ARRAYS

“A DATA STRUCTURE THAT HOLD MULTIPLE VALUES

AND THE VALUES ARE ACCESSED BY AN INDEX NUMBER “

Arrays is that they can only contain values of the same data types

---

```
#include <iostream>
int main() {
 std::string cars [ 3 ] ;                    //here the 3 is the size of the array

cars[0]= "Corvette";
cars[1]= "Mustang";
cars[2]= "Camry";

std::cout << cars[ 0 ] << '\n';
std::cout << cars[ 1 ] << '\n';
std::cout << cars[ 2 ] << '\n';
return 0;
}
```

```
#include <iostream>
using namespace std;

int main(){
char letter [ ] = {'F','U','C','K'};
cout<<letter[0]<<endl;
```

cout<<letter[1]<<endl;

cout<<letter[2]<<endl;

cout<<letter[3]<<endl;

return 0;

}

Sizeof ( ) operator

Use the sizeof() operator to confirm the size in byte of a variable, data types ,class,objects,etc on the system..

(A byte is the basic unit used in low-level programming for tasks like copying or comparing raw memory.Data is often read and written in terms of bytes when working with the binary files)

```
#include <iostream>
 int main() {
// sizeof() = determines the size in bytes of a: // variable, data type, class, objects, etc.
 std::string name = "B KIDDY";
double gpa = 2.5;
 char grade = 'F';
 bool student = true;
 char grades[] = {'A', 'B', 'C', 'D', 'F'};
std::string students[ ] = {"Spongebob", "Patrick", "Squidward", "Sandy"};
std::cout << sizeof(grade) << " bytes\n";
std::cout << sizeof(students)/sizeof(std::string) << " elements\n"; return 0;
}
```

Here we use the division to get how many elements are present . Size of the elements present.

Output

1 bytes

4 elements

ARRAY ITERATION / REPETITION

We can use for loop instead of using ..cout<<.....cout<<.....

```
#include <iostream>

int main() {

char grades[ ] = {'A', 'B', 'C', 'D', 'F'};

for(int i = 0; i < sizeof(grades)/sizeof(char); i++){
```

here we use division because we can add up or reduce the array, or we can use 5 instead of this here..!!

```
std::cout << grades[ i ] << '\n';
```

here,we use grade[i] to print the elements

```
 }

return 0; }
```

'FOR EACH' LOOP

We use the standard for loop, we have 3 statements ,but we can use for each loop ,there's less syntax than a typical for loop , for each loop we start at the begining and go to the end there's less flexibility.

```
#include <iostream>

int main() {

int grades[ ] = {65, 72, 81, 93};

for(int grade : grades){
```

we can use any name after int (here it's grade (similar)) : then the iterable data set (here,grades)

```
std::cout << grade << '\n';
```

```
}
 return 0;
}
```

PASS ARRAYS TO THE FUNCTIONS

```
#include <iostream>
double getTotal(double prices[ ], int size);
 int main()
 {
double prices[ ] = {49.99, 15.05, 75, 9.99};
int size = sizeof(prices)/sizeof(prices[0]);
double total = getTotal(prices, size);
std::cout << "The total is: $" << total;
return 0;
}
double getTotal(double prices[ ], int size) {
double total = 0;
for(int i = 0; i < size; i++)
{ total += prices[i];
}
return total;
}
```

here we can use sizeof(double) instead of sizeof(prices[0])

we can't directly use sizeof(prices)/sizeof(prices[0]) instead of size .because this function decays into pointer,that points to the address of where the array begins.this function has no idea how big this array is anymore,that's why we calculate the size.

Search an array

```
#include<iostream>
int searchArray(std::string array[], int size, std::string element);
int main() {
std::string foods[] = {"pizza", "hamburger", "hotdog"};
int size = sizeof(foods)/sizeof(foods[0]);
int index;
std::string myFood;
std::cout << "Enter element to search for: " << '\n';
std::getline(std::cin, myFood);
index = searchArray(foods, size, myFood);
 if(index != -1){
std::cout << myFood << " is at index " << index;
 }
else{
std::cout << myFood << " is not in the array";
 }
return 0;
 }
int searchArray(std::string array[], int size, std::string element){
for(int i = 0; i < size; i++){
if(array[i] == element){
return i;
}
}
 return -1;      -1 means that something wasn't found
}
```

output
Enter element to search for : hotdog
Hotdog is at index 2

SORT THE ARRAY

```
#include <iostream>
void sort(int array[], int size);
int main() {
int array[] = {10, 1, 9, 2, 8, 3, 7, 4, 6, 5};
int size = sizeof(array)/sizeof(array[0]);
sort(array, size);
for(int element : array){
std::cout << element << " ";
}
return 0;
}
void sort(int array[], int size){
int temp;
for(int i = 0; i < size - 1; i++)
{
for(int j = 0; j < size - i - 1; j++){
if(array[j] < array[j + 1]){
temp = array[j];
array[j] = array[j + 1];
array[j + 1] = temp;
}
}
}
 }
```

FILL( ) FUNCTION

```
#include <iostream>
int main() {
// fill() = Fills a range of elements with a specified value // fill(begin, end, value)
```

```
const int SIZE = 99;
std::string foods[SIZE];
fill(foods, foods + (SIZE/3), "pizza");                    *BEGIN,THEN IT ENDED BY 33 ,PIZZA*
fill(foods + (SIZE/3), foods + (SIZE/3)*2, "hamburger");  *BEGIN WITH 33,ENDED BY 66,HAMBURGER
fill(foods + (SIZE/3)*2, foods + SIZE, "hotdog");  *BEGIN WITH 66,ENDED BY 99,HOTDOG
for(std::string food : foods){
std::cout << food << '\n';
 }
return 0;
 }
```

MULTIDIMENSIONAL ARRAY

```
#include <iostream>
int main() {
std::string cars[ ][3] = {{"Mustang", "Escape", "F-150"},        //here the 2 dimensional array have 2[ ] .
 {"Corvette", "Equinox", "Silverado"},
                        {"Challenger", "Durango", "Ram 1500"}};
int rows = sizeof(cars)/sizeof(cars[0]);
int columns = sizeof(cars[0])/sizeof(cars[0][0]);
for(int i = 0; i < rows; i++){
for(int j = 0; j < columns; j++){
std::cout << cars[i][j] << " ";
}
std::cout << '\n';
}
return 0;
}
```

MEMORY ADDRESS (&)

#include <iostream>

int main() {

// memory address = a location in memory where data is stored // a memory address can accessed with & (address-of operator). its give a hexadecimal address.

std::string name = "Bro";

int age = 21;

bool student = true;

std::cout << &name << '\n';

std::cout << &age << '\n';

std::cout << &student << '\n';

return 0;

}

PASS BY REFERENCE AND PASS BY VALUE

Reference

A reference variable is a "reference" to an existing variable, and it is created with the & operator:

```
string food = "Pizza";
string &meal = food;

cout << food << "\n";  // Outputs Pizza
cout << meal << "\n";  // Outputs Pizza
```

#include <iostream>

void swap(std::string &x, std::string &y); //here we use & to swap the value

int main() {

std::string x = "Kool-Aid";

```
std::string y = "Water";

swap(x, y);

std::cout << "X: " << x << '\n';                //here we use the & to swap the reference

std::cout << "Y: " << y << '\n'; return 0;

}

void swap(std::string &x, std::string &y){          //value

std::string temp;

temp = x;

x = y;

y = temp;

 }
```

We passed the values we created the copies x and y with the prameters ,when we use the address of operator we're passing memory address to where the orginal x and y are located .

Memory address

the & operator was used to create a reference variable. But it can also be used to get the memory address of a variable; which is the location of where the variable is stored on the computer.

When a variable is created in C++, a memory address is assigned to the variable. And when we assign a value to the variable, it is stored in this memory address.

To access it, use the & operator, and the result will represent where the variable is stored:

Example

```
string food = "Pizza";

cout << &food; // Outputs 0x6dfed4
```

CONST PARAMETERS

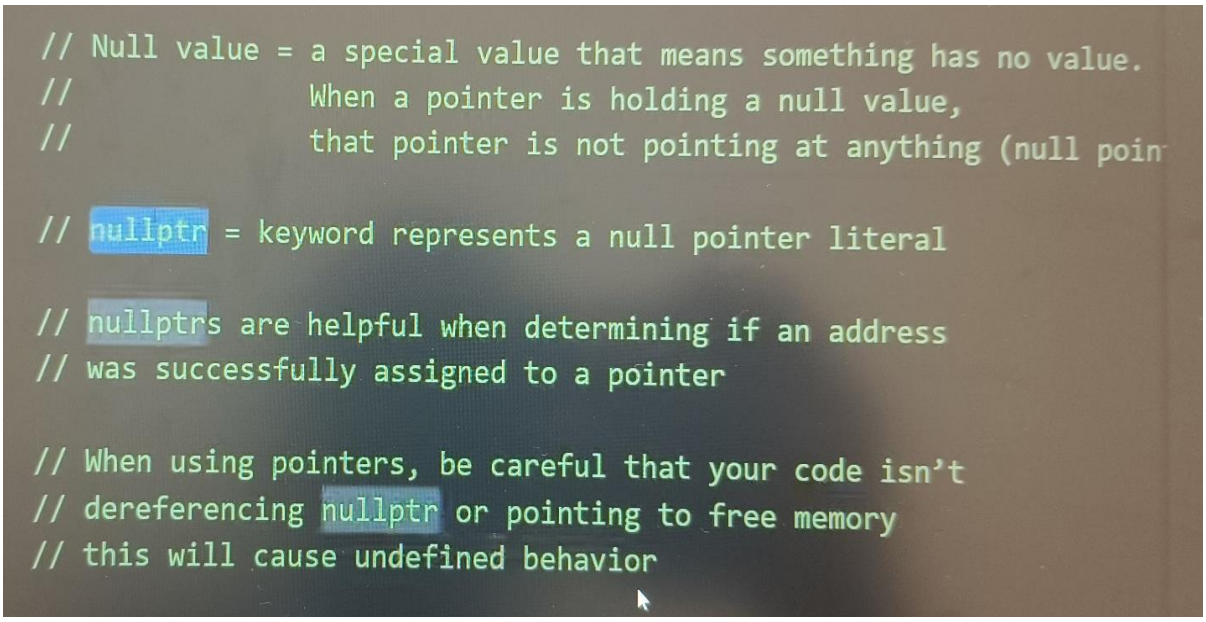
```
#include <iostream>

void printInfo(const std::string &name, const int &age);

int main() {

// const parameter = parameter that is effectively read-only // conveys intent & code is more secure
// useful for pointers and references

std::string name = "Bro"; int age = 21;

 printInfo(name, age); return 0;

}

void printInfo(const std::string &name, const int &age){

 //name = ""; //age = 0;

std::cout << name << '\n';

 std::cout << age << '\n';

}
```

POINTERS

```
#include <iostream>

 int main() {

 // pointers = variable that stores a memory address of another variable // sometimes it's easier to
work with an address // & address-of operator // * dereference operator

std::string name = "Bhagath";

int age = 18;

std::string freePizzas[5] = {"pizza1", "pizza2", "pizza3", "pizza4", "pizza5"};

std::string *pName = &name;                          // common name for pointers is p

 int *pAge = &age;                                  //the datatype of the pointer is same as the variable

std::string *pFreePizzas = freePizzas;

std::cout << *pName << '\n';                        //  here,we use * to get the value of the address

std::cout << *pAge << '\n';                         //     if we don't use * we will get a memory address

std::cout << *pFreePizzas << '\n';

 return 0;

 }
```

```
string food = "Pizza";  // A food variable of type string
string* ptr = &food;    // A pointer variable, with the name ptr, that stores the address of food

// Output the value of food (Pizza)
cout << food << "\n";

// Output the memory address of food (0x6dfed4)
cout << &food << "\n";

// reference : Output the memory address of food with the pointer (0x6dfed4)
cout << ptr << "\n";

// Dereference: Output the value of food with the pointer (Pizza)

cout << *ptr << "\n";

// Change the value of the pointer
*ptr = "Hamburger";

// Output the new value of the pointer (Hamburger)
cout << *ptr << "\n";

// Output the new value of the food variable (Hamburger)
cout << food << "\n";
```

NULL POINTER




```
#include <iostream>

int main() {

int *pointer = nullptr;

int x = 123;

pointer = &x;

if(pointer == nullptr)              //  its the good way to check the pointer still remain a null pointer or not
{ std::cout << "address was not assigned!\n";

} else{ std::cout << "address was assigned!\n";

std::cout << *pointer; }

return 0;

}
```

A null pointer is represented by the value 0 or by using the keyword NULL. With the new versions of C++ like C++11 and later, we can use "nullptr" to indicate a null pointer.

```
// C++ program to demonstrate the dereferencing and

// assignment of null pointer to another value.


#include <iostream>
```

```
using namespace std;
int main()
{
    int* ptr = nullptr;
    // Checking if the pointer is null before dereferencing
    if (ptr == nullptr) {
        cout << "Pointer is currently null." << endl;
    }
    else {
        cout << "Pointer is not null." << endl;
    }
    // *ptr = 10; (to avoid runtime error)
    // Assigning a valid memory address to the pointer
    int value = 5;
    ptr = &value;
    // Checking if the pointer is null after assigning a
    // valid address
    if (ptr == nullptr) {
        cout << "Pointer is currently null." << endl;
    }
    else {
        cout << "Pointer is not null." << endl;
        cout << "Value at the memory location pointed to "
                "by the pointer: "
             << *ptr << endl;
    }
return 0;
}
```

Outputs

```
Pointer is currently null.
Pointer is not null.
Value at the memory location pointed to by the pointer: 5
```

DYNAMIC MEMORY

```
#include <iostream>

int main () {

char *pGrades = NULL;

 int size; std::cout << "How many grades to enter in?: ";

 std::cin >> size;

pGrades = new char[size];

 for(int i = 0; i < size; i++){

 std::cout << "Enter grade #" << i + 1 << ": ";

 std::cin >> pGrades[i]; }

for(int i = 0; i < size; i++){

std::cout << pGrades[i] << " "; }

delete[] pGrades;                    //  we are freeing up the memory at this address

return 0;

}
```