be from the array.

We can also count the number of maximum elements by adding another loop after the loop, which will check the max value with each index and increase a counter if it is in the index.

- You can also find min by changing greater than sign to less than sign

Yousaf

Day: Date: / /

Comparing arrays:

- We can only compare index of an array and compare 2 arrays simply like:

```
arr1 == arr2 ;  ✗
```

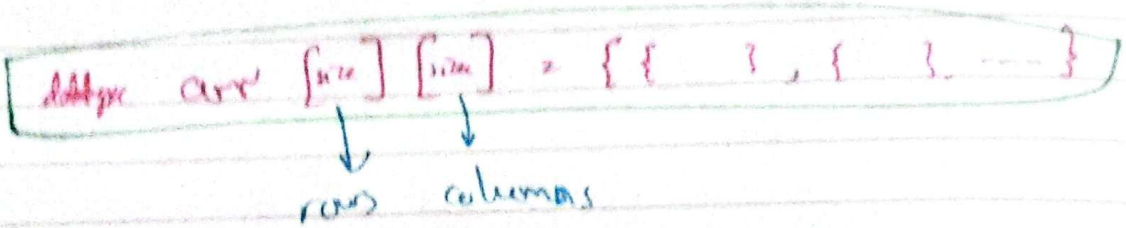
This will check the base addresses of both which will never be same. and always give false as a result.

Correct way to compare 2 whole arrays:

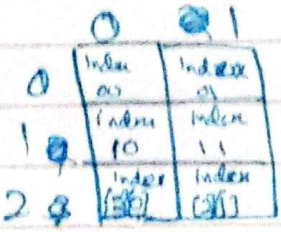
There are many ways but I am using a bool flag.

```
int main(){
    int arr1[5] = {1,2,3,4,5};
    int arr2[5] = {1,2,3,4,5};
    bool arrequal = true; //flag bool.
    for(int i=0; i<5; i++){
        if(arr1[i] != arr2[i]){
            arrequal = false;
        }
    }
if(arrequal){cout << "The arrays are equal".endl;}
else {cout << "Not equal arrays";}
}
```

2D Arrays:



e.g int arr [3] [2] ;



- 2D arrays are like matrices.

Size:

- Now size of array also changes. e.g int [2][3] we now have 6 places or elements so size = 6 × 4 = 24 Bytes

so 2 × 3 × 4 Bytes = size.

Page: Date: / /

Initializing:

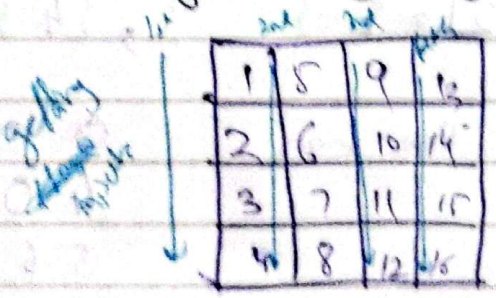
```
int arr[2][2] = {{1,2}, {1,3}};
```

1st row, 2nd column; 1st row; 2nd row

```
1 2
1 3
```

Examples

1) Printing (using nesting loops).
2) input.
3) ~~printing each~~ getting inputs for/in columns ie:



change outer loop for and make it for columns while inner loop for rows.

4) Summing the elements.

I have written all these codes on laptop.

Yousaf

Day: Date: / /

Intializing of 2D arrays:

e.g

1) int arr[3][2] = { {1,2}, {3,4}, {5,6} };

This means

1	2
3	4
5	6

2) int arr[3][2] = { { }, {3,4}, {5,6} };

This means

0	0
3	4
5	6

3) ~~int~~ int arr[3][2] = { { }, {4}, {5,6} };

This means

0	0
4	0
5	6

4) Int arr[3][2] = { 1, 2, 3, 4, 5, 6 }; allowed.

This means

1	2
3	4
5	6

~~Rows are~~ given priority, and filled.

Yours

Day: Date: / /

5) int arr [3][2] = {1,2,3};

This was

	0	1
0	1	2
1	3	0
2	0	0

6) Checking if i do,

int arr [3][1] = {1,2,3}

1
2
3

and i cout << arr[1];

This will give the address and not the value 1.