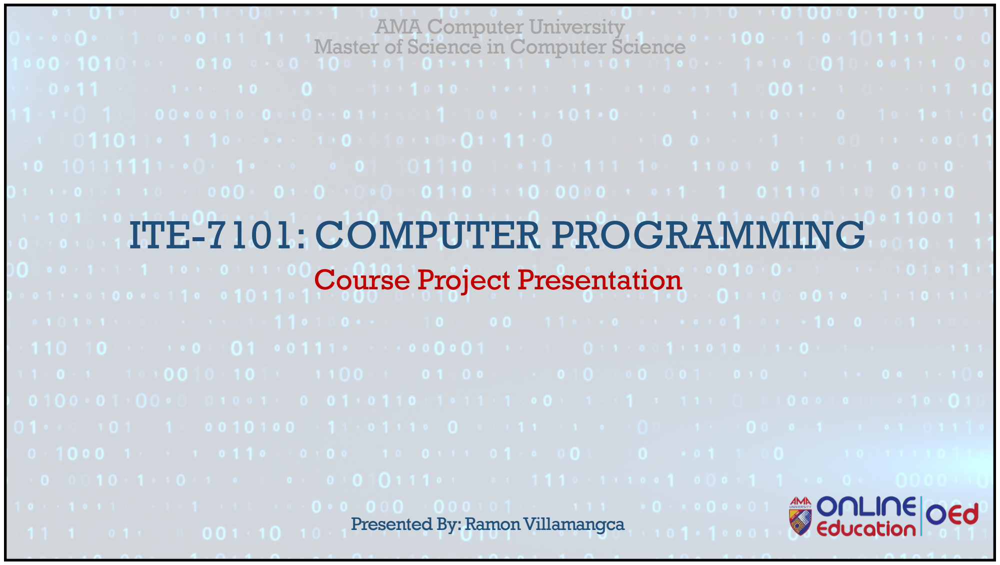


AMA Computer University
Master of Science in Computer Science

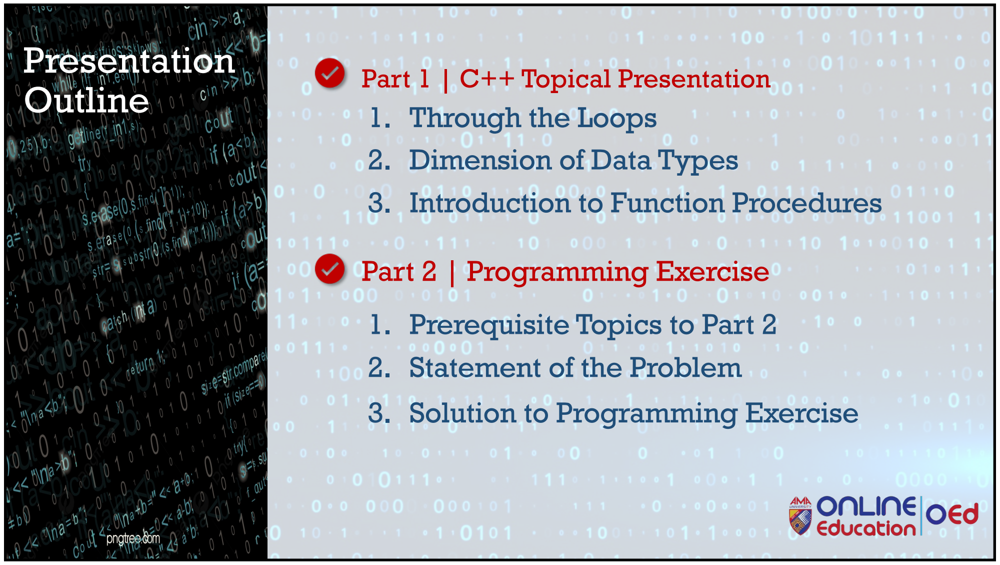
ITE-7101: COMPUTER PROGRAMMING

Course Project Presentation

Presented By: Ramon Villamangca

ONLINE Education
OEd

1



Presentation Outline

✓ Part 1 | C++ Topical Presentation

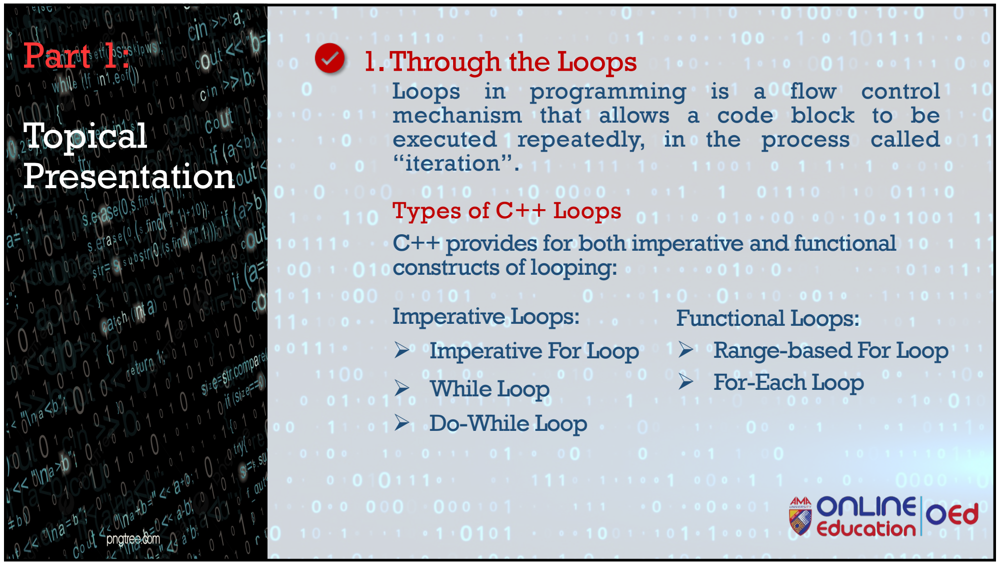
1. Through the Loops
2. Dimension of Data Types
3. Introduction to Function Procedures

✓ Part 2 | Programming Exercise

1. Prerequisite Topics to Part 2
2. Statement of the Problem
3. Solution to Programming Exercise

pngtree.com
ONLINE Education
OEd

2



Part 1:

Topical Presentation

1. Through the Loops

Loops in programming is a flow control mechanism that allows a code block to be executed repeatedly, in the process called "iteration".

Types of C++ Loops

C++ provides for both imperative and functional constructs of looping:

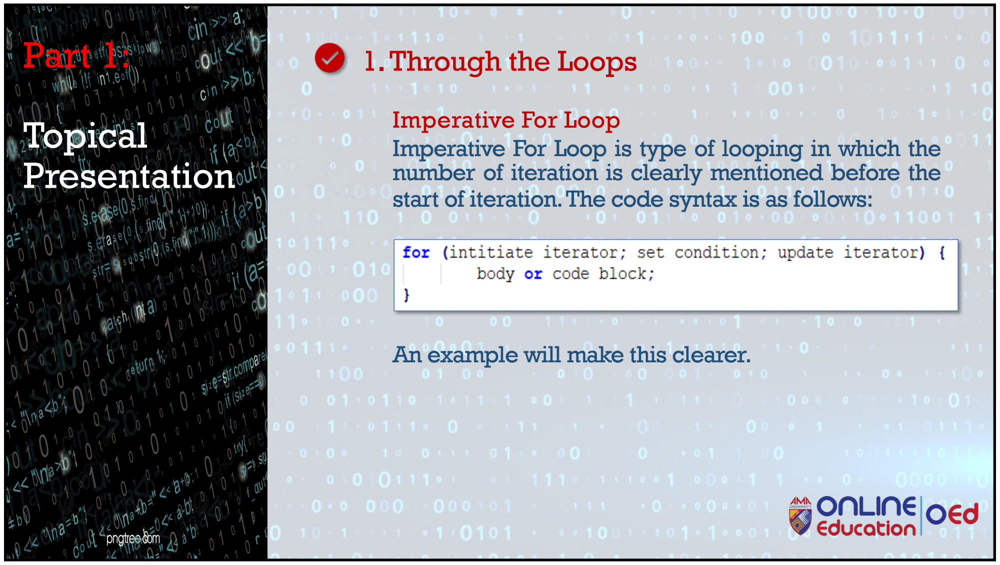
Imperative Loops:

- ➢ Imperative For Loop
- ➢ While Loop
- ➢ Do-While Loop

Functional Loops:

- ➢ Range-based For Loop
- ➢ For-Each Loop

3



Part 1:

Topical Presentation

1. Through the Loops

Imperative For Loop

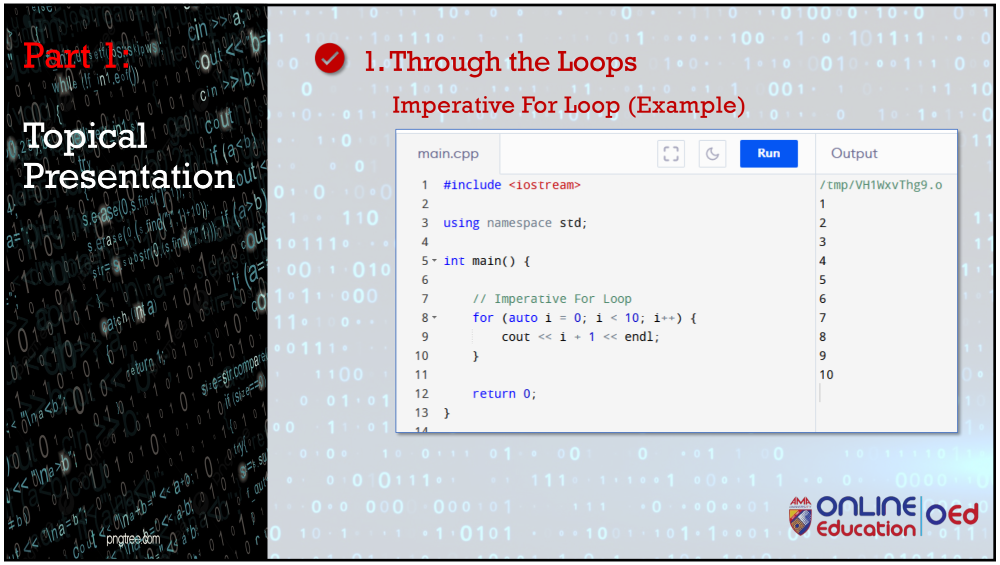
Imperative For Loop is type of looping in which the number of iteration is clearly mentioned before the start of iteration. The code syntax is as follows:

```
for (intitiate iterator; set condition; update iterator) {
        body or code block;
}
```

An example will make this clearer.

pngtree.com
ONLINE Education
OEd

4



Part 1:

Topical Presentation

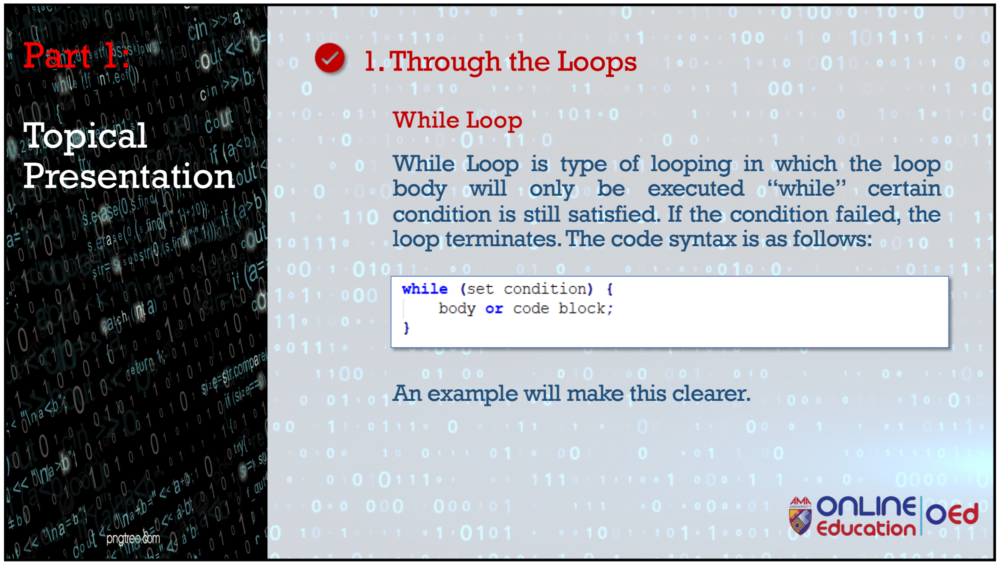
1. Through the Loops

Imperative For Loop (Example)

```cpp
#include <iostream>

using namespace std;

int main() {

    // Imperative For Loop
    for (auto i = 0; i < 10; i++) {
        cout << i + 1 << endl;
    }

    return 0;
}
```

```
/tmp/VH1WxvThg9.o
1
2
3
4
5
6
7
8
9
10
```

5



Part 1:

Topical Presentation

1. Through the Loops

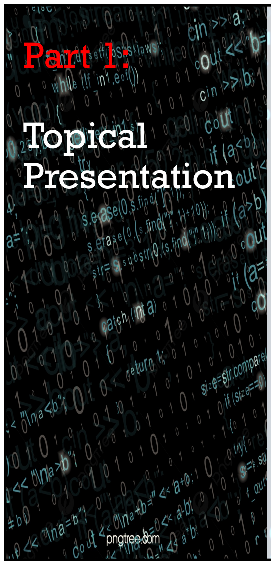
While Loop

While Loop is type of looping in which the loop body will only be executed "while" certain condition is still satisfied. If the condition failed, the loop terminates. The code syntax is as follows:

```
while (set condition) {
    body or code block;
}
```

An example will make this clearer.

6

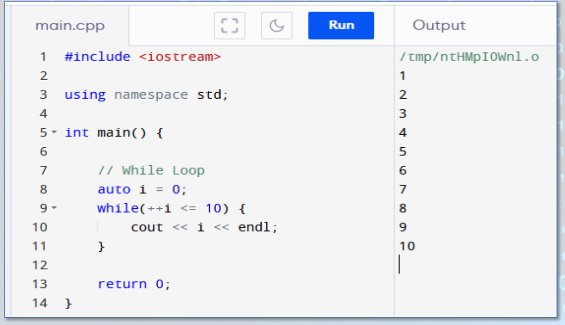


Part 1:

Topical Presentation

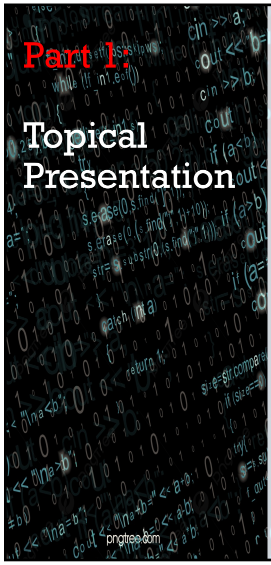
1. Through the Loops

While Loop (Example)



```cpp
#include <iostream>

using namespace std;

int main() {

    // While Loop
    auto i = 0;
    while(++i <= 10) {
        cout << i << endl;
    }

    return 0;
}
```

```
/tmp/ntHMpIOWnl.o
1
2
3
4
5
6
7
8
9
10
```



Part 1:

Topical Presentation

1. Through the Loops

Do-While Loop

Do-While Loop is variation of the While Loop wherein the conditional statement is at the end of the code block. The code syntax is as follows:

```
do {
    body or code block;
} while(set condition);
```

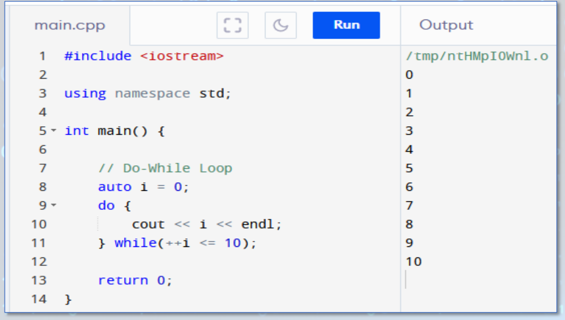
An example will make this clearer.

Part 1:

Topical Presentation

1. Through the Loops

Do-While Loop (Example)



```cpp
#include <iostream>

using namespace std;

int main() {

    // Do-While Loop
    auto i = 0;
    do {
        cout << i << endl;
    } while(++i <= 10);

    return 0;
}
```

```
/tmp/ntHMpIOWnl.o
0
1
2
3
4
5
6
7
8
9
10
```

9

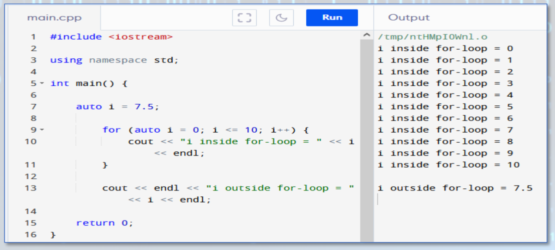
Part 1:

Topical Presentation

1. Through the Loops

Scope and Variable Lifetime

A local variable is only valid in the code block it resides on. The validity of a local variable within a code blocks is called its scope.

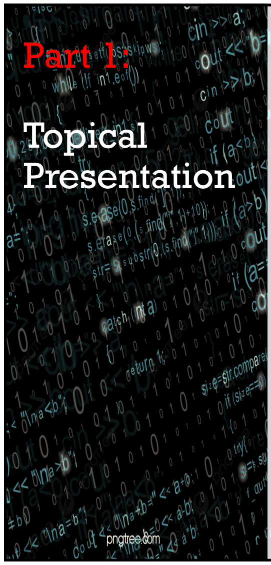


```cpp
#include <iostream>

using namespace std;

int main() {

    auto i = 7.5;

        for (auto i = 0; i <= 10; i++) {
            cout << "i inside for-loop = " << i
                << endl;
        }

        cout << endl << "i outside for-loop = "
            << i << endl;

    return 0;
}
```

```
/tmp/ntHMpIOWnl.o
i inside for-loop = 0
i inside for-loop = 1
i inside for-loop = 2
i inside for-loop = 3
i inside for-loop = 4
i inside for-loop = 5
i inside for-loop = 6
i inside for-loop = 7
i inside for-loop = 8
i inside for-loop = 9
i inside for-loop = 10

i outside for-loop = 7.5
```

Variables created inside a loop, lose their scope when the loop ends.

10



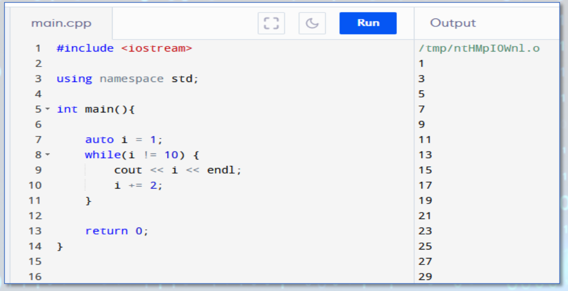
Part 1:

Topical Presentation

1. Through the Loops

Infinite Loop

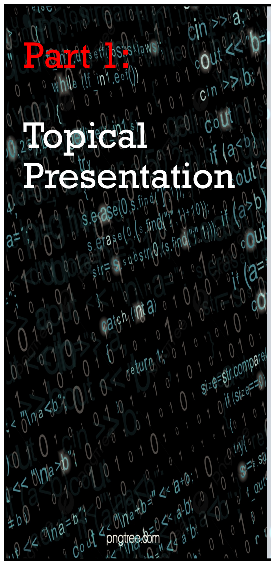
The situation wherein the terminating condition can never be satisfied and therefore the loop cannot terminate, is called Infinite Loop.



```
#include <iostream>

using namespace std;

int main(){

    auto i = 1;
    while(i != 10) {
        cout << i << endl;
        i += 2;
    }

    return 0;
}

```

```
/tmp/ntHMpIOWnl.o
1
3
5
7
9
11
13
15
17
19
21
23
25
27
29
```

11



Part 1:

Topical Presentation

1. Through the Loops

Break and Continue Statements

Break statement halts the execution of all statements inside the loop, after the "break" and terminates the loop completely.

Continue statement halts the execution of all statements inside the loop, after the "continue" and returns to the start of the loop.

An example will make this clearer.

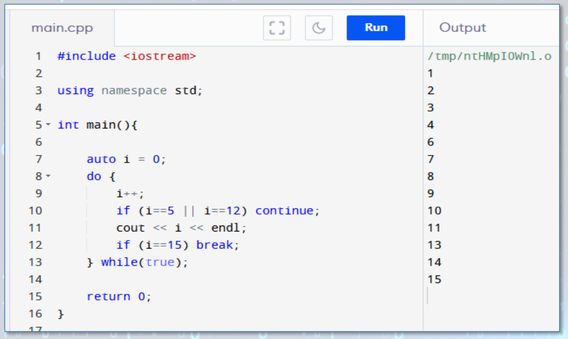
12

Part 1:

Topical Presentation

1. Through the Loops

Break and Continue (Example)



```
#include <iostream>

using namespace std;

int main(){

    auto i = 0;
    do {
        i++;
        if (i==5 || i==12) continue;
        cout << i << endl;
        if (i==15) break;
    } while(true);

    return 0;
}
```

```
/tmp/ntHMpIOWnl.o
1
2
3
4
6
7
8
9
10
11
13
14
15
```

ONLINE Education OEd

13

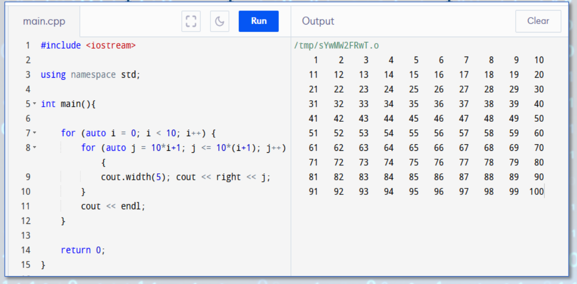
Part 1:

Topical Presentation

1. Through the Loops

Nested Loops

Nested loops are loops inside another loop/s.



```
#include <iostream>

using namespace std;

int main(){

    for (auto i = 0; i < 10; i++) {
        for (auto j = 10*i+1; j <= 10*(i+1); j++)
            {
            cout.width(5); cout << right << j;
        }
        cout << endl;
    }

    return 0;
}
```

```
/tmp/sYwMW2FRwT.o
    1    2    3    4    5    6    7    8    9   10
   11   12   13   14   15   16   17   18   19   20
   21   22   23   24   25   26   27   28   29   30
   31   32   33   34   35   36   37   38   39   40
   41   42   43   44   45   46   47   48   49   50
   51   52   53   54   55   56   57   58   59   60
   61   62   63   64   65   66   67   68   69   70
   71   72   73   74   75   76   77   78   79   80
   81   82   83   84   85   86   87   88   89   90
   91   92   93   94   95   96   97   98   99  100
```

We need to be careful in using nested loops, as the number of steps is compounded by the number of loops.

ONLINE Education OEd

14

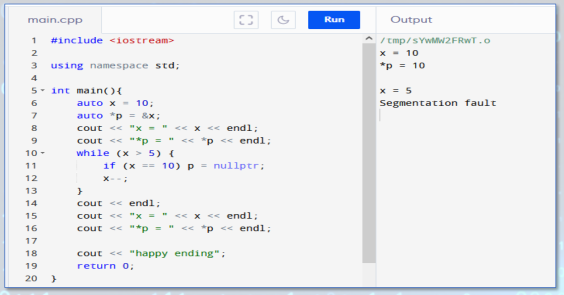
Part 1:

Topical Presentation

1. Through the Loops

Side Effects

Side effect is the ability of code block to modify some states outside of its scope.



```cpp
#include <iostream>

using namespace std;

int main(){
    auto x = 10;
    auto *p = &x;
    cout << "x = " << x << endl;
    cout << "*p = " << *p << endl;
    while (x > 5) {
        if (x == 10) p = nullptr;
        x--;
    }
    cout << endl;
    cout << "x = " << x << endl;
    cout << "*p = " << *p << endl;

    cout << "happy ending";
    return 0;
}
```

```
/tmp/sYwMW2FRwT.o
x = 10
*p = 10

x = 5
Segmentation fault
```

ONLINE Education | OEd

15

Part 1:

Topical Presentation

1. Through the Loops

Range-based For Loop

Range-based For Loop is a way to iterate a container without changing or modifying the content of the container. The code syntax is as follows:

```
for (declare member variable : container) {
        body or code block;
}
```

An example will make this clearer.

ONLINE Education | OEd

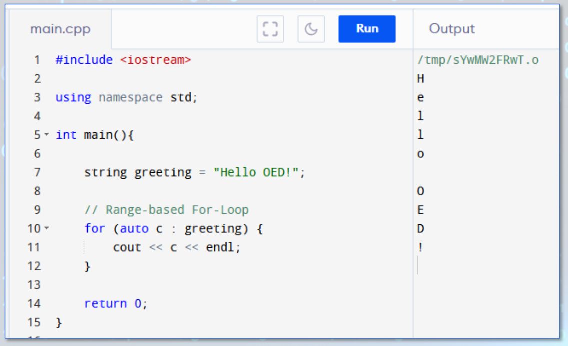
16

Part 1:

Topical Presentation

1. Through the Loops

Range-based For Loop (Example)



```cpp
#include <iostream>

using namespace std;

int main(){

    string greeting = "Hello OED!";

    // Range-based For-Loop
    for (auto c : greeting) {
        cout << c << endl;
    }

    return 0;
}
```

```
/tmp/sYwMW2FRwT.o
H
e
l
l
o

O
E
D
!
```

ONLINE Education OEd

17

Part 1:

Topical Presentation

1. Through the Loops

For-each Loop

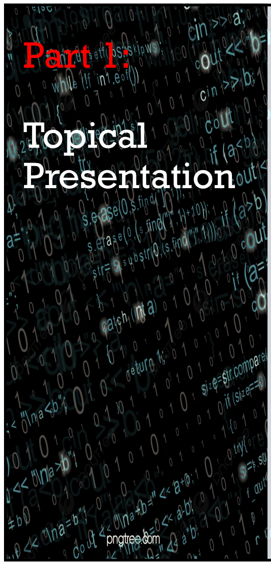
Like the range-based for-loop, For-Each can be applied to elements of a container. But it is actually a function taking in three parameters. The code syntax is as follows:

```
for_each (start pointer, end pointer, function);
```

An example will make this clearer.

ONLINE Education OEd

18



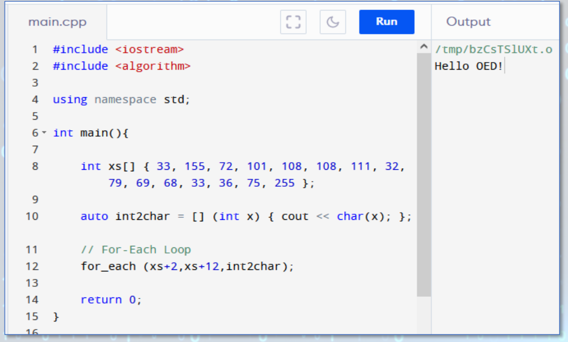
Part 1:

Topical Presentation

pngtree.com

1. Through the Loops

For-each Loop (Example)

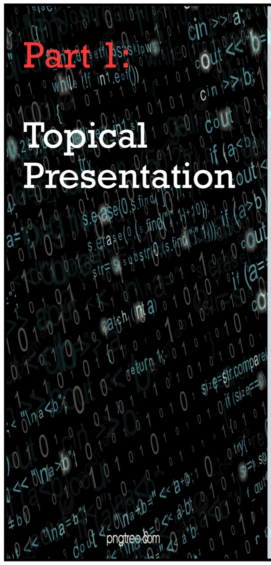


```
#include <iostream>
#include <algorithm>

using namespace std;

int main(){

    int xs[] { 33, 155, 72, 101, 108, 108, 111, 32,
        79, 69, 68, 33, 36, 75, 255 };

    auto int2char = [] (int x) { cout << char(x); };

    // For-Each Loop
    for_each (xs+2,xs+12,int2char);

    return 0;
}
```

```
/tmp/bzCsTSlUXt.o
Hello OED!
```

ONLINE Education | OEd

19



Part 1:

Topical Presentation

pngtree.com

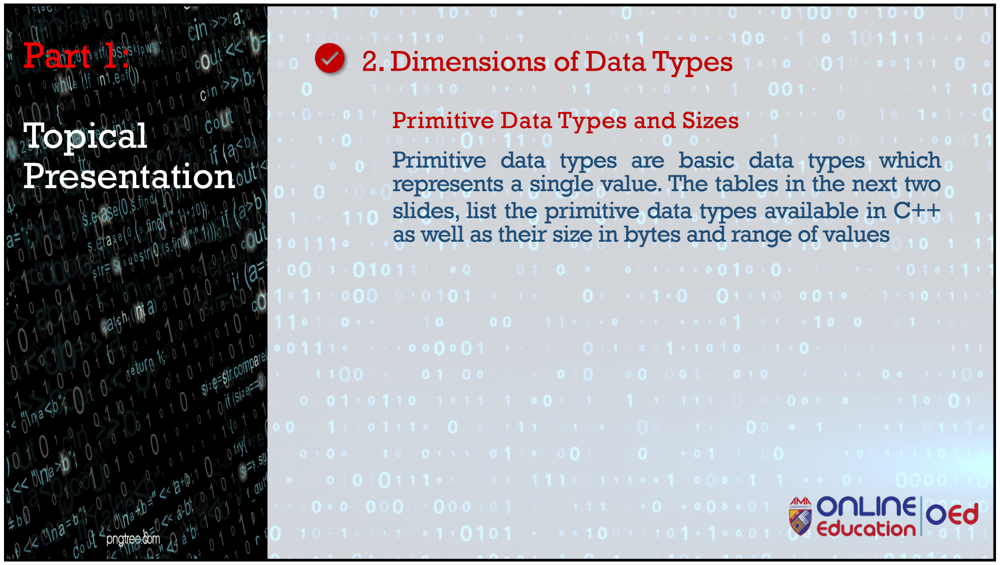
2. Dimensions of Data Types

Data Type refers to the type of value a variable has. Data types has size which is the amount of memory will be allocated to a certain data type.

- ➢ Primitive Data Types and Sizes
- ➢ The “sizeof” and “typeid” Operators
- ➢ C++ STL Arrays and Dimensions
- ➢ Derived Data Types and Sizes

ONLINE Education | OEd

20


pngtree.com

Part 1:

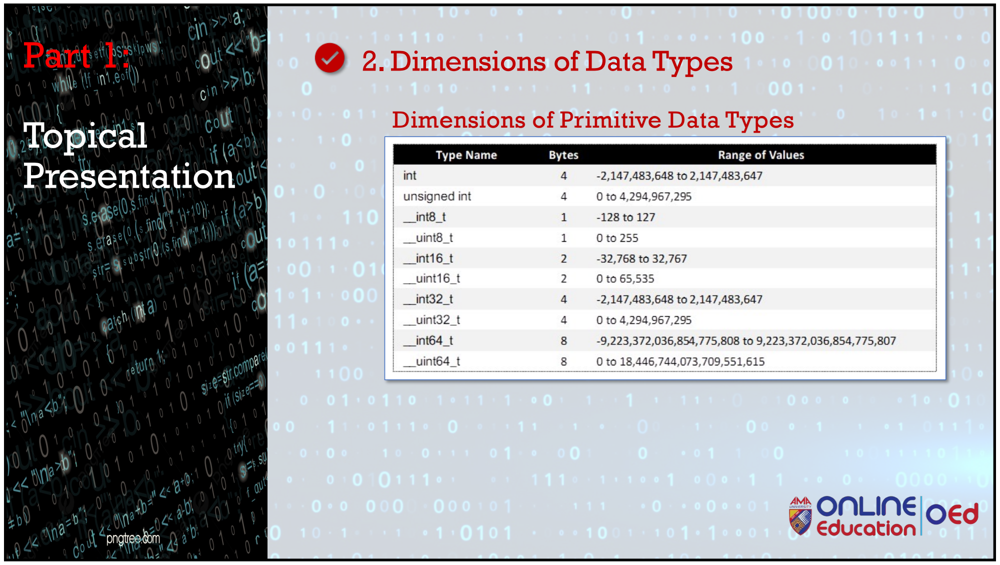
Topical Presentation

2. Dimensions of Data Types

Primitive Data Types and Sizes

Primitive data types are basic data types which represents a single value. The tables in the next two slides, list the primitive data types available in C++ as well as their size in bytes and range of values

21


pngtree.com

Part 1:

Topical Presentation

2. Dimensions of Data Types

Dimensions of Primitive Data Types

Type Name	Bytes	Range of Values
int	4	-2,147,483,648 to 2,147,483,647
unsigned int	4	0 to 4,294,967,295
__int8_t	1	-128 to 127
__uint8_t	1	0 to 255
__int16_t	2	-32,768 to 32,767
__uint16_t	2	0 to 65,535
__int32_t	4	-2,147,483,648 to 2,147,483,647
__uint32_t	4	0 to 4,294,967,295
__int64_t	8	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
__uint64_t	8	0 to 18,446,744,073,709,551,615

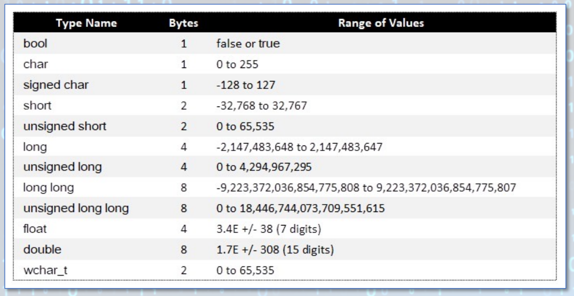
22

Part 1:

Topical Presentation

2. Dimensions of Data Types

Dimensions of Primitive Data Types



Type Name	Bytes	Range of Values
bool	1	false or true
char	1	0 to 255
signed char	1	-128 to 127
short	2	-32,768 to 32,767
unsigned short	2	0 to 65,535
long	4	-2,147,483,648 to 2,147,483,647
unsigned long	4	0 to 4,294,967,295
long long	8	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
unsigned long long	8	0 to 18,446,744,073,709,551,615
float	4	3.4E +/- 38 (7 digits)
double	8	1.7E +/- 308 (15 digits)
wchar_t	2	0 to 65,535

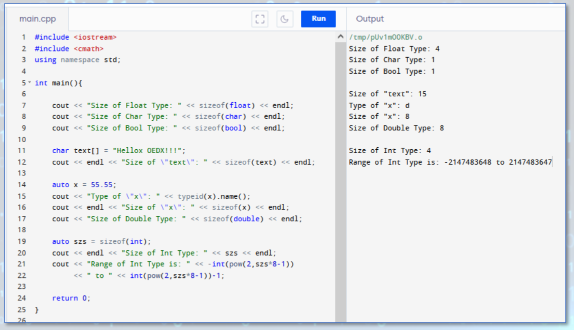
23

Part 1:

Topical Presentation

2. Dimensions of Data Types

"sizeof" and "typeid" Operators

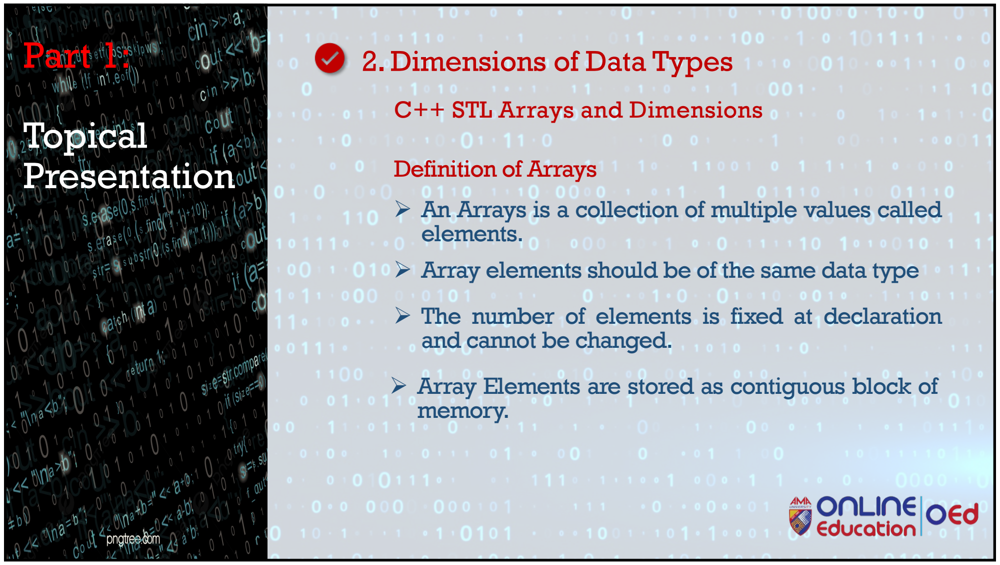


```cpp
#include <iostream>
#include <cmath>
using namespace std;

int main(){

    cout << "Size of Float Type: " << sizeof(float) << endl;
    cout << "Size of Char Type: " << sizeof(char) << endl;
    cout << "Size of Bool Type: " << sizeof(bool) << endl;

    char text[] = "Hellox OEDX!!!";
    cout << endl << "Size of \"text\": " << sizeof(text) << endl;

    auto x = 55.55;
    cout << "Type of \"x\": " << typeid(x).name();
    cout << endl << "Size of \"x\": " << sizeof(x) << endl;
    cout << "Size of Double Type: " << sizeof(double) << endl;

    auto szs = sizeof(int);
    cout << endl << "Size of Int Type: " << szs << endl;
    cout << "Range of Int Type is: " << -int(pow(2,szs*8-1))
         << " to " << int(pow(2,szs*8-1))-1;

    return 0;
}
```

```
/tmp/pUv1mOOKBV.o
Size of Float Type: 4
Size of Char Type: 1
Size of Bool Type: 1

Size of "text": 15
Type of "x": d
Size of "x": 8
Size of Double Type: 8

Size of Int Type: 4
Range of Int Type is: -2147483648 to 2147483647
```

int	4	-2,147,483,648 to 2,147,483,647

24



Part 1:

Topical Presentation

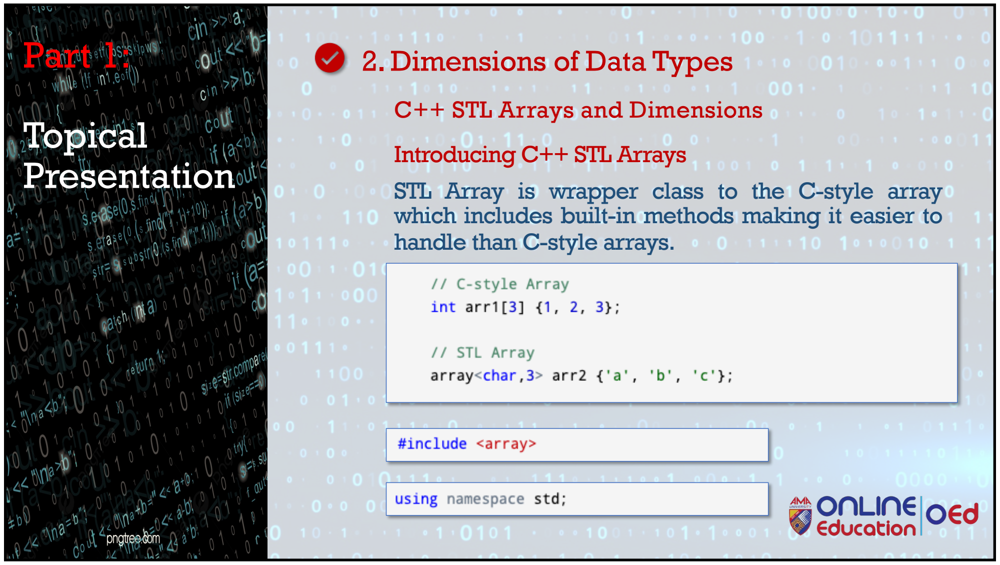
2. Dimensions of Data Types

C++ STL Arrays and Dimensions

Definition of Arrays

- An Arrays is a collection of multiple values called elements.
- Array elements should be of the same data type
- The number of elements is fixed at declaration and cannot be changed.
- Array Elements are stored as contiguous block of memory.

25



Part 1:

Topical Presentation

2. Dimensions of Data Types

C++ STL Arrays and Dimensions

Introducing C++ STL Arrays

STL Array is wrapper class to the C-style array which includes built-in methods making it easier to handle than C-style arrays.

```
// C-style Array
int arr1[3] {1, 2, 3};

// STL Array
array<char,3> arr2 {'a', 'b', 'c'};
```

```
#include <array>
```

```
using namespace std;
```

26

Part 1:

Topical Presentation

2. Dimensions of Data Types

C++ STL Arrays and Dimensions

Accessing Elements of Arrays

- Index is the position of elements in cell
- Arrays is zero-based
- n-th element is at index "n-1"

An example will make this clearer.

27

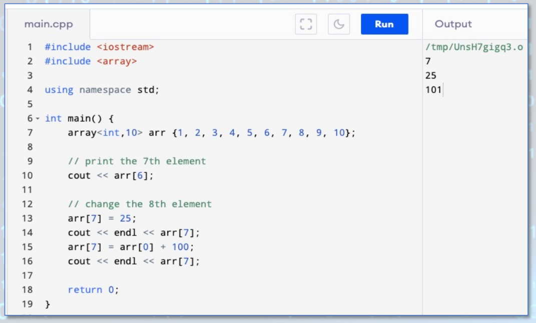
Part 1:

Topical Presentation

2. Dimensions of Data Types

C++ STL Arrays and Dimensions

Accessing Elements of Arrays (Example)



```cpp
#include <iostream>
#include <array>

using namespace std;

int main() {
    array<int,10> arr {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

    // print the 7th element
    cout << arr[6];

    // change the 8th element
    arr[7] = 25;
    cout << endl << arr[7];
    arr[7] = arr[0] + 100;
    cout << endl << arr[7];

    return 0;
}
```

```
/tmp/UnsH7gigq3.o
7
25
101
```

28

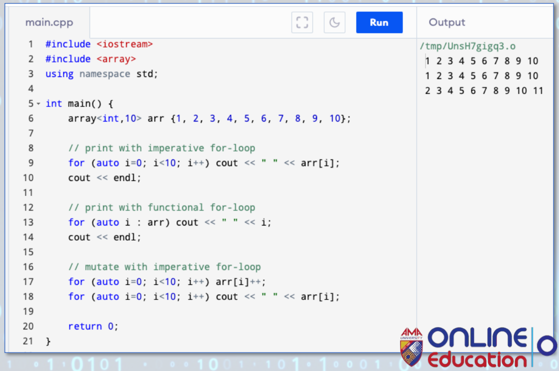
Part 1:

Topical Presentation

2. Dimensions of Data Types

C++ STL Arrays and Dimensions

Traversing and Iterating Arrays



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

int main() {
    array<int,10> arr {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

    // print with imperative for-loop
    for (auto i=0; i<10; i++) cout << " " << arr[i];
    cout << endl;

    // print with functional for-loop
    for (auto i : arr) cout << " " << i;
    cout << endl;

    // mutate with imperative for-loop
    for (auto i=0; i<10; i++) arr[i]++;
    for (auto i=0; i<10; i++) cout << " " << arr[i];

    return 0;
}
```

```
/tmp/UnsH7gigq3.o
 1 2 3 4 5 6 7 8 9 10
 1 2 3 4 5 6 7 8 9 10
 2 3 4 5 6 7 8 9 10 11
```

29

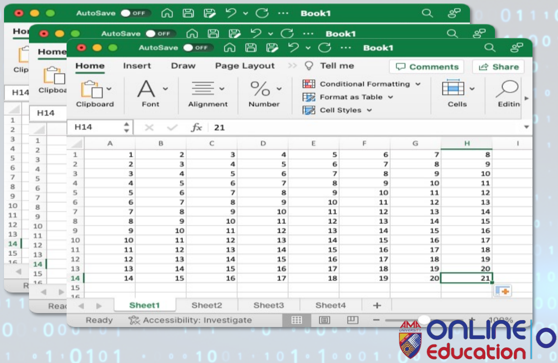
Part 1:

Topical Presentation

2. Dimensions of Data Types

C++ STL Arrays and Dimensions

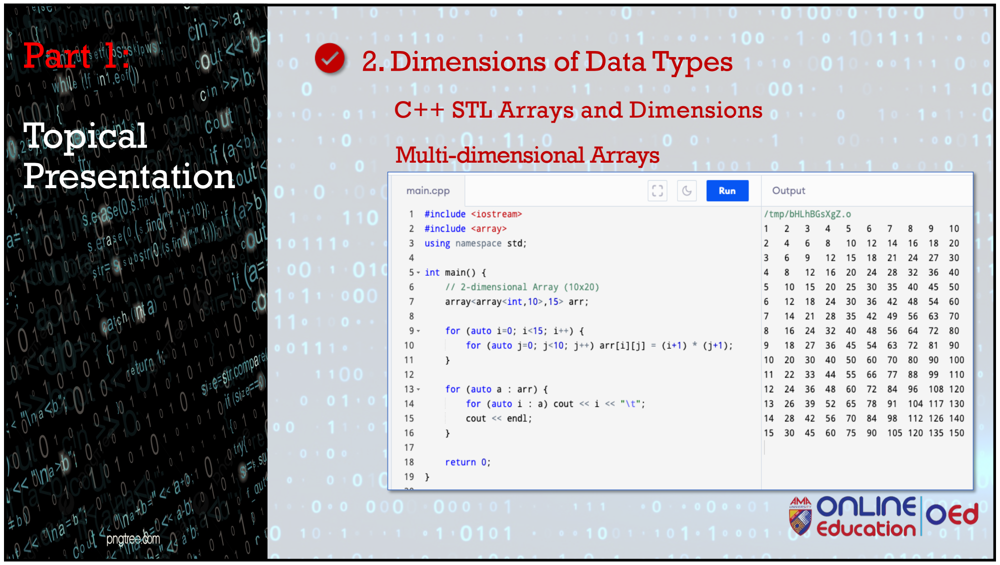
Multi-dimensional Arrays



H14 fx 21

	A	B	C	D	E	F	G	H	I
1	1	2	3	4	5	6	7	8	
2	2	3	4	5	6	7	8	9	
3	3	4	5	6	7	8	9	10	
4	4	5	6	7	8	9	10	11	
5	5	6	7	8	9	10	11	12	
6	6	7	8	9	10	11	12	13	
7	7	8	9	10	11	12	13	14	
8	8	9	10	11	12	13	14	15	
9	9	10	11	12	13	14	15	16	
10	10	11	12	13	14	15	16	17	
11	11	12	13	14	15	16	17	18	
12	12	13	14	15	16	17	18	19	
13	13	14	15	16	17	18	19	20	
14	14	15	16	17	18	19	20	21	
15									

30



Part 1:

Topical Presentation

2. Dimensions of Data Types

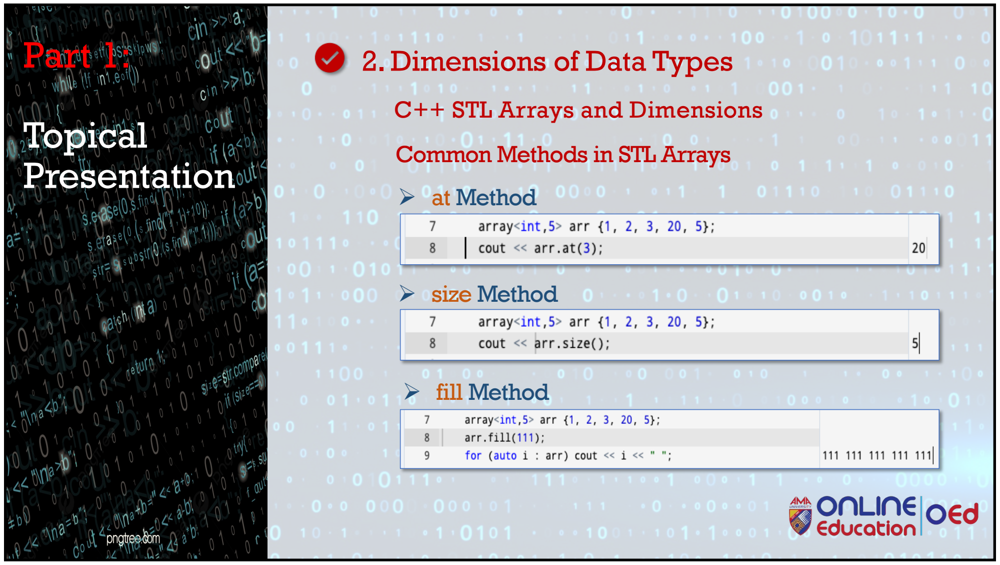
C++ STL Arrays and Dimensions

Multi-dimensional Arrays

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

int main() {
    // 2-dimensional Array (10x20)
    array<array<int,10>,15> arr;

    for (auto i=0; i<15; i++) {
        for (auto j=0; j<10; j++) arr[i][j] = (i+1) * (j+1);
    }

    for (auto a : arr) {
        for (auto i : a) cout << i << "\t";
        cout << endl;
    }

    return 0;
}
```

```
/tmp/bHLhBGsXgZ.o
1   2   3   4   5   6   7   8   9   10
2   4   6   8   10  12  14  16  18  20
3   6   9   12  15  18  21  24  27  30
4   8   12  16  20  24  28  32  36  40
5   10  15  20  25  30  35  40  45  50
6   12  18  24  30  36  42  48  54  60
7   14  21  28  35  42  49  56  63  70
8   16  24  32  40  48  56  64  72  80
9   18  27  36  45  54  63  72  81  90
10  20  30  40  50  60  70  80  90  100
11  22  33  44  55  66  77  88  99  110
12  24  36  48  60  72  84  96  108 120
13  26  39  52  65  78  91  104 117 130
14  28  42  56  70  84  98  112 126 140
15  30  45  60  75  90  105 120 135 150
```

31



Part 1:

Topical Presentation

2. Dimensions of Data Types

C++ STL Arrays and Dimensions

Common Methods in STL Arrays

- at Method

```cpp
array<int,5> arr {1, 2, 3, 20, 5};
cout << arr.at(3);
```

```
20
```

- size Method

```cpp
array<int,5> arr {1, 2, 3, 20, 5};
cout << arr.size();
```

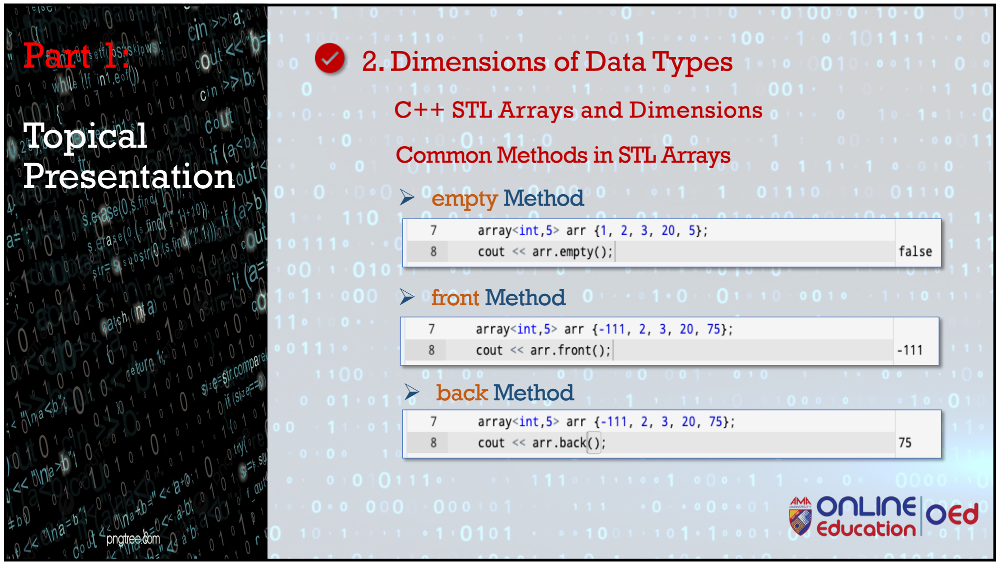
```
5
```

- fill Method

```cpp
array<int,5> arr {1, 2, 3, 20, 5};
arr.fill(111);
for (auto i : arr) cout << i << " ";
```

```
111 111 111 111 111
```

32



Part 1:

Topical Presentation

2. Dimensions of Data Types

C++ STL Arrays and Dimensions

Common Methods in STL Arrays

- empty Method

```
    array<int,5> arr {1, 2, 3, 20, 5};
    cout << arr.empty();
```

```
false
```

- front Method

```
    array<int,5> arr {-111, 2, 3, 20, 75};
    cout << arr.front();
```

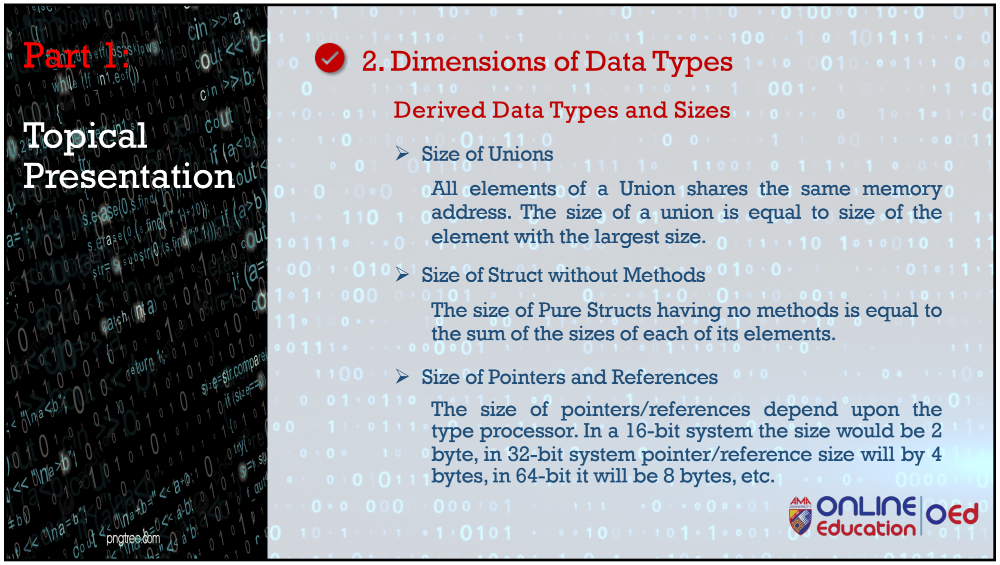
```
-111
```

- back Method

```
    array<int,5> arr {-111, 2, 3, 20, 75};
    cout << arr.back();
```

```
75
```

33



Part 1:

Topical Presentation

2. Dimensions of Data Types

Derived Data Types and Sizes

- Size of Unions

  All elements of a Union shares the same memory address. The size of a union is equal to size of the element with the largest size.

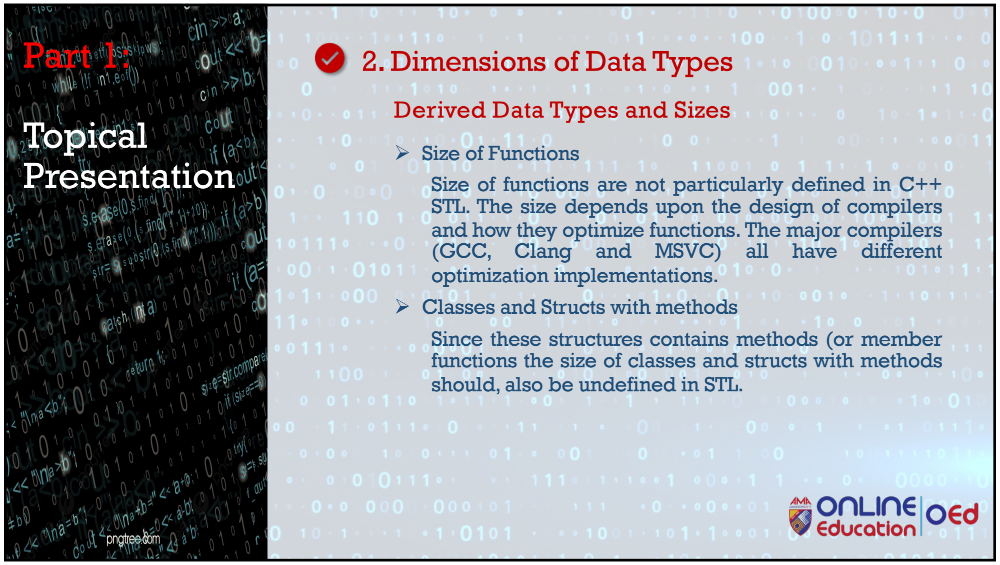
- Size of Struct without Methods

  The size of Pure Structs having no methods is equal to the sum of the sizes of each of its elements.

- Size of Pointers and References

  The size of pointers/references depend upon the type processor. In a 16-bit system the size would be 2 byte, in 32-bit system pointer/reference size will by 4 bytes, in 64-bit it will be 8 bytes, etc.

34



Part 1:

Topical Presentation

2. Dimensions of Data Types

Derived Data Types and Sizes

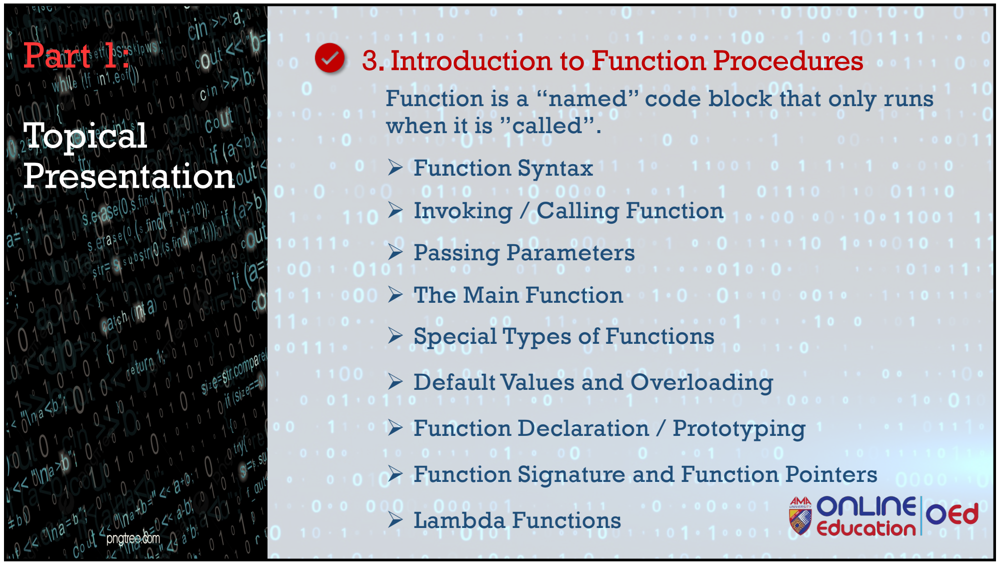
- ➢ Size of Functions

  Size of functions are not particularly defined in C++ STL. The size depends upon the design of compilers and how they optimize functions. The major compilers (GCC, Clang and MSVC) all have different optimization implementations.

- ➢ Classes and Structs with methods

  Since these structures contains methods (or member functions the size of classes and structs with methods should, also be undefined in STL.

35



Part 1:

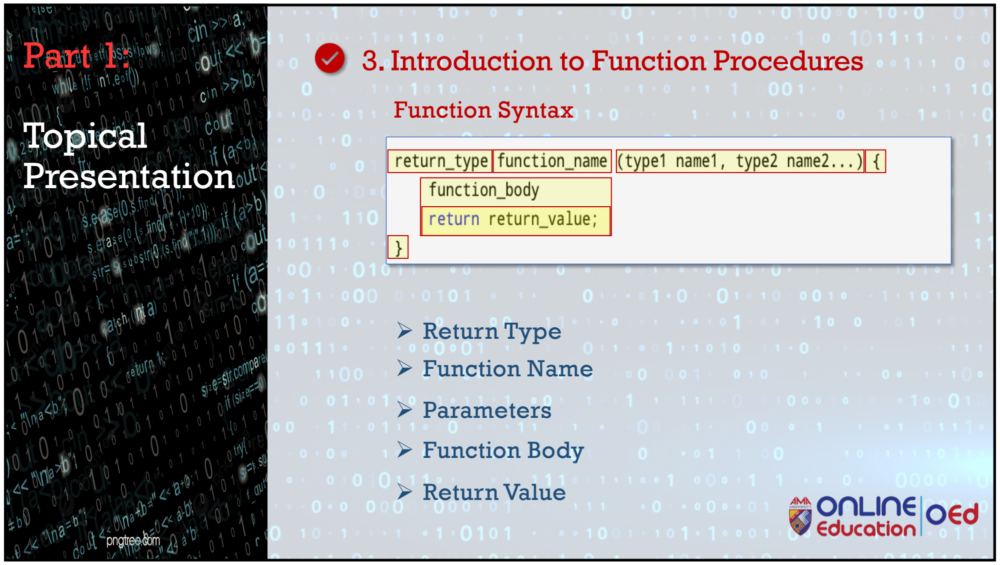
Topical Presentation

3. Introduction to Function Procedures

Function is a "named" code block that only runs when it is "called".

- ➢ Function Syntax
- ➢ Invoking / Calling Function
- ➢ Passing Parameters
- ➢ The Main Function
- ➢ Special Types of Functions
- ➢ Default Values and Overloading
- ➢ Function Declaration / Prototyping
- ➢ Function Signature and Function Pointers
- ➢ Lambda Functions

36



Part 1:

Topical Presentation

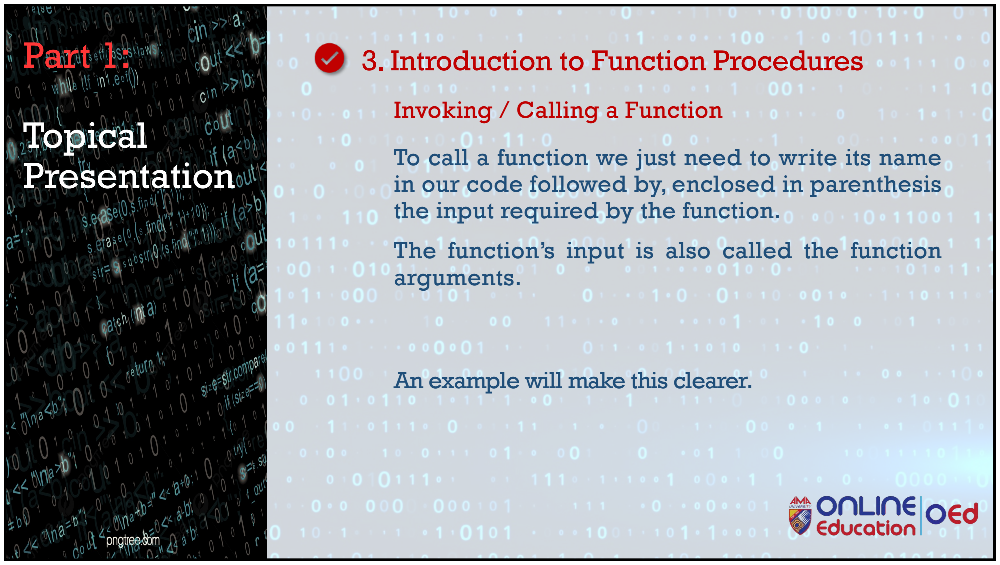
3. Introduction to Function Procedures

Function Syntax

```
return_type function_name (type1 name1, type2 name2...) {
    function_body
    return return_value;
}
```

- ➢ Return Type
- ➢ Function Name
- ➢ Parameters
- ➢ Function Body
- ➢ Return Value

37



Part 1:

Topical Presentation

3. Introduction to Function Procedures

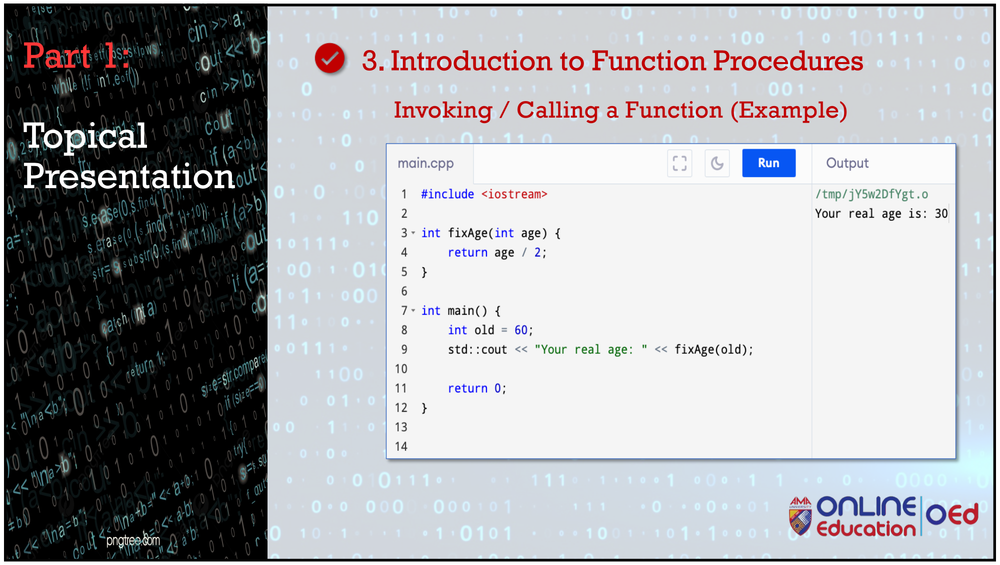
Invoking / Calling a Function

To call a function we just need to write its name in our code followed by, enclosed in parenthesis the input required by the function.

The function's input is also called the function arguments.

An example will make this clearer.

38



Part 1:

Topical Presentation

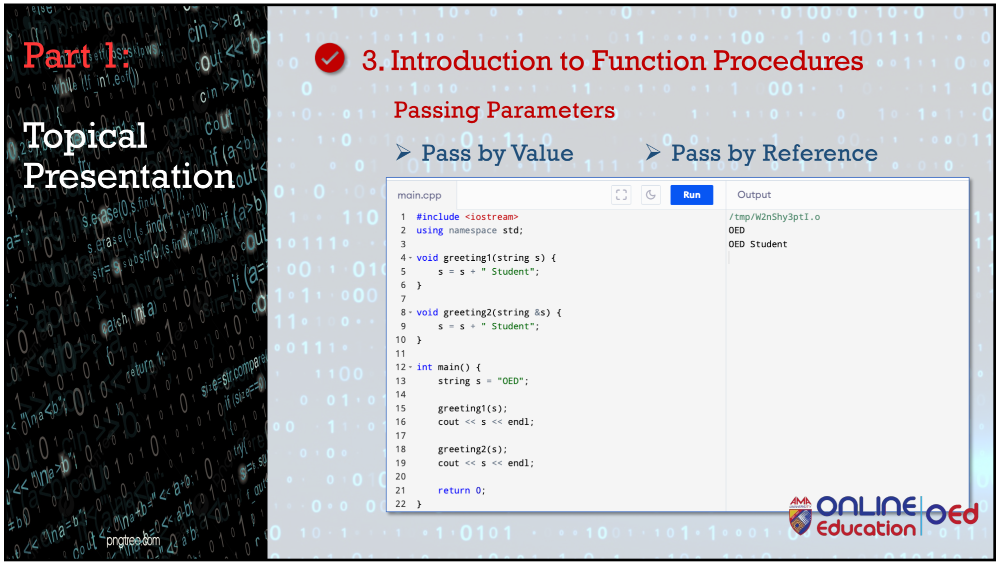
3. Introduction to Function Procedures

Invoking / Calling a Function (Example)

```cpp
#include <iostream>

int fixAge(int age) {
    return age / 2;
}

int main() {
    int old = 60;
    std::cout << "Your real age: " << fixAge(old);

    return 0;
}


```

```
/tmp/jY5w2DfYgt.o
Your real age is: 30
```

39



Part 1:

Topical Presentation

3. Introduction to Function Procedures

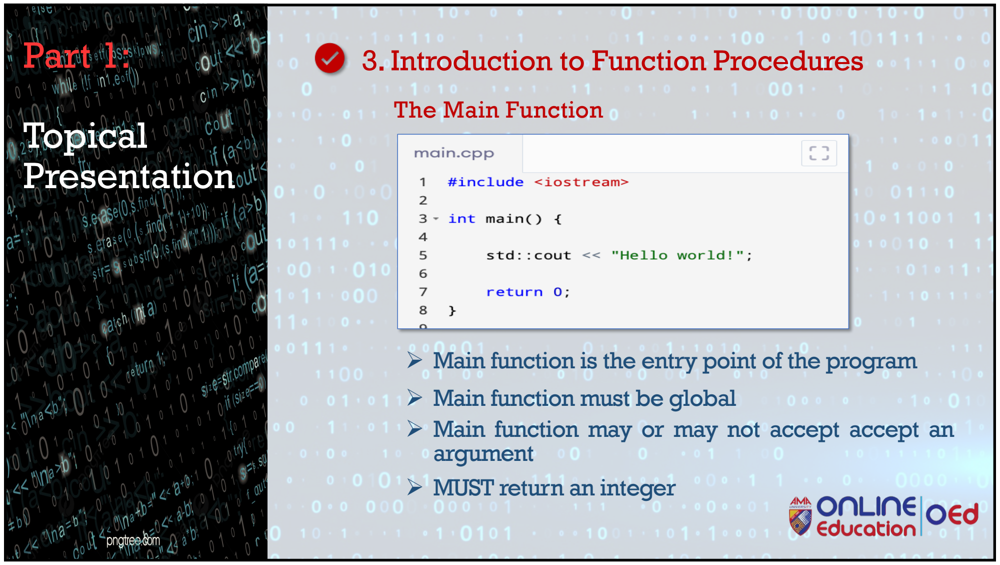
Passing Parameters

- ➢ Pass by Value
- ➢ Pass by Reference

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

void greeting1(string s) {
    s = s + " Student";
}

void greeting2(string &s) {
    s = s + " Student";
}

int main() {
    string s = "OED";

    greeting1(s);
    cout << s << endl;

    greeting2(s);
    cout << s << endl;

    return 0;
}
```

```
/tmp/W2nShy3ptI.o
OED
OED Student
```

40



Part 1:

Topical Presentation

3. Introduction to Function Procedures

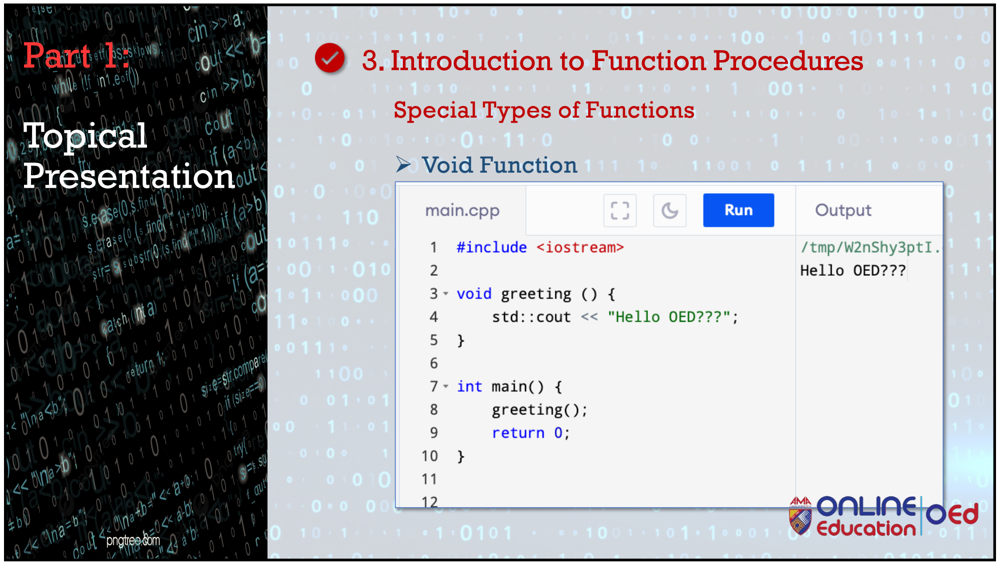
The Main Function

main.cpp

```cpp
#include <iostream>

int main() {

    std::cout << "Hello world!";

    return 0;
}
```

- Main function is the entry point of the program
- Main function must be global
- Main function may or may not accept accept an argument
- MUST return an integer

41



Part 1:

Topical Presentation

3. Introduction to Function Procedures

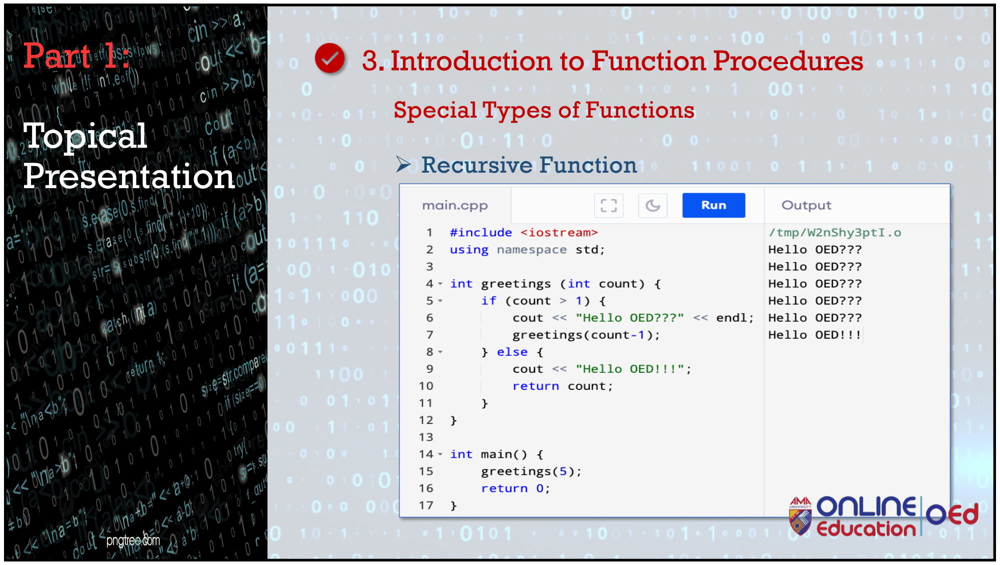
Special Types of Functions

- Void Function

```cpp
#include <iostream>

void greeting () {
    std::cout << "Hello OED???";
}

int main() {
    greeting();
    return 0;
}

```

```
/tmp/W2nShy3ptI.
Hello OED???
```

42



Part 1:

Topical Presentation

3. Introduction to Function Procedures

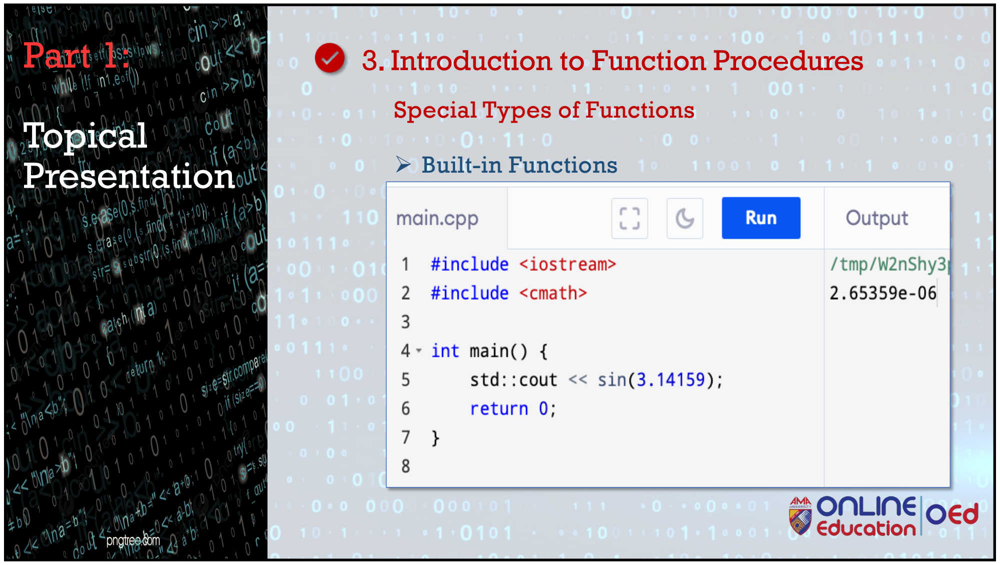
Special Types of Functions

➢ Recursive Function

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

int greetings (int count) {
    if (count > 1) {
        cout << "Hello OED???" << endl;
        greetings(count-1);
    } else {
        cout << "Hello OED!!!";
        return count;
    }
}

int main() {
    greetings(5);
    return 0;
}
```

```
/tmp/W2nShy3ptI.o
Hello OED???
Hello OED???
Hello OED???
Hello OED???
Hello OED!!!
```

43



Part 1:

Topical Presentation

3. Introduction to Function Procedures

Special Types of Functions

➢ Built-in Functions

```
#include <iostream>
#include <cmath>

int main() {
    std::cout << sin(3.14159);
    return 0;
}
```

```
/tmp/W2nShy3
2.65359e-06
```

44

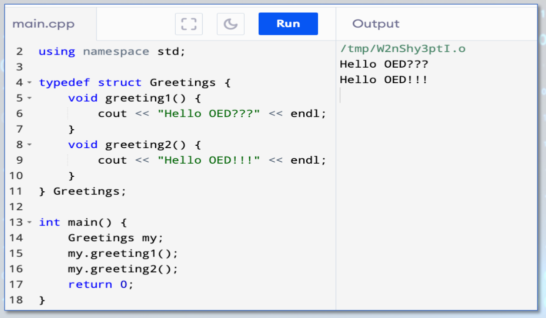
Part 1:

Topical Presentation

3. Introduction to Function Procedures

Special Types of Functions

➢ Methods



```
using namespace std;

typedef struct Greetings {
    void greeting1() {
        cout << "Hello OED???" << endl;
    }
    void greeting2() {
        cout << "Hello OED!!!" << endl;
    }
} Greetings;

int main() {
    Greetings my;
    my.greeting1();
    my.greeting2();
    return 0;
}
```

```
/tmp/W2nShy3ptI.o
Hello OED???
Hello OED!!!
```

45

Part 1:

Topical Presentation

3. Introduction to Function Procedures

Default Values and Overloading

Default Values are the values given to a parameter when no argument is provided to it in the function call.

Overloading is way of assigning different functionalities to a function of the same name, depending upon the type and number of parameters.

An example will make this clearer.

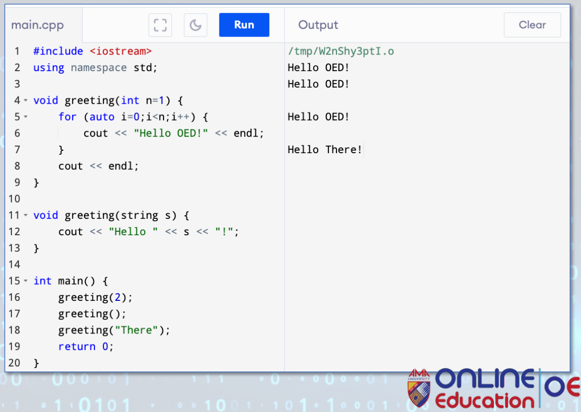
46

Part 1:

Topical Presentation

3. Introduction to Function Procedures

Default Values and Overloading (Example)



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

void greeting(int n=1) {
    for (auto i=0;i<n;i++) {
        cout << "Hello OED!" << endl;
    }
    cout << endl;
}

void greeting(string s) {
    cout << "Hello " << s << "!";
}

int main() {
    greeting(2);
    greeting();
    greeting("There");
    return 0;
}
```

```
/tmp/W2nShy3ptI.o
Hello OED!
Hello OED!

Hello OED!

Hello There!
```

47

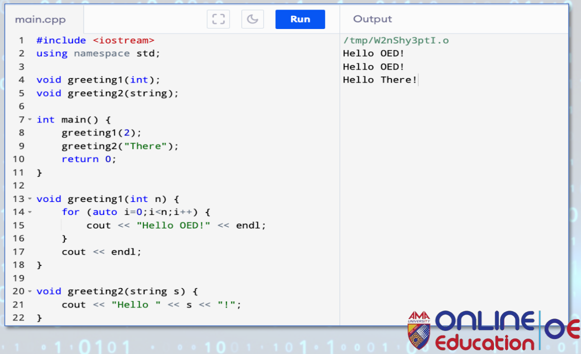
Part 1:

Topical Presentation

3. Introduction to Function Procedures

Function Declaration / Prototyping

Prototyping is a declaration of a function without implementation.



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

void greeting1(int);
void greeting2(string);

int main() {
    greeting1(2);
    greeting2("There");
    return 0;
}

void greeting1(int n) {
    for (auto i=0;i<n;i++) {
        cout << "Hello OED!" << endl;
    }
    cout << endl;
}

void greeting2(string s) {
    cout << "Hello " << s << "!";
}
```

```
/tmp/W2nShy3ptI.o
Hello OED!
Hello OED!
Hello There!
```

48

Part 1:

Topical Presentation

3. Introduction to Function Procedures

Function Signature and Function Pointers

Function Signature is the function declaration strip-out of the function and parameter names.

```
int calc(string text,double num);
```

Function Pointer is a pointer to a function with the

```
void              (int,string);
```

An example will make this clearer.

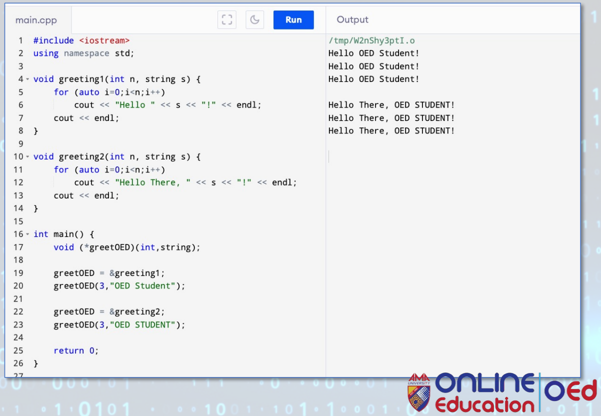
49

Part 1:

Topical Presentation

3. Introduction to Function Procedures

Function Pointer (Example)



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

void greeting1(int n, string s) {
    for (auto i=0;i<n;i++)
        cout << "Hello " << s << "!" << endl;
    cout << endl;
}

void greeting2(int n, string s) {
    for (auto i=0;i<n;i++)
        cout << "Hello There, " << s << "!" << endl;
    cout << endl;
}

int main() {
    void (*greetOED)(int,string);

    greetOED = &greeting1;
    greetOED(3,"OED Student");

    greetOED = &greeting2;
    greetOED(3,"OED STUDENT");

    return 0;
}
```

```
/tmp/W2nShy3ptI.o
Hello OED Student!
Hello OED Student!
Hello OED Student!

Hello There, OED STUDENT!
Hello There, OED STUDENT!
Hello There, OED STUDENT!
```

50

Part 1:

Topical Presentation

3. Introduction to Function Procedures

Lambda Function

A lambda function is a way of defining function inside another function.

```
auto function_name = [] (parameters) {
    function_body
};
```

[]   [=]   [&]   [&x,y]

An example will make this clearer.

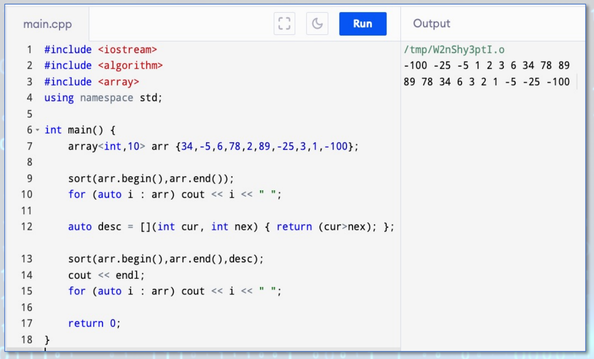
51

Part 1:

Topical Presentation

3. Introduction to Function Procedures

Lambda Function



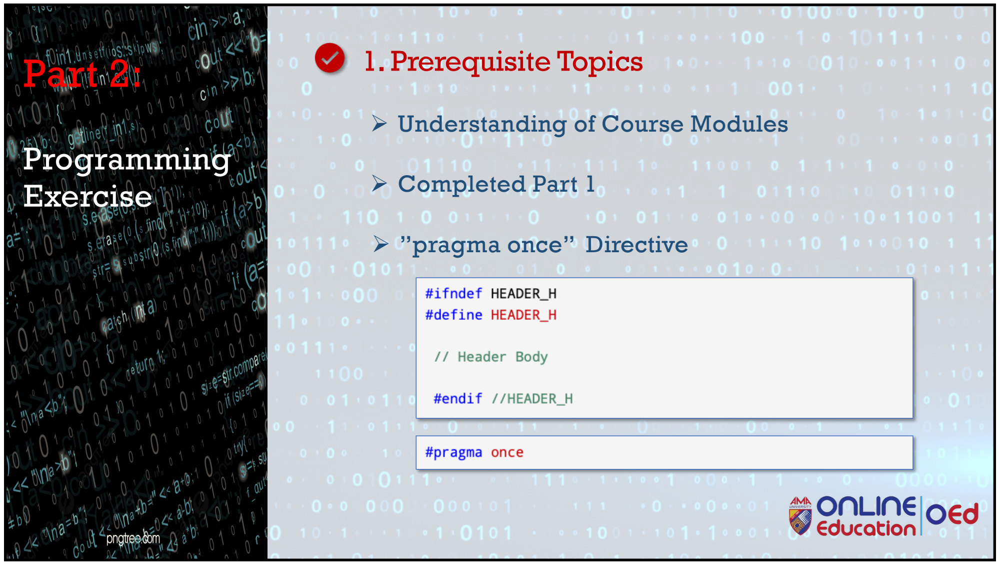
```
#include <iostream>
#include <algorithm>
#include <array>
using namespace std;

int main() {
    array<int,10> arr {34,-5,6,78,2,89,-25,3,1,-100};

    sort(arr.begin(),arr.end());
    for (auto i : arr) cout << i << " ";

    auto desc = [](int cur, int nex) { return (cur>nex); };

    sort(arr.begin(),arr.end(),desc);
    cout << endl;
    for (auto i : arr) cout << i << " ";

    return 0;
}
```

```
/tmp/W2nShy3ptI.o
-100 -25 -5 1 2 3 6 34 78 89
89 78 34 6 3 2 1 -5 -25 -100
```

52

END OF PART #1

53



Part 2:

Programming Exercise

pngtree.com

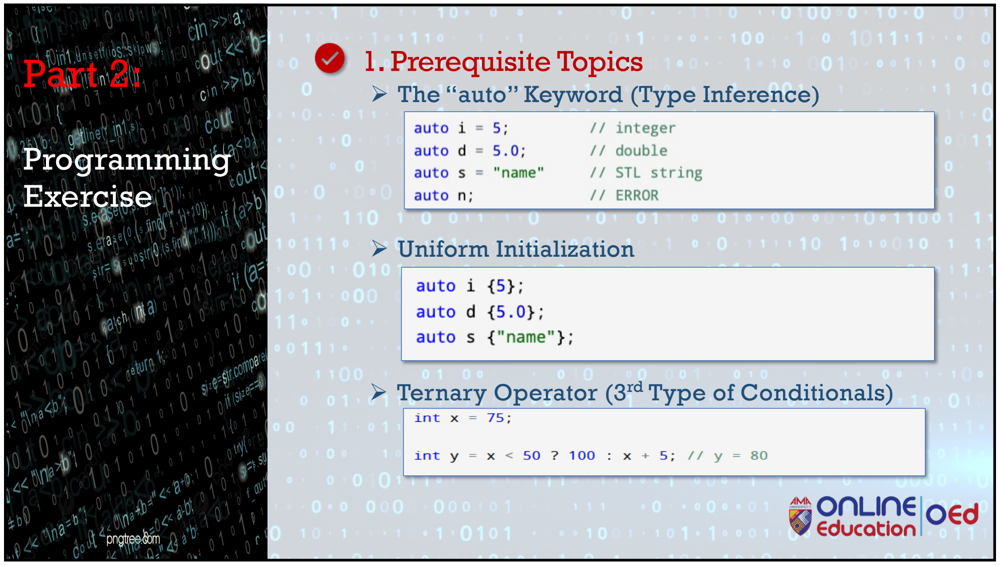
1. Prerequisite Topics

- Understanding of Course Modules
- Completed Part 1
- ”pragma once” Directive

```
#ifndef HEADER_H
#define HEADER_H

 // Header Body

 #endif //HEADER_H
```

```
#pragma once
```

54



Part 2:

Programming Exercise

1. Prerequisite Topics

- The “auto” Keyword (Type Inference)

```
auto i = 5;         // integer
auto d = 5.0;       // double
auto s = "name"     // STL string
auto n;             // ERROR
```

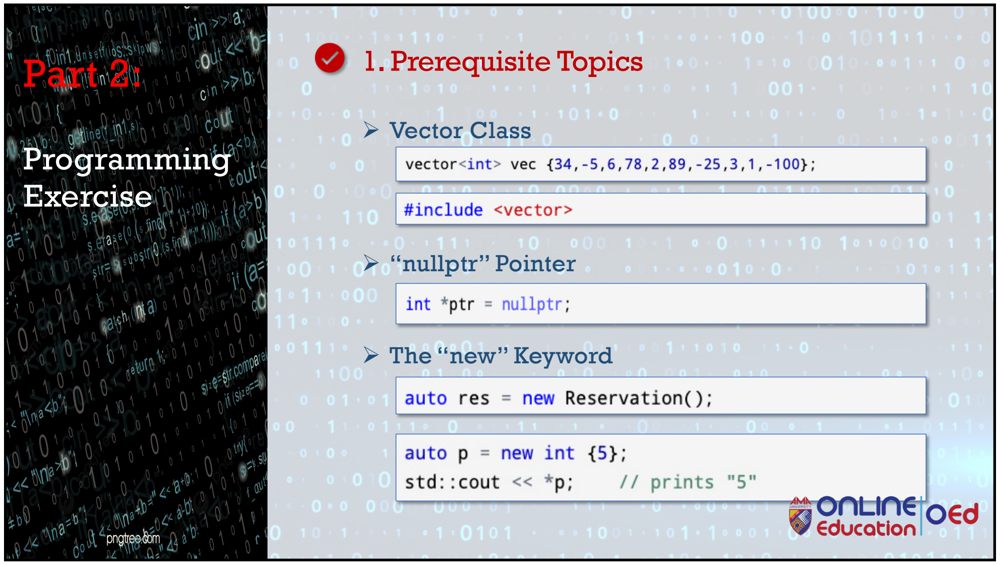
- Uniform Initialization

```
auto i {5};
auto d {5.0};
auto s {"name"};
```

- Ternary Operator (3rd Type of Conditionals)

```
int x = 75;

int y = x < 50 ? 100 : x + 5; // y = 80
```

55



Part 2:

Programming Exercise

1. Prerequisite Topics

- Vector Class

```
vector<int> vec {34,-5,6,78,2,89,-25,3,1,-100};
```

```
#include <vector>
```

- “nullptr” Pointer

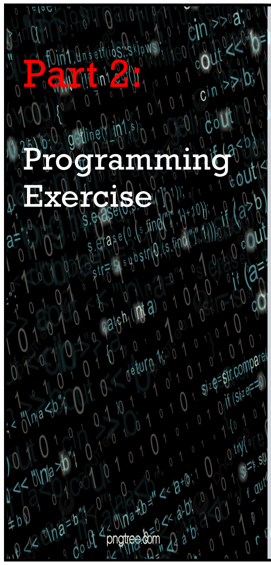
```
int *ptr = nullptr;
```

- The “new” Keyword

```
auto res = new Reservation();
```

```
auto p = new int {5};
std::cout << *p;     // prints "5"
```

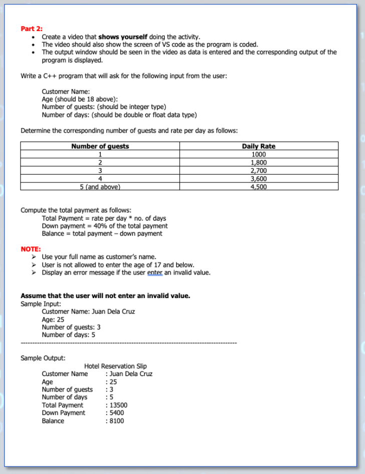
56



Part 2:

Programming Exercise

2. Statement of the Problem



Part 2:

- Create a video that **shows yourself** doing the activity.
- The video should also show the screen of VS code as the program is coded.
- The output window should be seen in the video as data is entered and the corresponding output of the program is displayed.

Write a C++ program that will ask for the following input from the user:

Customer Name:
Age (should be 18 above):
Number of guests: (should be integer type)
Number of days: (should be double or float data type)

Determine the corresponding number of guests and rate per day as follows:

Number of guests	Daily Rate
1	1000
2	1,800
3	2,700
4	3,600
5 (and above)	4,500

Compute the total payment as follows:

Total Payment = rate per day * no. of days
Down payment = 40% of the total payment
Balance = total payment – down payment

NOTE:

- ➢ Use your full name as customer's name.
- ➢ User is not allowed to enter the age of 17 and below.
- ➢ Display an error message if the user enter an invalid value.

Assume that the user will not enter an invalid value.

Sample Input:

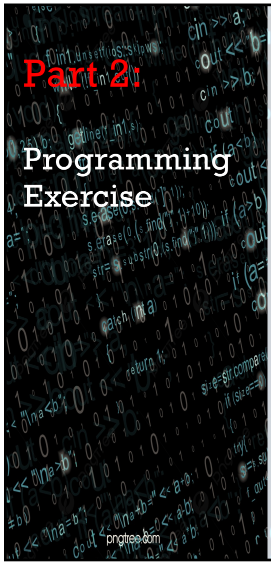
```
        Customer Name: Juan Dela Cruz
        Age: 25
        Number of guests: 3
        Number of days: 5
------------------------------------------------------------------------------------
```

Sample Output:

```
                         Hotel Reservation Slip
        Customer Name        : Juan Dela Cruz
        Age                  : 25
        Number of guests     : 3
        Number of days       : 5
        Total Payment        : 13500
        Down Payment         : 5400
        Balance              : 8100
```

ONLINE Education OEd

57



Part 2:

Programming Exercise

2. Solution to Programming Exercise

First, instead of re-writing the code. I will present to you the finished solution and then we will explain each code one-by-one. This is to save time and minimize error.

Second, I will not be using VSCode. For small projects, I prefer to use a ”pure” text editor, so that I would have better control of the compiler commands. I will be using Sublime Text. And, for compilation I will be using the g++ version provided by Clang, which in turn is the default C++ compiler suit in MacOS.

ONLINE Education OEd

58

Thank YOU!!!

59