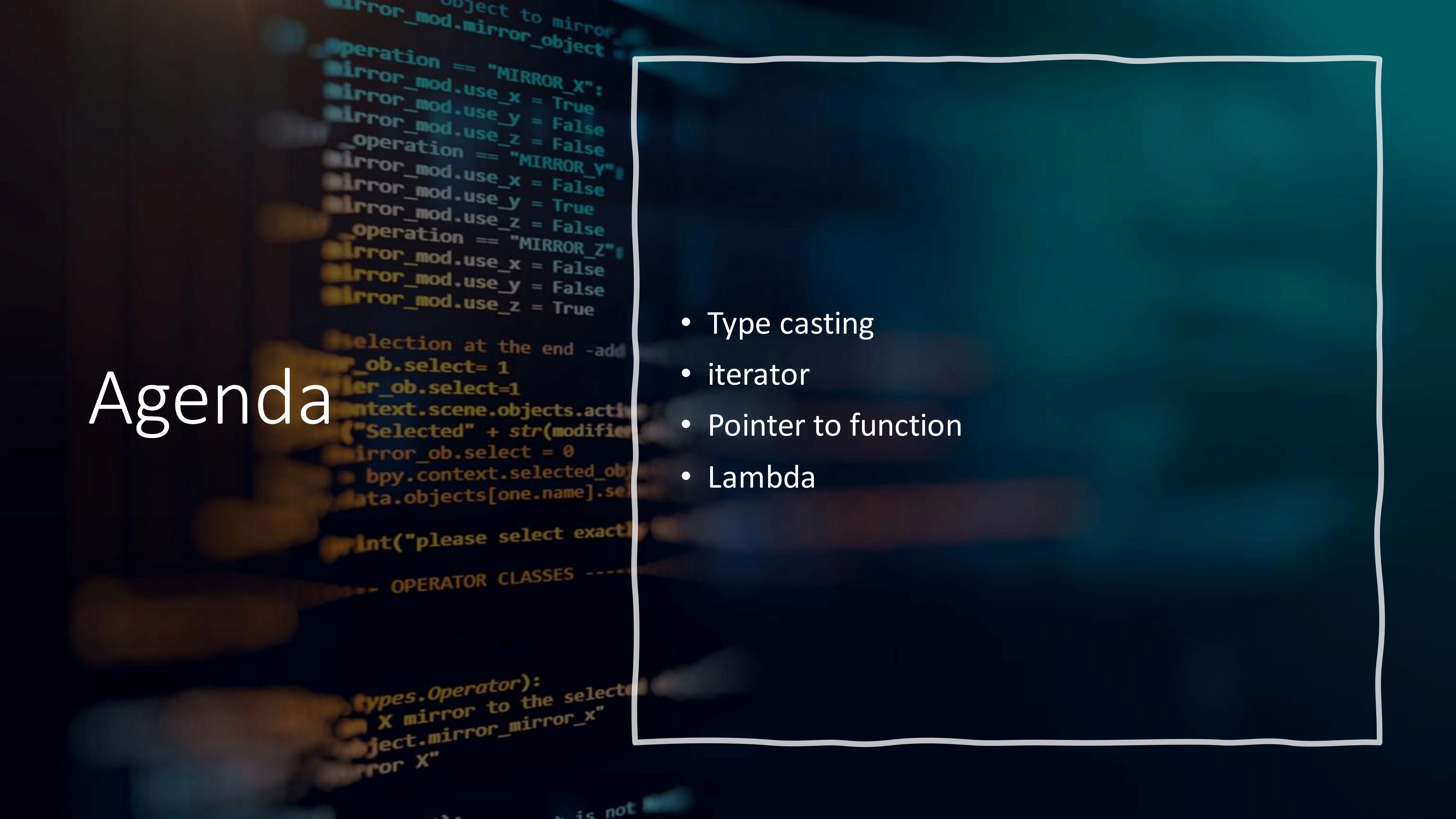
Session 4

Mohamed Saied



Agenda

- Type casting
- iterator
- Pointer to function
- Lambda

Implicit Conversion

- Done by the compiler on its own, without any external trigger from the user.
- Generally takes place when in an expression more than one data type is present. In such condition type conversion (type promotion) takes place to avoid lose of data.
- All the data types of the variables are upgraded to the data type of the variable with largest data type.
- bool -> char -> short int -> int ->
- unsigned int -> long -> unsigned ->
- long long -> float -> double -> long double

Implcit conversion

```
// An example of implicit conversion

#include <iostream>
using namespace std;

int main()
{
    int x = 10; // integer x
    char y = 'a'; // character c

    // y implicitly converted to int. ASCII
    // value of 'a' is 97
    x = x + y;

    // x is implicitly converted to float
    float z = x + 1.0;

    cout << "x = " << x << endl
         << "y = " << y << endl
         << "z = " << z << endl;

    return 0;
}
```

Output:

```
x = 107 y = a z = 108
```

Explicit conversion

- Explicit Type Conversion: This process is also called type casting and it is user-defined. Here the user can typecast the result to make it of a particular data type.
- In C++, it can be done by two ways:

- Converting by assignment: This is done by explicitly defining the required type in front of the expression in parenthesis. This can be also considered as forceful casting.
- Syntax:

- (type) expression
- where type indicates the data type to which the final result is converted

Explicit conversion

```
// C++ program to demonstrate
// explicit type casting

#include <iostream>
using namespace std;

int main()
{
    double x = 1.2;

    // Explicit conversion from double to int
    int sum = (int)x + 1;

    cout << "Sum = " << sum;

    return 0;
}
```

Explicit conversion

- Conversion using Cast operator: A Cast operator is an unary operator which forces one data type to be converted into another data type.
- C++ supports four types of casting:
- Static Cast : `int b = static_cast<int>(f);`
- Dynamic Cast
- Const Cast `int* c1 = const_cast <int *> (b1);`
- Reinterpret Cast : `data_type *var_name = reinterpret_cast <data_type *>(pointer_variable);`

Const Cast

- const_cast is considered safer than simple type casting. It’safer in the sense that the casting won’t happen if the type of cast is not same as original object. For example, the following program fails in compilation because ‘int *’ is being typecasted to ‘char *’

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

int main(void)
{
   int a1 = 40;
   const int* b1 = &a1;
   char* c1 = const_cast <char *> (b1); // compiler error
   *c1 = 'A';
   return 0;
}
```

output:

```
prog.cpp: In function ‘int main()’: prog.cpp:8: error: invalid
const_cast from type 'const int*' to type 'char*'
```