Week 6

(OOP Intro-Encapsulation)

Agenda

- Why OOP ?
- What is OOP ?
- Classes
- Objects
- Declaring Classes
- Accessing Class Members
- Encapsulation
- Setters and Getters
- Constructors
- Destructors

1 Why OOP ?

???

Why OOP ?

If you want to store student name :

```
string StudentName;
cin >> StudentName;
```

Why OOP ?

Now , if you need to store his name , id , age , and his marks ?

```
string StudentName;
int id,age;
float arabic, math, english;

cin >> StudentName;
cin >> id >> age;
cin >> arabic, math, english;
```

Why OOP ?

Now , what if you want to store all class students data ?

```
string StudentName2;
int id2, age2;
float arabic2, math2, english2;

cin >> StudentName2;
cin >> id2 >> age2;
cin >> arabic2, math2, english2;

/*
to the last class student
.
.
.
*/
string StudentName40;
int id40, age40;
float arabic40, math40, english40;

cin >> StudentName40;
cin >> id40 >> age40;
cin >> arabic40, math40, english40;
```

Why OOP ?

- Now , store all school classes students ?

A very long code , with many many variables , right ?

Why OOP ?

- To solve that we need to organize our code by making a student class and reuse the **class** code by making many **objects** :

  (**Student s1, s2, s3 , ..., s1000;**)

- This method **OOP** is depend on **Reality Simulation** concept .

2

What is OOP ?

???

What is OOP ?

- **OOP : Object Oriented Programming**
  A **Style** of programming based on object concept, it **doesn't** refer to a specific language but it is a way to build a program by making **Simulation of Reality**

What is OOP ?

- **Classes and Objects :**

```
class Student
{
    string StudentName;
    int id, age;
    float arabic, math, english;
};

//the objects :
Student s1, s2, s3, s1000;
```

3 Classes

What is the class ?

Classes

- **Blueprint** from which Objects are created
- A user defined **Data-type**
- The class has :
  1. **Attributes** (his data) -variables-
  2. **Methods** (his behavior) –functions-
  3. **Constructors** (a special method member)
  4. **Destructors** (a special method member)
- Can **hide** data and methods
- Provide public interface
- Examples :
  - Account
  - Student
  - Image

4 Objects

What is the Object ?

Objects

- **Created** from a class
- Represents a specific **instance** of a class
- Can create **many many** objects from one class
- The objects has :
  - It's own **identity**

    (It's own data, methods, Constructors, Destructor)
- Each can use the defined class methods
- Examples :
  - **Ahmed's bank account** is an instance of an Account
  - **Sara's bank account** is an instance of an Account
    - And each has it's own balance , can make deposits, withdrawals, ...

5

Declaring Class

How to define a class ?

Declaring Class

```
class Class_name
{
    // declaration(s);
};
```

- Note : it's known between developers that the first letter of class is a **Capital** letter, so that we can different between classes (capital letter), and functions (all letters are small)

6

Accessing Class members

How to access a class ?

Accessing Class members

- We use dot operator (.) to access public members at main function :

```
class Student
{
public:
    string StudentName;
    int id, age;
    float arabic, math, english;
};

int main()
{
//the objects :
Student s1, s2, s3, s1000;

s2.StudentName = "Abd elrahman";
s2.math = 50;
s1.
}
```

```
age
arabic
english
id
math
StudentName
```

(field) public : int Student::age
File: ITI-Week6.cpp

Ex 1 :

- Create account class with :
  - Attributes :
    - Name
    - Id
    - balance
  - One Method :
    - Printing info
- And then create objects in main function

Quiz 1 :

- Create Student class with :
  - Attributes :
    - Name
    - Id
    - Arabic Grade, Math Grade
  - One Method :
    - Printing info
- And then create objects in main function and ask the user to insert Student data

7

Encapsulation

To hide data

Encapsulation

- **Encapsulation :** the first OOP concepts, it represents the ability of OOP to **hide** or **public** objects data
- **Access Modifiers :**
  - **Public** : public members are accessible **everywhere**
  - **Private** : private members are only accessible by **class members**
  - **Protected** : used with **inheritance** (next session)
- Usually we make all attributes **private** members to secure them from any external access (private balance;)
- And we make **public** methods for each attribute to be accessed by certain users (public show_balance( ) { cout<<balance; })

What is The default Access Modifiers in :

- Class ?
- Struct ?

What is The default Access Modifiers in :

- Class : Private
- Struct : Public

8

Setters and Getters

To hide data

Setters

- A method that we create for each attribute to control it's **enter** access

```
class Account
{
private:
    float balance;

public:
    void set_balance(float b) {
        balance = b;
    }
};

int main()
{
    Account osama;
    osama.set_balance(1000);
}
```

Note that :

- Setters functions are always has no return type (**void**) , and always **has parameters** , all what it do is to set value of it's attribute

Getters

- A method that we create for each attribute to control it's **show** access

```
class Account
{
private:
    float balance;

public:
    void set_balance(float b) {
        balance = b;
    }
    float get_balance() {
        return balance;
    }
};

int main()
{
    Account osama;
    osama.set_balance(1000);
    cout << osama.get_balance();
}
```

Note that :

- Getters functions are always has a return type is the **same** type of it's getting attribute and **has no parameters** and all what it do is to return it

Ex 2 :

Create an account class with :

- attributes :

  Name, id, balance

- methods :

  setters and getters for each attribute and a method to print all info

Quiz 2 :

Create a student class with :

- attributes :

  Name, id, Arabic marks
  Math marks

- methods :

  setters and getters for each attribute and a method to print all info

9 Constructors

At creation

Constructors

- A special member method :
  - Has **no return type**
  - His name is **same** as class name
  - Self **invoked automatically** during object creation
- Default Constructor is useful for **initialization**
- Constructor can be **overloaded** and we can use Parameterized constructor to send data at creation
- If you don't create any constructor, **C++ will create** the default constructor for you, but it will be empty without any code or logic
- But if you create a parameterized constructor without creating default constructor, **C++ will not create** create it and you will be asked to send data at each time as you will use the parameterized constructor
- So don't create any parameterized constructor **before** creating the default one

Constructors

```
class Account
{
private:
    string name;
    int id;
    float balance;
public:
    //Default constructor
    Account() {
        name = " ";
        id = 0;
        balance = 0;
    }
    //Parameterized constructor
    Account(string n, int i, float b) {
        name = n;
        id = i;
        balance = b;
    }
}
```

```
int main(){

Account a1; //default cons is called
a1.print_info();

Account a2("amira",3,20000); //Para cons is called
a2.print_info();
}
```

```
Microsoft Visual Studio Debug Console
Customer Name:
id  : 0
Balance = 0
================================
Customer Name: amira
id  : 3
Balance = 20000
================================
```

10

Destructors

At Destroying

Destructors

- A special member method :
  - Has **no return type**
  - His name is **same** as class name with (**~**) sign
  - Self **invoked automatically** when object is destroyed

- Default destructor is useful for **releasing memory**

- Destructor can not be **overloaded** only one destructor at the class

- If you don't create destructor, **C++ will create** it for you, but it will be empty without any code or logic

Destructors

```
class Account
{
private:
    string name;
    int id;
    float balance;
public:
    //Default constructor
    Account() {
        name = " ";
        id = 0;
        balance = 0;
    }
    //Parameterized constructor
    Account(string n, int i, float b) {
        name = n;
        id = i;
        balance = b;
    }
    //Destructor
    ~Account() {

    }
}
```

Ex 3 :

Create an account class with :

- Default constructor
- Parameterized constructor
- Destructor and print inside it “Bye Bye”

◇ attributes :

Name, id, balance

◇ methods :

setters and getters for each attribute and a method to print all info

Quiz 3 :

Create a student class with :

- Default constructor
- Parameterized constructor
- Destructor and print inside it “Bye Bye”

◇ attributes :

Name, id, Arabic marks
Math marks

◇ methods :

setters and getters for each attribute and a method to print all info

Adding class in IDE

- Visual Studio :
  Right click on source file , choose add class

- Code blocks or dev :

```
#ifndef class_name
#define class_name
class class_name
{
//code
};
#endif
```

- In main function or in cpp file:
  #include “class_name.h”

Quiz 4 :

Create a rectangle class contains:

- Height and width as attributes
- Setters , getters as methods to get area calculation
- Default constructor
- Parameterized constructor
- Destructor and print inside it “Bye Bye”

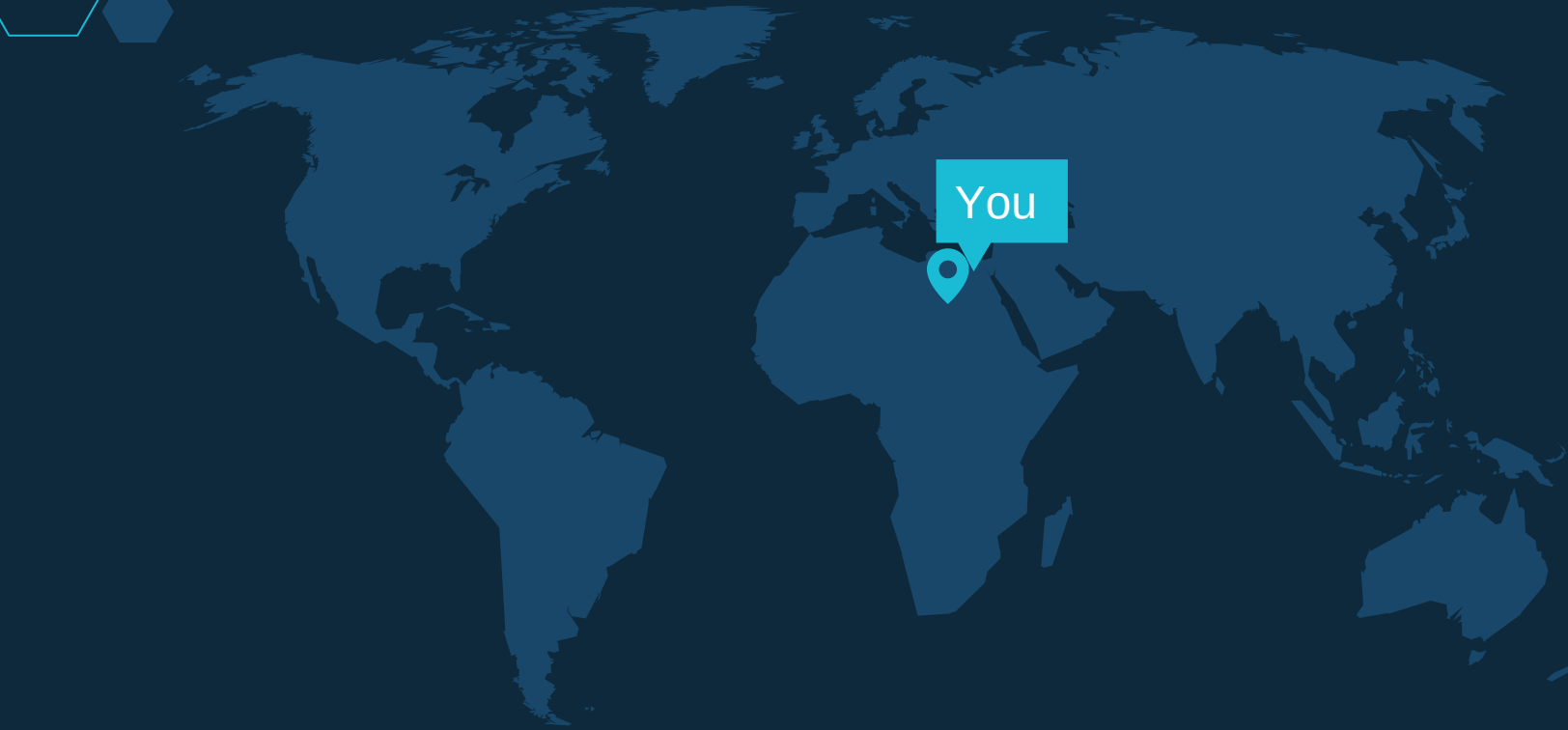
Thanks!

Any questions?

You can find me at:

- itiroutealexiti.com

Be sure that you are making a difference .




Break

Week 7

(OOP Inheritance-Static Members and Methods)

Agenda

- Why Inheritance ?
- What is Inheritance ?
- Deriving class from existing classes
- Protected members
- Constructors in Inheritance
- Passing Arguments to Parameterized Constructors
- Destructors in Inheritance
- Methods Overriding
- Inheritance Types
- Inheritance Vs Composition
- Static Members
- Static Methods

1

Why Inheritance ?

???

Why Inheritance ?

If you have a person class, and you want to add student and teacher class, what will be your code ?

```
class Person {
private:
    string name;
    int id;
public:
    void set_name(string n) {
        name = n;
    }
    void set_id(int i) {
        id = i;
    }
    string get_name() {
        return name;
    }
    int get_id() {
        return id;
    }
};
```

```
class Student {
private:
    string name;
    int id;
    float total_marks;
public:
    void set_name(string n) {
        name = n;
    }
    void set_id(int i) {
        id = i;
    }
    void set_total_marks(int t) {
        total_marks = t;
    }
    string get_name() {
        return name;
    }
    int get_id() {
        return id;
    }
    float get_total_marks() {
        return total_marks;
    }
};
```

```
class Teacher {
private:
    string name;
    int id;
    double salary;
public:
    void set_name(string n) {
        name = n;
    }
    void set_id(int i) {
        id = i;
    }
    void set_salary(double s) {
        salary = s;
    }
    string get_name() {
        return name;
    }
    int get_id() {
        return id;
    }
    double get_salary() {
        return salary;
    }
};
```

Why Inheritance ?

- So we **duplicate** the same data of person into student and teacher classes, this is against **DRY** concept.
- So we use **inheritance** to create **new** classes from **existing** classes
- The new class (**child class**) have the same data and methods of the existing class (**parent class**)
- Allow child class to **modify** parent methods (Overriding) **without modify** original class

```
class Person {
protected:
    string name;
    int id;
public:
    void set_name(string n) {
        name = n;
    }
    void set_id(int i) {
        id = i;
    }
    string get_name() {
        return name;
    }
    int get_id() {
        return id;
    }
};
```

```
class Student:public Person {
private:
    float total_marks;
public:
    void set_total_marks(int t) {
        total_marks = t;
    }
    float get_total_marks() {
        return total_marks;
    }
};
```

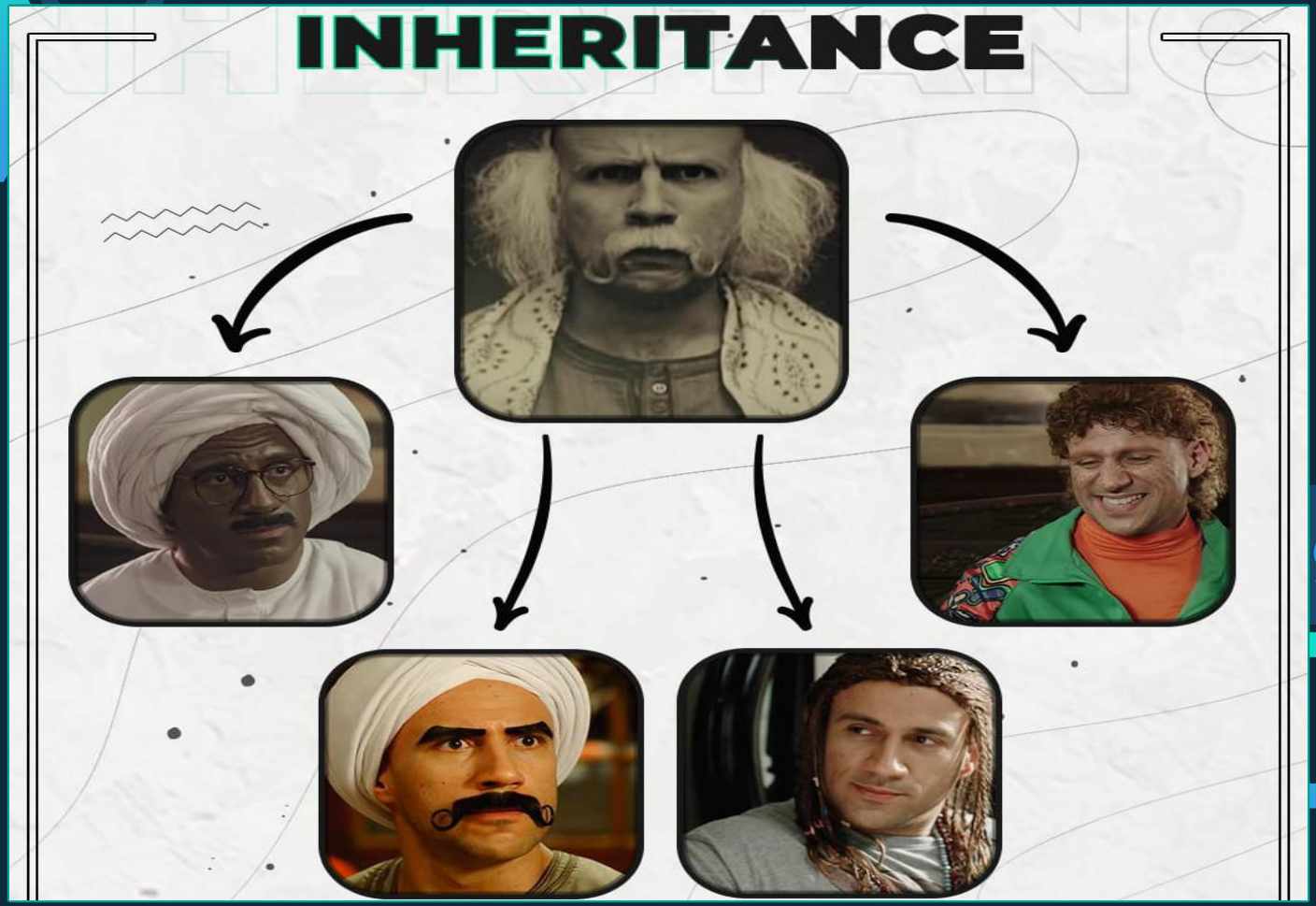
```
class Teacher :public Person {
private:
    double salary;
public:
    void set_salary(double s) {
        salary = s;
    }
    double get_salary() {
        return salary;
    }
};
```

2

What is Inheritance ?

???

What is Inheritance?



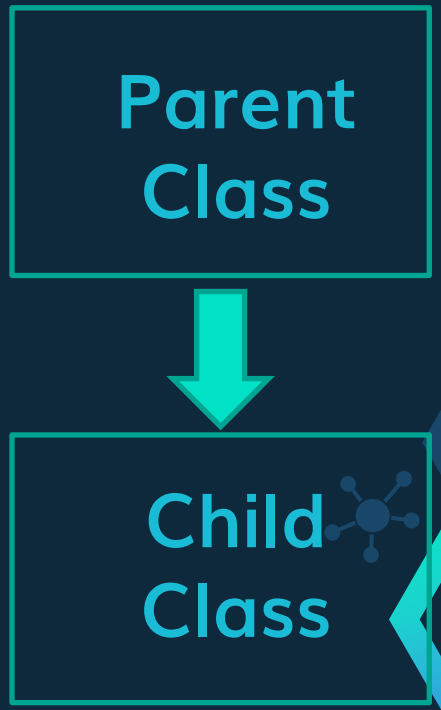
What is Inheritance?

- **Inheritance :** the **Second** OOP concepts that allows us to create **new** classes from **existing** classes, the child class will be able to inherit all parent data and methods and could **modify** the inherited parent's methods

What is Inheritance?

- **Parent Class (Base or Super Class) :** the class being inherited or extended from

- **Child Class (Derived or Sub Class) :** the class being created from parent class which will inherit all attributes and methods



3

Deriving class from existing classes

How to inherit ?

Deriving class from existing classes

```
class Parent{
    //Code
};
class Child : access_specifier Parent{
    //Code
};
```

- **Access specifier** can be : public, private, protected
- **Public** is the most common access modifier

Deriving class from existing classes

Parent Class

Public : a
Protected : b
Private : c

Public Inheritance

Child Class

Public : a
Protected : b
c : No Access

Deriving class from existing classes

Parent Class

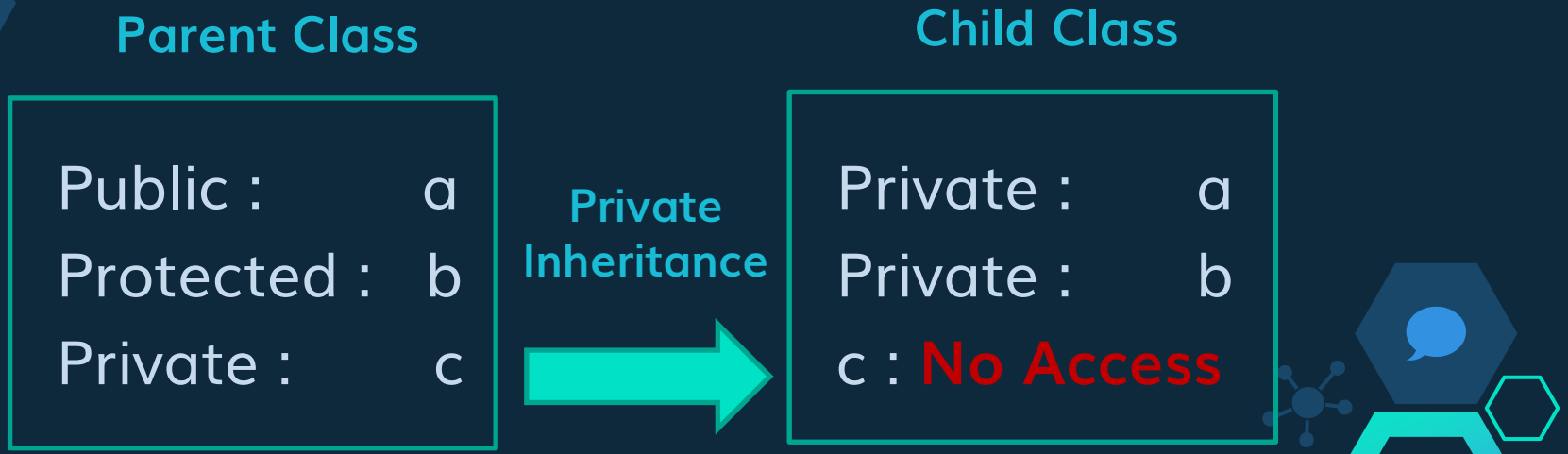
Public : a
Protected : b
Private : c

Protected Inheritance

Child Class

Protected : a
Protected : b
c : **No Access**

Deriving class from existing classes



4

Protected members

Only child can access

Protected members

```
class Parent{

protected:
    string title;
    int id;
};
```

- Protected Members Are accessible only by:
  - Parent class members
  - Child class members

Ex 1:

Create a Person class that includes :

- Attributes :
  - String name, int id
- Methods :
  - Setters, Getters

Then create Teacher class which inherit from Person

All data and methods and has :

- Attributes :
  - Float Salary
- Methods :
  - Setters, Getters
  - Get annual salary (12*salary)
  - Get raise (0.07*salary)

Then create Student class which inherit from Person

All data and methods and has :

- Attributes :
  - Float total marks
- Methods :
  - Setters, Getters

Quiz 1:

Create a Shape class that includes :

- Attributes :
  - Float height
- Methods : Setters, Getters
  - Calc_area (return 0;)

Then create Rectangle class which inherit from Shape

All data and methods and has :

- Attributes :
  - Float Width
- Methods :
  - Setters, Getters
  - Calc_area

  (return width*height;)

Then create Tringle class which inherit from Shape

All data and methods and has :

- Attributes :
  - Float base
- Methods :
  - Setters, Getters
  - Calc_area

  (return 0.5*base*height;)

5

Constructors in Inheritance

Which one is called firstly, Parent or Child Constructor ?

Constructors in Inheritance

```
class Parent{

public:
    Parent() {
        cout << "Parent Constructor" << endl;
    }
};

class Child : public Parent{

public:
    Child() {
        cout << "Child Constructor" << endl;
    }
};


int main()
{
    Child c;
}
```

```
Microsoft Visual Studio D
Parent Constructor
Child Constructor

E:\Courses\1. Programm
ce.exe (process 12684)
Press any key to close
```

The parent constructor is executed firstly then the child constructor executed

Passing Arguments to Parameterized Constructors in Inheritance

How to pass ?

Passing Arguments to Parameterized Constructors in Inheritance

```
class Parent{
private:
    string name;
    int age;
public:
    Parent();
    Parent(string n,int a) {
        name = n;
        age = a;
    }
};

class Child : public Parent{
private:
    double expenses;
public:
    Child();
    Child(string n,int a,double e):Parent(n,a) {
        expenses = e;
    }
};
```

7

Destructors in Inheritance

Which one is called firstly, Parent or Child Destructor ?

Destructors in Inheritance

```cpp
class Parent{

public:
    ~Parent() {
        cout << "Parent Destructor" << endl;
    }
};

class Child : public Parent{

public:
    ~Child() {
        cout << "Child Destructor" << endl;
    }
};


int main()
{
    Child c;
}
```

```
Microsoft Visual Stu
Child Destructor
Parent Destructor

E:\Courses\1. Progr
ce.exe (process 155
Press any key to cl
```

The child destructor is executed firstly then the Parent destructor executed

8

Methods Overriding

Child can override on parent methods

Methods Overriding

```
class Parent{

public:
    void hello() {
        cout << "Hi i am Parent" << endl;
    }
};

class Child : public Parent{

public:
    void hello() {
        cout << "Hi i am Child" << endl;
    }
};


int main()
{
    Parent p;
    Child c;

    p.hello();
    c.hello();
}
```

```
Microsoft Visual St
Hi i am Parent
Hi i am Child

E:\Courses\1. Prog
ce.exe (process 81
Press any key to c
```

Ex 2:

Create a Person class that includes :

- **Constructors, Destructor**
- Attributes :
  - String name, int id
- Methods :
  - Setters, Getters

Then create Employee class which inherit from Person

All data and methods and has :

- **Constructors, Destructor**
- Attributes :
  - Float Salary
- Methods :
  - Setters, Getters
  - Get annual salary (12*salary)
  - Get raise (0.07*salary)

Then create Student class which inherit from Person

All data and methods and has :

- **Constructors, Destructor**
- Attributes :
  - Float total marks
- Methods :
  - Setters, Getters

Quiz 2:

Create a Shape class that includes :

- **Constructors, Destructor**
- Attributes :
  - Float height
- Methods : Setters, Getters
  - Calc_area (return 0;)

Then create Rectangle class which inherit from Shape

All data and methods and has :

- **Constructors, Destructor**
- Attributes :
  - Float Width
- Methods :
  - Setters, Getters
  - Calc_area

  (return width*height;)

Then create Tringle class which inherit from Shape

All data and methods and has :

- **Constructors, Destructor**
- Attributes :
  - Float base
- Methods :
  - Setters, Getters
  - Calc_area

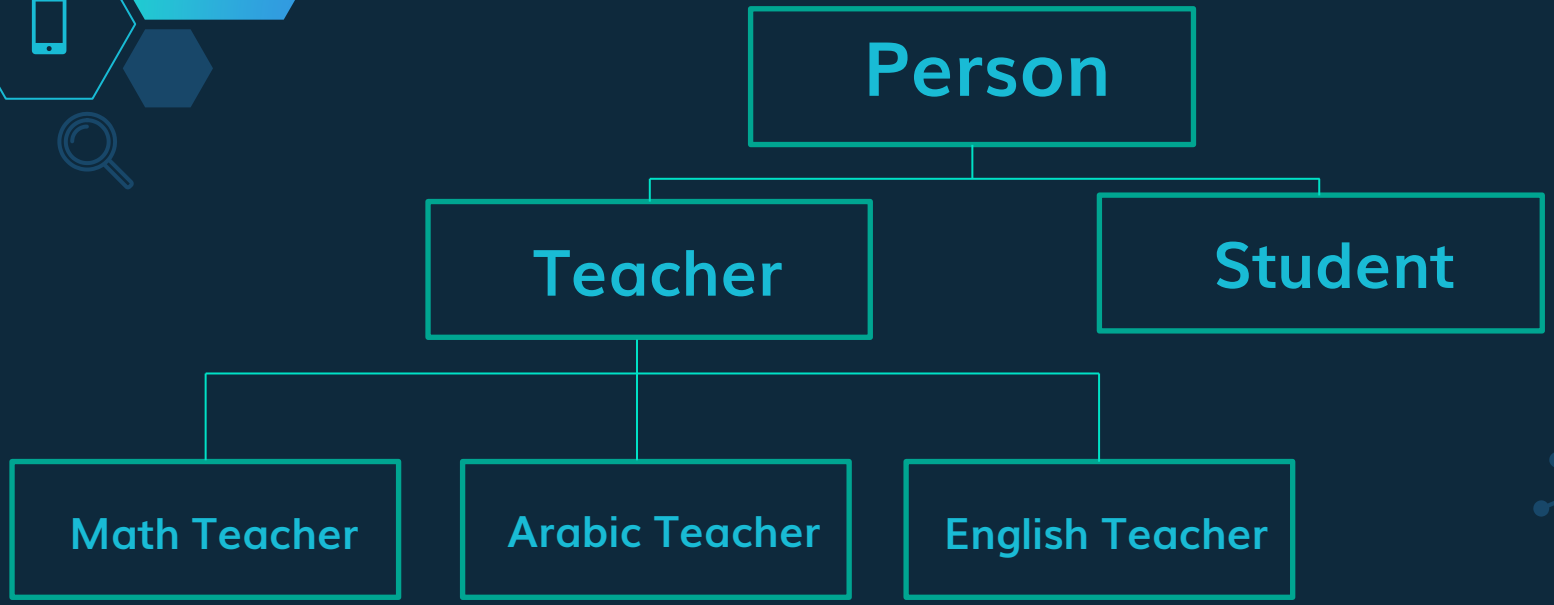
  (return 0.5*base*height;)

9

Inheritance Types

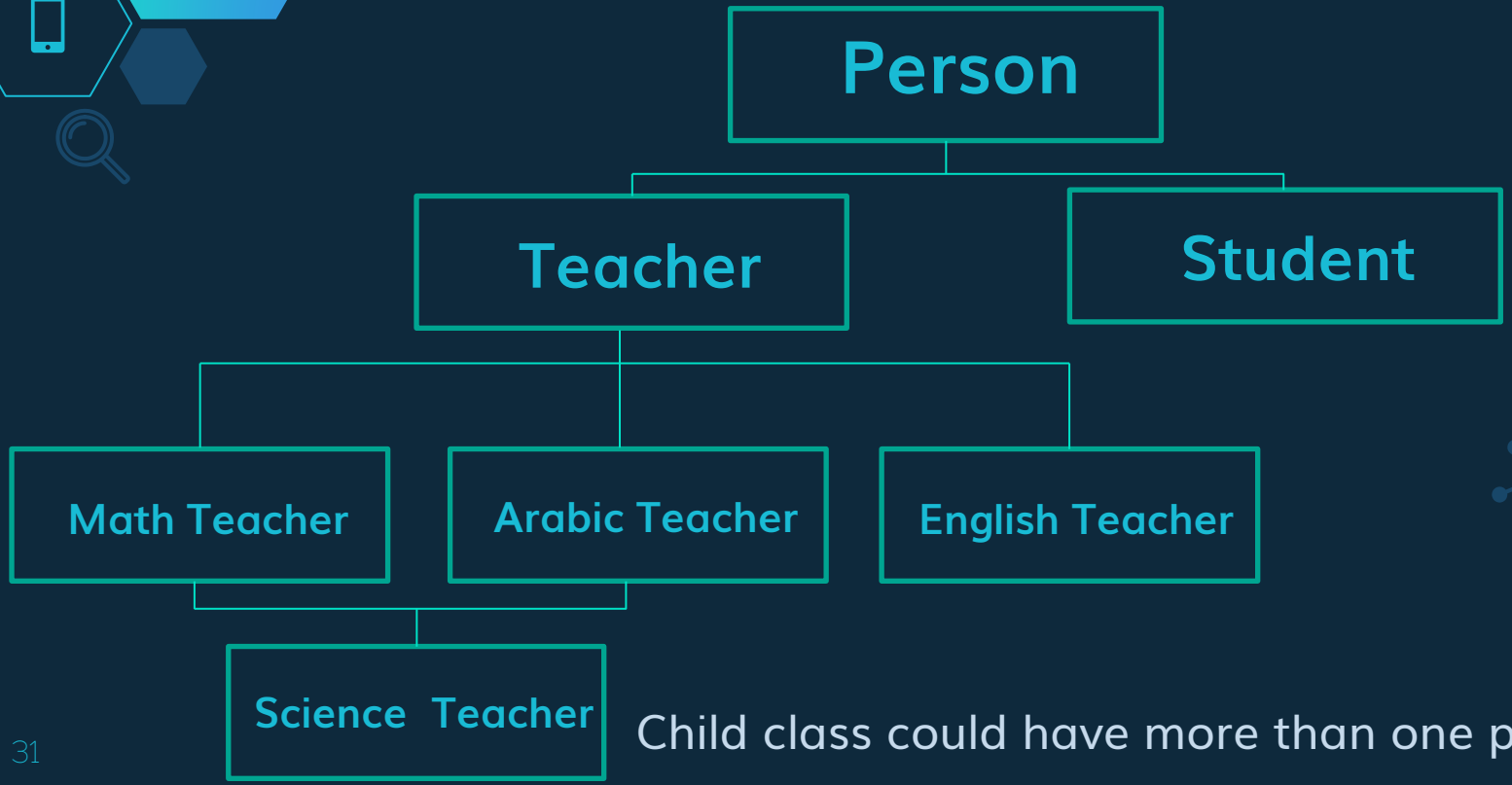
Multi-Level Inheritance , Multiple Inheritance

Multi-Level Inheritance (Single Inheritance)



Each child class has only one parent Class

Multiple Inheritance



Child class could have more than one parent Class

Multiple Inheritance

```
class Junior
{
public:
    void say_junior() {
        cout << "i am Junior" << endl;
    }
};

class Senior
{
public:
    void say_senior() {
        cout << "i am Senior" << endl;
    }
};

class Manager : public Junior, public Senior
{
public:
void say_manager() {
    cout << "i manage junior, and Senior" << endl;
}
};

int main()
{
    Manager m1;
    m1.say_junior();
    m1.say_senior();
    m1.say_manager();
}
```

```
Microsoft Visual Studio Debug Co
i am Junior
i am Senior
i manage junior, and Senior

E:\Courses\1. Programming\1.
ce.exe (process 9740) exited
Press any key to close this
```

Multiple Inheritance

Multiple Inheritance :

- **Not supported** by all programing Languages
- Other languages use **public interface** in case of more parents
- C++ **don't** have interfaces
- C++ **use** it instead of interface
- **Not recommended** to use
- **Causes The diamond problem (2 Copies of data)**

10

Inheritance Vs Composition

What is the difference ?

Inheritance Vs Composition

Public Inheritance:

- Achieve (**is a**) relationship
  - Employee is a person
  - Student is a person

Composition:

- Achieve (**has a**) relationship
  - Employee has a name
  - Student has marks

11

Static Members

All objects are related to it

Static Members

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

class Student
{
public:
    static int id;
    Student() {
        id++;
    }
    int getId() {
        return id;
    }
};

int Student::id = 0;

int main() {
    Student s1;
    cout << "S1 id = " << s1.getId() << endl;
    Student s2;
    cout << "S2 id = " << s2.getId() << endl;
    Student s3;
    cout << "S3 id = " << s3.getId() << endl;
}
```

```
Microsoft Visu
S1 id = 1
S2 id = 2
S3 id = 3

D:\Courses\1.
 17580) exited
Press any key
```

The static member are shared between all objects created from class

12

Static Methods

Could be accessed without creating an object

Static Methods

```
class Calculator {
public:
    int add(int a, int b) {
        return a + b;
    }

    static int multiply(int a, int b) {
        return a * b;
    }
};

int main()
{
    cout << Calculator::multiply(3, 5) << endl;
}
```

```
15

E:\Cours
ce.exe
Press a
```

The static method could be used without creating objects like(multiply function), but (add function) is not a static function, so it can only be used by creating objects

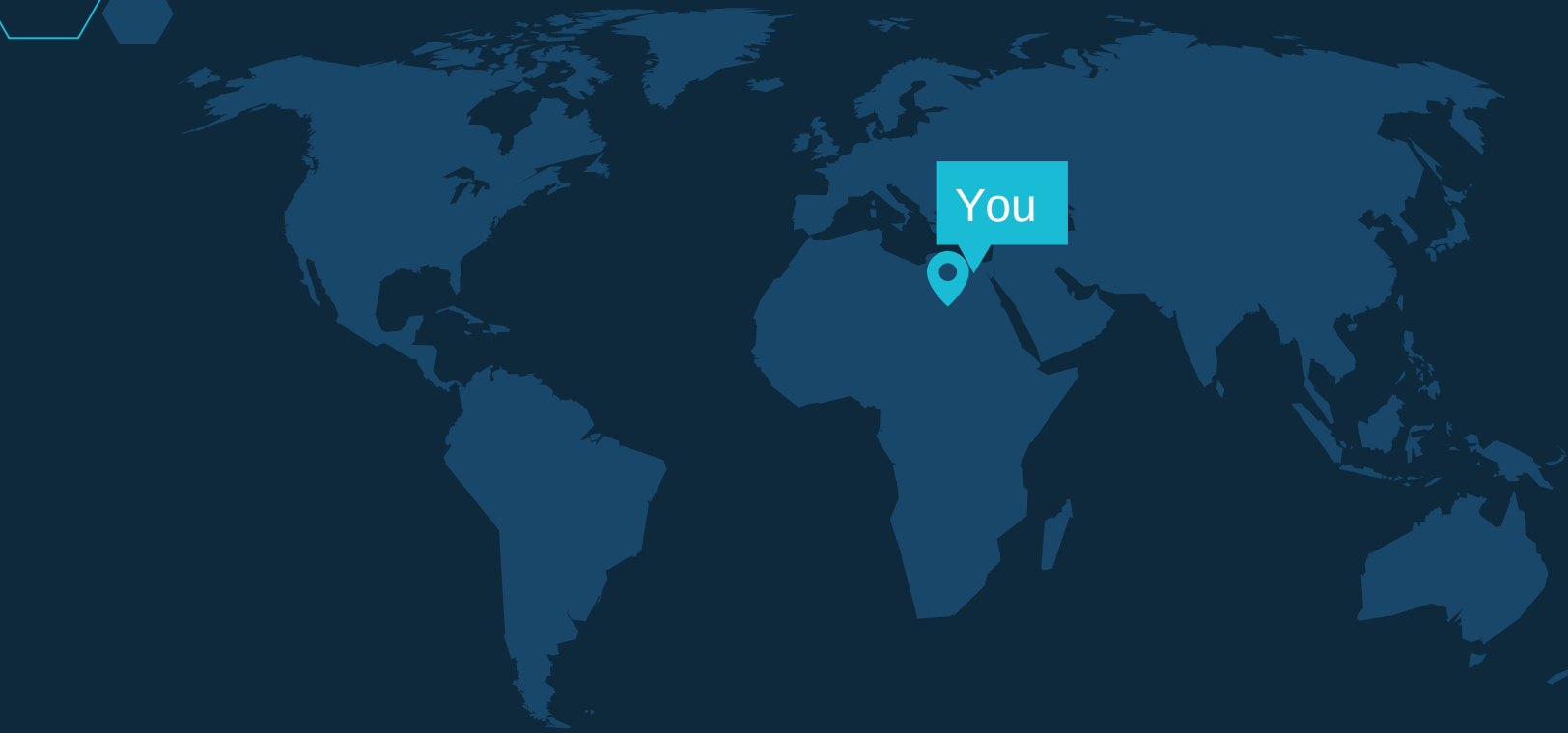
Thanks!

Any questions?

You can find me at:

- itiroutealexiti.com

Be sure that you are making a difference .




Break

Week 8

(Abstraction-Pointers-Polymorphism)

Agenda

- What is Abstraction ?
- Template
- Pointers
- Declaring and Initializing Pointers
- Size of Pointers
- Storing Address in Pointer
- Accessing Data that the pointer points to
- Pointers and Arrays
- Passing Pointers to Functions
- Objects as Pointers
- Memory Types
- Heap Memory
- What is Polymorphism ?
- Exception Handling

1

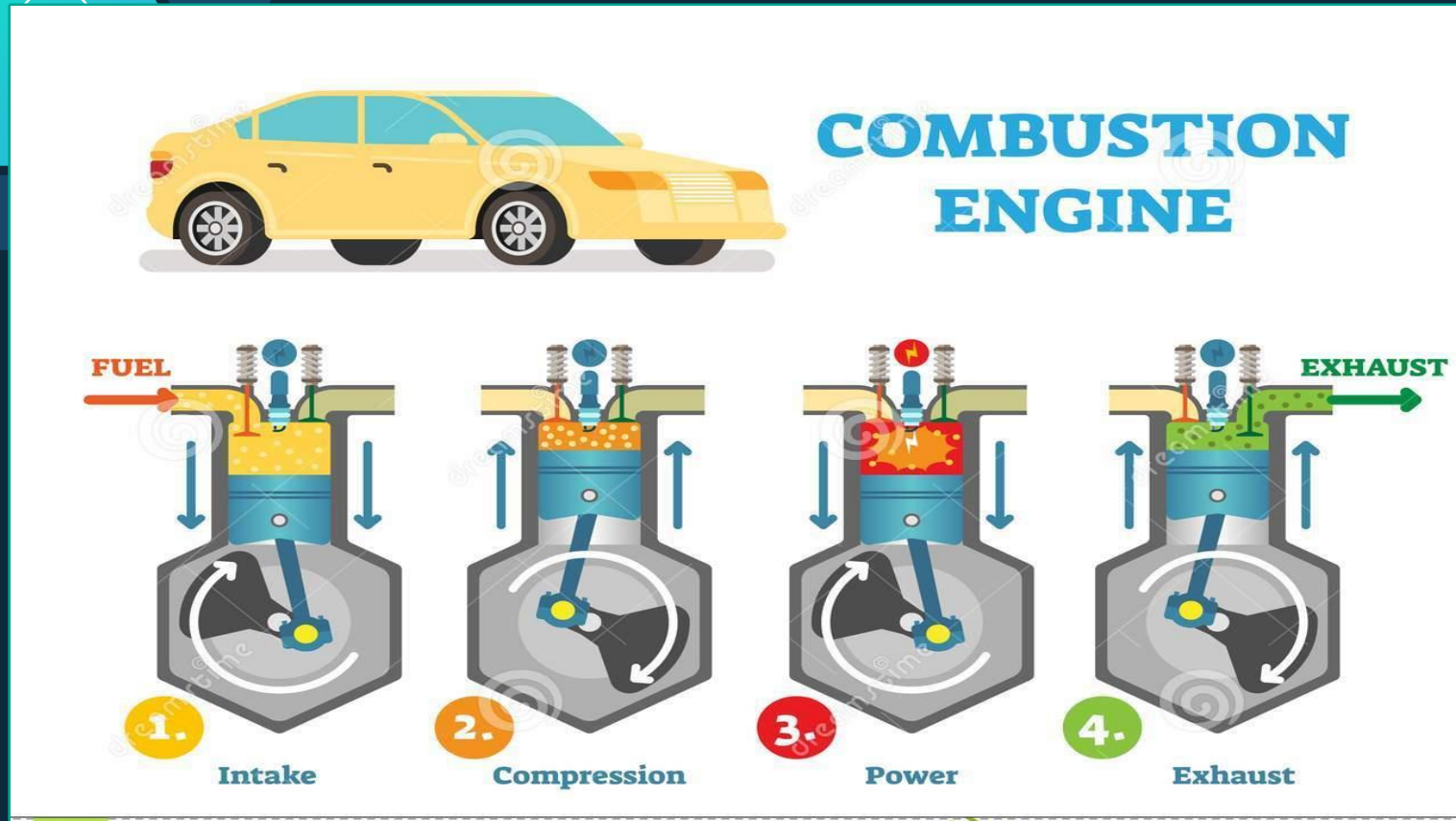
What is Abstraction ?

???

What is Abstraction ?

- Third OOP Concept, It is an Interface describes the behavior of the class **without** any implementation of that class.
- It is declared by declaring at least **one pure** virtual function.
- Children must **override** on abstracted functions, and if they didn't override them, children will be abstracted classes.
- **Can not** take any objects from abstracted class.

What is Abstraction ?



What is Abstraction ?

```cpp
class Parent {
protected:
    string lname;
public:
    //abstracted function
    virtual void showName() = 0;
};

class Child : public Parent {
private:
    string fname;
public:
    void setNames(string f,string l){
        lname = l;
        fname = f;
    }
    //must declare the abstracted function
    void showName() {
        cout << fname << " " << lname;
    }
};
int main()
{
    Child c;
    c.setNames("Ibrahim", "Yasser");
    c.showName();
}
```

```
Microsoft Visual
Ibrahim Yasser
E:\Courses\1. Pr
cess 15384) exit
Press any key to
```

Ex 1:

- Create Abstracted Shape class with pure virtual function calcArea = 0;
- Then create three classes
  (Rectangle, Triangle, Circle)
  And override the calcArea function

Quiz 1:

- Create Abstracted Animal class with pure virtual function showType = 0;
- Then create three classes
  (Dolphin, Cat, Dog)
  And override the showType function to print the type

2

Template

???

What is Template?

- **Template :** A temporary data type defined at certain time.

```
template <class t, class t1>
class MyPair
{
private:
    t key;
    t1 value;
public:
    void setKey(t n) {
        key = n;
    }
    void setValue(t1 x) {
        value = x;
    }
    t getKey() {
        return key;
    }
    t1 getvalue() {
        return value;
    }
};

int main() {
    MyPair<char,int> p;
    p.setKey('a');
    p.setValue(5);
    cout << p.getKey() << endl;
    cout << p.getvalue() << endl;

}
```

```
a
5

E:\Co
cess
Press
```

```
template <class t, class t1>
class MyPair
{
private:
    t key;
    t1 value;
public:
    void setKey(t n) {
        key = n;
    }
    void setValue(t1 x) {
        value = x;
    }
    t getKey() {
        return key;
    }
    t1 getvalue() {
        return value;
    }
};

int main() {
    MyPair<double,string> p;
    p.setKey(10.5);
    p.setValue("ahmed");
    cout << p.getKey() << endl;
    cout << p.getvalue() << endl;

}
```

```
10.5
ahmed

E:\Cour
cess 50
Press a
```

3 Pointers

???

What are Pointers ?

- **A variable:** whose value is an address
- What can be at that address ?
  - Another variable
  - A function
- So pointers **point** to variable or function , if X is an integer and it's value is 10 then you can declare a pointer that points to it.
- Pointers are used to allocate memory on **heap** , this memory doesn't even have names for variables, so the only way to access this data is via **Pointers.**

4 Declaring and Initializing Pointers

How to declare and initialize ?

Declaring and Initializing Pointers

```
Variable_type* Pointer_name;

int* x;
double* y;
char* z;
string* s;

Declaring
```

```
Variable_type* Pointer_name {nullptr};

int* x    { nullptr };
double* y { nullptr };
char* z   { nullptr };
string* s { nullptr };

Intializing
```

- Initialize pointer to 'point nowhere' using **nullptr**

Accessing Pointers Address

```cpp
int main() {

    int num = 10;
    cout << "Value of num = " << num << endl;
    cout << "Size of num = " << sizeof num << endl;
    cout << "Address of num = " << & num << endl;

}
```

```
Microsoft Visual Studio Debug Cons
Value of num = 10
Size of num = 4
Address of num = 00BFFD5C

E:\Courses\1. Programming\1. I
cess 10628) exited with code 0.
Press any key to close this wi
```

```cpp
int main() {

    int num = 10;
    int *p = &num;
    cout << "Address of num = " << &num << endl;
    cout << "Value of p = " << p << endl; //num address
    cout << "Size of p = " << sizeof p << endl;
    cout << "Address of p = " << & p << endl;
    p = nullptr; //points nowhere
    cout << "Value of p = " << p << endl; //garbage

}
```

```
Microsoft Visual Studio Debug
Address of num = 002AFCFC
Value of p = 002AFCFC
Size of p = 4
Address of p = 002AFCF0
Value of p = 00000000

E:\Courses\1. Programming\1
cess 7096) exited with code
Press any key to close this
```

5

Size of Pointers

What is the size of what pointers have ?

Size of Pointer

```cpp
int main() {

    int* p1{ nullptr };
    cout << "Size of p1 = " << sizeof p1 << endl;

    double* p2{ nullptr };
    cout << "Size of p2 = " << sizeof p2 << endl;

    unsigned long long* p3{ nullptr };
    cout << "Size of p3 = " << sizeof p3 << endl;

    string* p4{ nullptr };
    cout << "Size of p4 = " << sizeof p4 << endl;

    vector<string>* p5{ nullptr };
    cout << "Size of p5 = " << sizeof p5 << endl;

}
```

```
Microsoft Visual S
Size of p1 = 4
Size of p2 = 4
Size of p3 = 4
Size of p4 = 4
Size of p5 = 4

E:\Courses\1. Pro
cess 4184) exited
Press any key to
```

- Don't confuse the size of pointer, and the size of what it points to
- All pointers in the system have **the same size**, they may points to large or small types.

6

Storing Address in Pointer

How to store ?

Storing Address in Pointer

- The type of pointer must be **the same** type of what it points to
- The compiler will make sure the address stored is the same type of pointer type

```
int main() {
    int x = 10;
    double y = 3.7;

    int* p1{ nullptr };
    p1 = &x;            //ok
    p1 = &y;            //Error
}
```

1 Error | 0 Warnings | 0 Messages | Build + IntelliSense

Description	Project	File
a value of type "double *" cannot be assigned to an entity of type "int *"	ITI-Week8	ITI-Week8.cpp

Storing Address in Pointer

- Pointers are variables, so they can change
- Pointers can be null

```
int main() {
    int x = 15;
    int y = 7;

    int* p{ nullptr };
    p = &x;      //    p points to x
    p = &y;     // Now p points to y
}
```

7

Accessing Data that the pointer points to

How to access ?

Accessing Data that the pointer points to

```
int main() {
    int x = 100;
    int y = -5;

    int *p = &x;

    cout << *p << endl << endl;//100
    *p = 200;
    cout << *p << endl;          //200
    cout << x << endl << endl; //200

    p = &y;//Now p points to y
    cout << *p << endl << endl;//-5
    *p *= 5;
    cout << *p << endl;          //-25
    cout << y << endl;           //-25
}
```

```
Microso
100

200
200

-5

-25
-25

E:\Courses
cess 1332)
Press any
```

- You can access the data that the pointer points to using * operator

8

Pointers and Arrays

How to deal ?

Pointers and Arrays

- The value of an array is the address of the first element in the array
- The value of a pointer is an address
- When the pointer points to an array it's value equal the address of first element

```cpp
int main() {
    int arr[]{ 10,20,30 };
    cout << arr << endl;  // address of first element
    cout << *arr << endl; // value of first element

    int* p{ arr };
    cout << p << endl;  // address of first element
    cout << *p << endl; // value of first element
//  To print the array through the pointer:
    cout << "===== Array Values =====\n";
    cout << p[0] << endl;
    cout << p[1] << endl;
    cout << p[2] << endl;
    cout << "===== Array Addresses =====\n";
    cout << p << endl;
    cout << (p+1) << endl;
    cout << (p+2) << endl;
}
```

```
Microsoft Visual Studio Debug Co
008FFA2C
10
008FFA2C
10
===== Array Values =====
10
20
30
===== Array Addresses =====
008FFA2C
008FFA30
008FFA34

E:\Courses\1. Programming\1.
cess 7192) exited with code 0
Press any key to close this w
```

9

Passing Pointers to Functions

How to pass ?

Passing Pointers to Functions

- Pass by reference with pointer parameters
- The function parameter can be a pointer or address of a variable

```
void double_salary(int* s) {
    *s *= 2;
}

int main() {
    int salary = 1000;
    cout << salary << endl;
    double_salary(&salary);
    cout << salary << endl;

}
```

```
Micro
1000
2000

E:\Cours
cess 862
Press an
```

10

Objects as Pointers

How to access ?

Objects as Pointers

Note 1:

- Any class has an internal pointer called this, created automatically

Note 2:

- To access data of object pointer use " -> "Operator instead of " . " Operator

```
class Person {
private:
    string name;
    int age;
public:
    Person(){}
    Person(string name, int age) {
        this->name = name;
        this->age = age;
    }
    void setName(string name) {
        this->name = name;
    }
    void setAge(int age) {
        this->age = age;
    }
    string getName() {
        return name;
    }
    int getAge() {
        return age;
    }
};

int main() {
    Person* p = new Person;
    p->setName("Ahmed");
    p->setAge(30);
    cout << p->getName() << endl;
    cout << p->getAge() << endl;
}
```

```
Ahmed
30

E:\Cour
cess 11
Press a
```

11

Memory Types

How many types ?

Memory Types

Stack	Heap
A **static** memory allocated at **compile** time	A **dynamic** memory allocated at **run** time
Variables are stored **continuously** in memory at **compile** time	Variables are stored **randomly** in memory at **run** time
Access to memory is **faster**, as variables are already arranged **automatically**	Access to memory is **slower**, as variables are not arranged and must be accessed **manually** through **pointers**
Size is **fixed** and **limited**	Size is **dynamic** and **larger**, (need to **delete** unused variables)

12

Heap Memory

How to allocate heap ?

Heap Memory

- We use pointer to allocate new memory at heap at run time, using **new** keyword

```
int main() {

    int* p{ nullptr };
    p = new int;         // allocate an int in heap
    cout << p << endl; // address of new int
    cout << *p << endl;// value of new int(garbage)
    *p = 10;
    cout << *p << endl;// value of new int(10)

}
```

```
Microsoft
00628A40
-842150451
10

E:\Courses\1
cess 3864)
Press any k
```

Heap Memory

- We deallocate variable storage, using **delete** keyword

```
int main() {
    int* p{ nullptr };
    p = new int;         // allocate an int in heap

    // code ...

    delete p;            // Frees the allocated storage
}
```

Heap Memory

- We use **new[ ]** to allocate storage for array in heap
- And we use **[ ]delete** to delete array from heap

```
int main() {
    int* p { nullptr };
    int size = 0;

    cout << "Enter Array size\n";
    cin >> size;

    p = new int[size]; // allocate array in heap

    // code ...

    delete [] p;       // Frees the allocated storage
}
```

13

What is Polymorphism ?

???

What is Polymorphism ?

- The fourth OOP concept that refers to the ability of object to take multiple forms, so objects with common parent may have the same name of function but with different behaviors at run time .

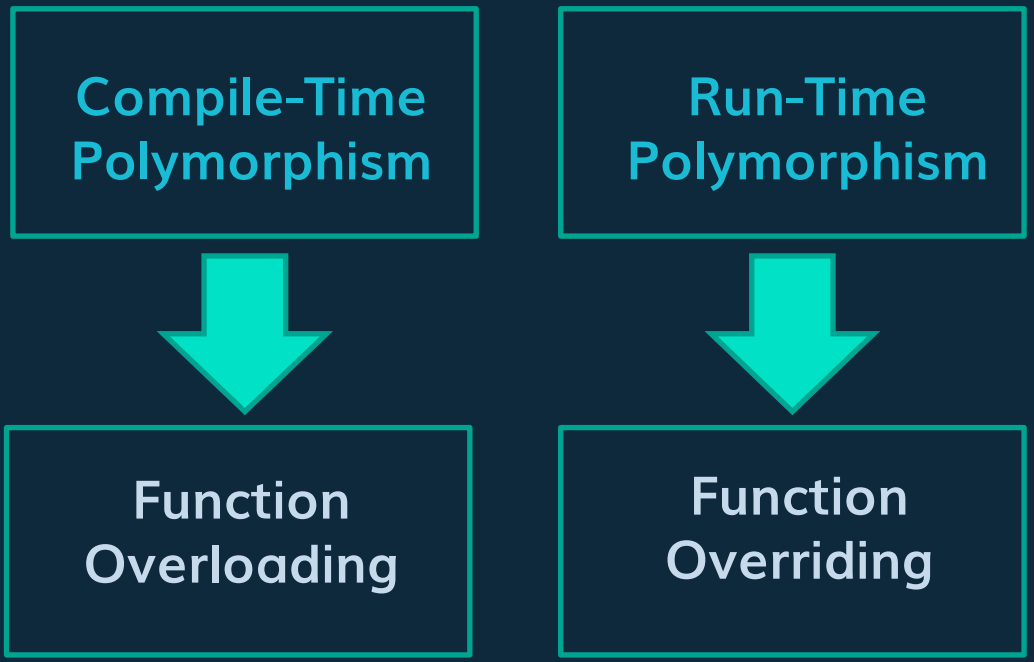
Polymorphism

Example 2

Polymorphism

Quiz 2

What is Polymorphism ?



Difference between overloading and overriding

- Overloading:
  - A rewritten function with the same name of another function but with different signature (Number of parameters, Type of parameters) and its processing happened in compile time (Compile Time Polymorphism).
- Overriding:
  - A rewritten function in child class with the same name of another function in parent class and with the same signature (Number of parameters, Type of parameters) and its processing happened in run time (Run Time Polymorphism).

14

Exception Handling

To deal with errors

Exception Handling

- What happens if y = 0 ?
  - Crash ?
  - It depends!

```
int divide(int x, int y) {
    return x / y;
}

int main() {
    cout << divide(5, 0) << endl;
}
```

Exception Handling

- What happens if y = 0 ?
  - Crash ?
  - It depends!

```
int divide(int x, int y) {
    if (y == 0) {
        // what to do ?
    }
    else {
        return x / y;
    }
}

int main() {

    cout << divide(5, 0) << endl;
}
```

Exception Handling

- Exception:
  - An object or primitive type where error occurred

- Try Block:
  - The block that contain the code that would cause error

- Catch Block:
  - The block that handle the error

Exception Handling

- C++ standard library has exception class you can use it to handle errors , #include < exception >

- And you can inherit from it many other exception classes to create your own exception class by editing what() function, and through polymorphism there will be only one exception class called when it's error occurred

```
const char* what() const throw()
{
    return "Can not divide by zero";
}
```

Exception Handling

Example 3

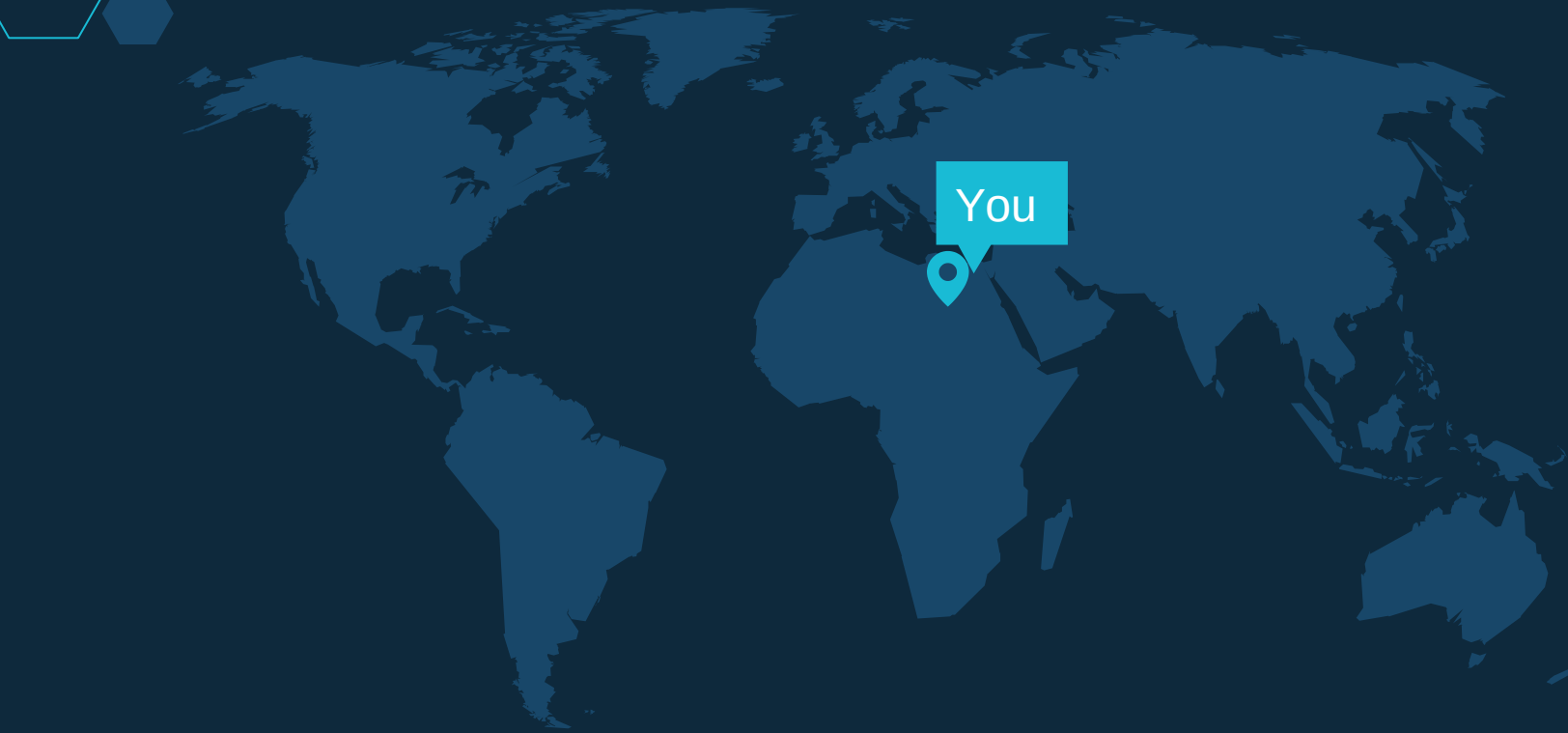
Thanks!

Any questions?

You can find me at:

- itiroutealexiti.com

Be sure that you are making a difference .




Break

Week 9

(OOP Revision – Part 1)

Agenda

- Why OOP ?
- What is OOP ?
- Classes
- Objects
- Declaring Classes
- Accessing Class Members
- Encapsulation
- Setters and Getters
- Constructors
- Destructors

1

Why OOP ?

???

Why OOP ?

If you want to store student name :

```
string StudentName;
cin >> StudentName;
```

Why OOP ?

Now , if you need to store his name , id , age , and his marks ?

```
string StudentName;
int id,age;
float arabic, math, english;

cin >> StudentName;
cin >> id >> age;
cin >> arabic, math, english;
```

Why OOP ?

Now , what if you want to store all class students data ?

```
string StudentName2;
int id2, age2;
float arabic2, math2, english2;

cin >> StudentName2;
cin >> id2 >> age2;
cin >> arabic2, math2, english2;

/*
to the last class student
.
.
.
*/
string StudentName40;
int id40, age40;
float arabic40, math40, english40;

cin >> StudentName40;
cin >> id40 >> age40;
cin >> arabic40, math40, english40;
```

Why OOP ?

- Now , store all school classes students ?

A very long code , with many many variables , right ?

Why OOP ?

- To solve that we need to organize our code by making a student class and reuse the **class** code by making many **objects** :

  (**Student s1, s2, s3 , …, s1000;**)

- This method **OOP** is depend on **Reality Simulation** concept .

2

What is OOP ?

???

What is OOP ?

- **OOP : Object Oriented Programming**
  A **Style** of programming based on object concept, it **doesn't** refer to a specific language but it is a way to build a program by making **Simulation of Reality**

What is OOP ?

- **Classes and Objects :**

```
class Student
{
    string StudentName;
    int id, age;
    float arabic, math, english;
};

//the objects :
Student s1, s2, s3, s1000;
```

3 Classes

What is the class ?

Classes

- **Blueprint** from which Objects are created
- A user defined **Data-type**
- The class has :
  1. **Attributes** (his data) -variables-
  2. **Methods** (his behavior) –functions-
  3. **Constructors** (a special method member)
  4. **Destructors** (a special method member)
- Can **hide** data and methods
- Provide public interface
- Examples :
  - Account
  - Student
  - Image

4 Objects

What is the Object ?

Objects

- **Created** from a class
- Represents a specific **instance** of a class
- Can create **many many** objects from one class
- The objects has :
  - It's own **identity**

    (It's own data, methods, Constructors, Destructor)
- Each can use the defined class methods
- Examples :
  - **Ahmed's bank account** is an instance of an Account
  - **Sara's bank account** is an instance of an Account
    - And each has it's own balance , can make deposits, withdrawals, ...

5

Declaring Class

How to define a class ?

Declaring Class

```
class Class_name
{
    // declaration(s);
};
```

- Note : it's known between developers that the first letter of class is a **Capital** letter, so that we can different between classes (capital letter), and functions (all letters are small)

6

Accessing Class members

How to access a class ?

Accessing Class members

- We use dot operator (.) to access public members at main function :

```
class Student
{
public:
    string StudentName;
    int id, age;
    float arabic, math, english;
};

int main()
{
//the objects :
Student s1, s2, s3, s1000;

s2.StudentName = "Abd elrahman";
s2.math = 50;
s1.
}
```

```
age
arabic
english
id
math
StudentName
```

(field) public : int Student::age
File: ITI-Week6.cpp

Ex 1 :

- Create account class with :
  - Attributes :
    - Name
    - Id
    - balance
  - One Method :
    - Printing info
- And then create objects in main function

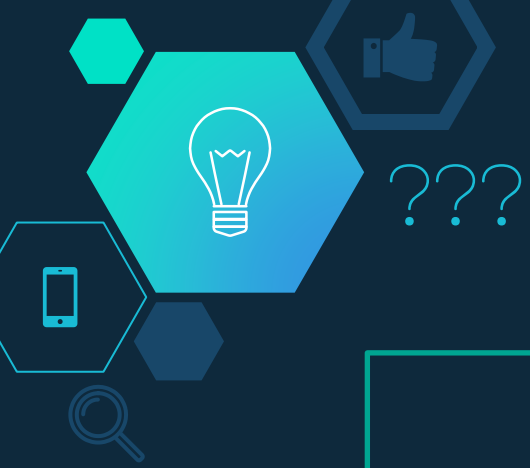
7

Encapsulation

To hide data

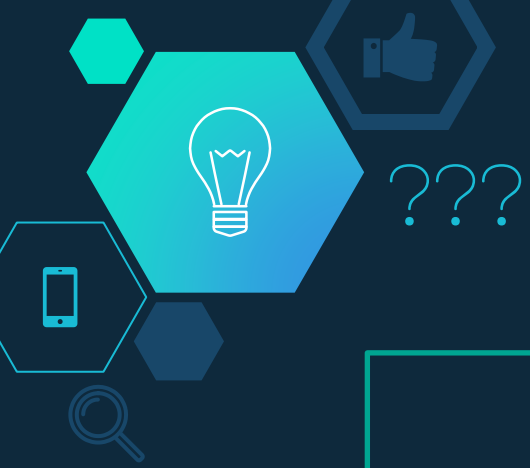
Encapsulation

- **Encapsulation :** the first OOP concepts, it represents the ability of OOP to **hide** or **public** objects data
- **Access Modifiers :**
  - **Public** : public members are accessible **everywhere**
  - **Private** : private members are only accessible by **class members**
  - **Protected** : used with **inheritance** (next session)
- Usually we make all attributes **private** members to secure them from any external access (private balance;)
- And we make **public** methods for each attribute to be accessed by certain users (public show_balance( ) { cout<<balance; })



What is The default Access Modifiers in :

- Class ?
- Struct ?



What is The default Access Modifiers in :

- Class : Private
- Struct : Public

8

Setters and Getters

To hide data

Setters

- A method that we create for each attribute to control it's **enter** access

```
class Account
{
private:
    float balance;

public:
    void set_balance(float b) {
        balance = b;
    }
};

int main()
{
    Account osama;
    osama.set_balance(1000);
}
```

Note that :

- Setters functions are always has no return type (**void**) , and always **has parameters** , all what it do is to set value of it's attribute

Getters

- A method that we create for each attribute to control it's **show** access

```
class Account
{
private:
    float balance;

public:
    void set_balance(float b) {
        balance = b;
    }
    float get_balance() {
        return balance;
    }
};

int main()
{
    Account osama;
    osama.set_balance(1000);
    cout << osama.get_balance();
}
```

Note that :

- Getters functions are always has a return type is the **same** type of it's getting attribute and **has no parameters** and all what it do is to return it

Ex 2 :

Create an account class with :

- attributes :

  Name, id, balance

- methods :

  setters and getters for each attribute and a method to print all info

9 Constructors

At creation

Constructors

- A special member method :
  - Has **no return type**
  - His name is **same** as class name
  - Self **invoked automatically** during object creation
- Default Constructor is useful for **initialization**
- Constructor can be **overloaded** and we can use Parameterized constructor to send data at creation
- If you don't create any constructor, **C++ will create** the default constructor for you, but it will be empty without any code or logic
- But if you create a parameterized constructor without creating default constructor, **C++ will not create** create it and you will be asked to send data at each time as you will use the parameterized constructor
- So don't create any parameterized constructor **before** creating the default one

Constructors

```
class Account
{
private:
    string name;
    int id;
    float balance;
public:
    //Default constructor
    Account() {
        name = " ";
        id = 0;
        balance = 0;
    }
    //Parameterized constructor
    Account(string n, int i, float b) {
        name = n;
        id = i;
        balance = b;
    }
}
```

```
int main(){

Account a1; //default cons is called
a1.print_info();

Account a2("amira",3,20000); //Para cons is called
a2.print_info();
}
```

```
Microsoft Visual Studio Debug Console
Customer Name:
id  : 0
Balance = 0
================================
Customer Name: amira
id  : 3
Balance = 20000
================================
```

10

Destructors

At Destroying

Destructors

- A special member method :
  - Has **no return type**
  - His name is **same** as class name with (**~**) sign
  - Self **invoked automatically** when object is destroyed

- Default destructor is useful for **releasing memory**

- Destructor can not be **overloaded** only one destructor at the class

- If you don't create destructor, **C++ will create** it for you, but it will be empty without any code or logic

Destructors

```
class Account
{
private:
    string name;
    int id;
    float balance;
public:
    //Default constructor
    Account() {
        name = " ";
        id = 0;
        balance = 0;
    }
    //Parameterized constructor
    Account(string n, int i, float b) {
        name = n;
        id = i;
        balance = b;
    }
    //Destructor
    ~Account() {

    }
}
```

Ex 3 :

Create an account class with :

- Default constructor
- Parameterized constructor
- Destructor and print inside it “Bye Bye”

◇ attributes :

Name, id, balance

◇ methods :

setters and getters for each attribute and a method to print all info

Adding class in IDE

- Visual Studio :
  Right click on source file , choose add class

- Code blocks or dev :

```
#ifndef class_name
#define class_name
class class_name
{
//code
};
#endif
```

- In main function or in cpp file:
  #include “class_name.h”

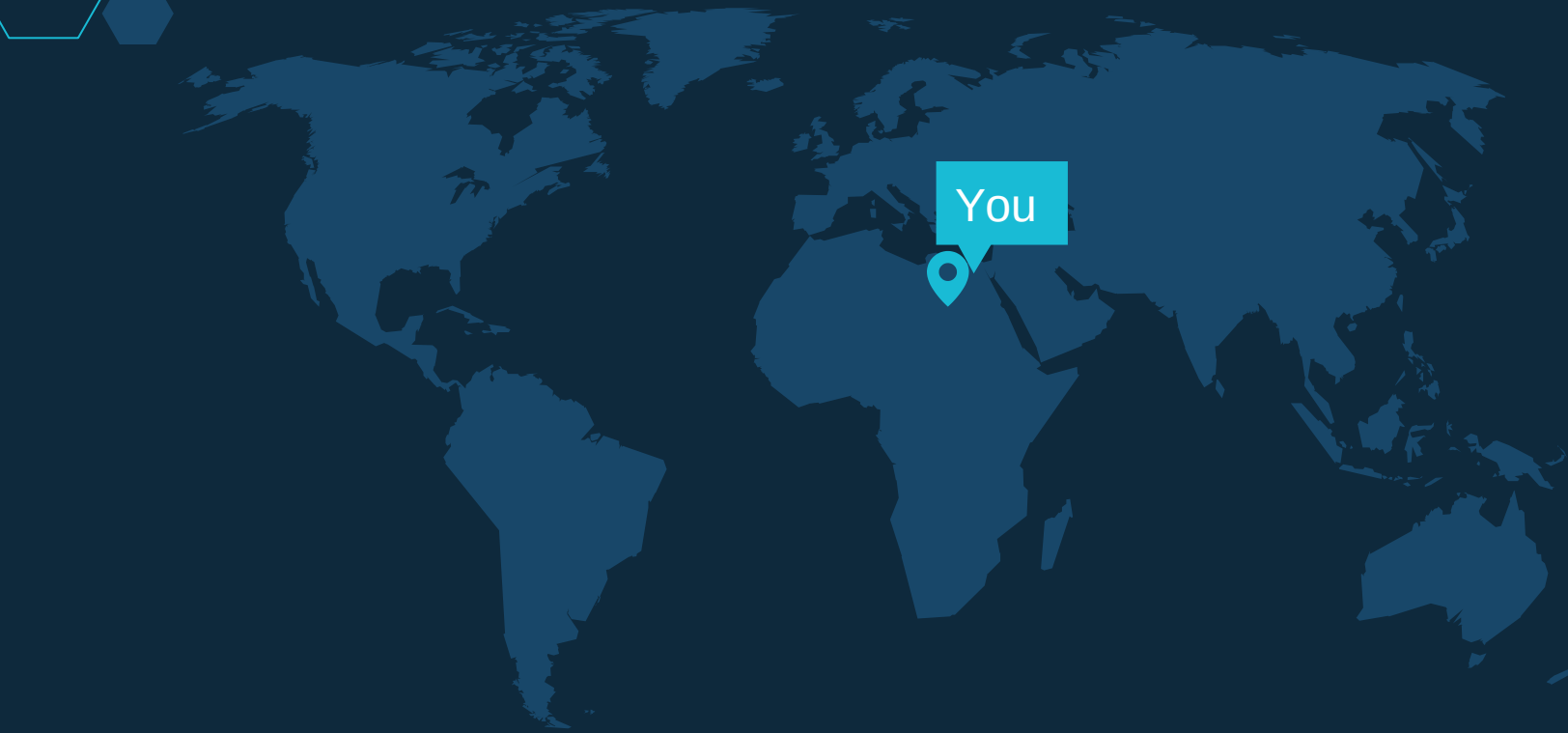
Thanks!

Any questions?

You can find me at:

- itiroutealexiti.com

Be sure that you are making a difference .




Break

Week 9

(OOP Revision – Part 2)

Agenda

- Why Inheritance ?
- What is Inheritance ?
- Deriving class from existing classes
- Protected members
- Constructors in Inheritance
- Passing Arguments to Parameterized Constructors
- Destructors in Inheritance
- Methods Overriding
- Inheritance Types
- Inheritance Vs Composition
- Static Members
- Static Methods

1

Why Inheritance ?

???

Why Inheritance ?

If you have a person class, and you want to add student and teacher class, what will be your code ?

```
class Person {
private:
    string name;
    int id;
public:
    void set_name(string n) {
        name = n;
    }
    void set_id(int i) {
        id = i;
    }
    string get_name() {
        return name;
    }
    int get_id() {
        return id;
    }
};
```

```
class Student {
private:
    string name;
    int id;
    float total_marks;
public:
    void set_name(string n) {
        name = n;
    }
    void set_id(int i) {
        id = i;
    }
    void set_total_marks(int t) {
        total_marks = t;
    }
    string get_name() {
        return name;
    }
    int get_id() {
        return id;
    }
    float get_total_marks() {
        return total_marks;
    }
};
```

```
class Teacher {
private:
    string name;
    int id;
    double salary;
public:
    void set_name(string n) {
        name = n;
    }
    void set_id(int i) {
        id = i;
    }
    void set_salary(double s) {
        salary = s;
    }
    string get_name() {
        return name;
    }
    int get_id() {
        return id;
    }
    double get_salary() {
        return salary;
    }
};
```

Why Inheritance ?

- So we **duplicate** the same data of person into student and teacher classes, this is against **DRY** concept.
- So we use **inheritance** to create **new** classes from **existing** classes
- The new class (**child class**) have the same data and methods of the existing class (**parent class**)
- Allow child class to **modify** parent methods (Overriding) **without modify** original class

```
class Person {
protected:
    string name;
    int id;
public:
    void set_name(string n) {
        name = n;
    }
    void set_id(int i) {
        id = i;
    }
    string get_name() {
        return name;
    }
    int get_id() {
        return id;
    }
};
```

```
class Student:public Person {
private:
    float total_marks;
public:
    void set_total_marks(int t) {
        total_marks = t;
    }
    float get_total_marks() {
        return total_marks;
    }
};
```

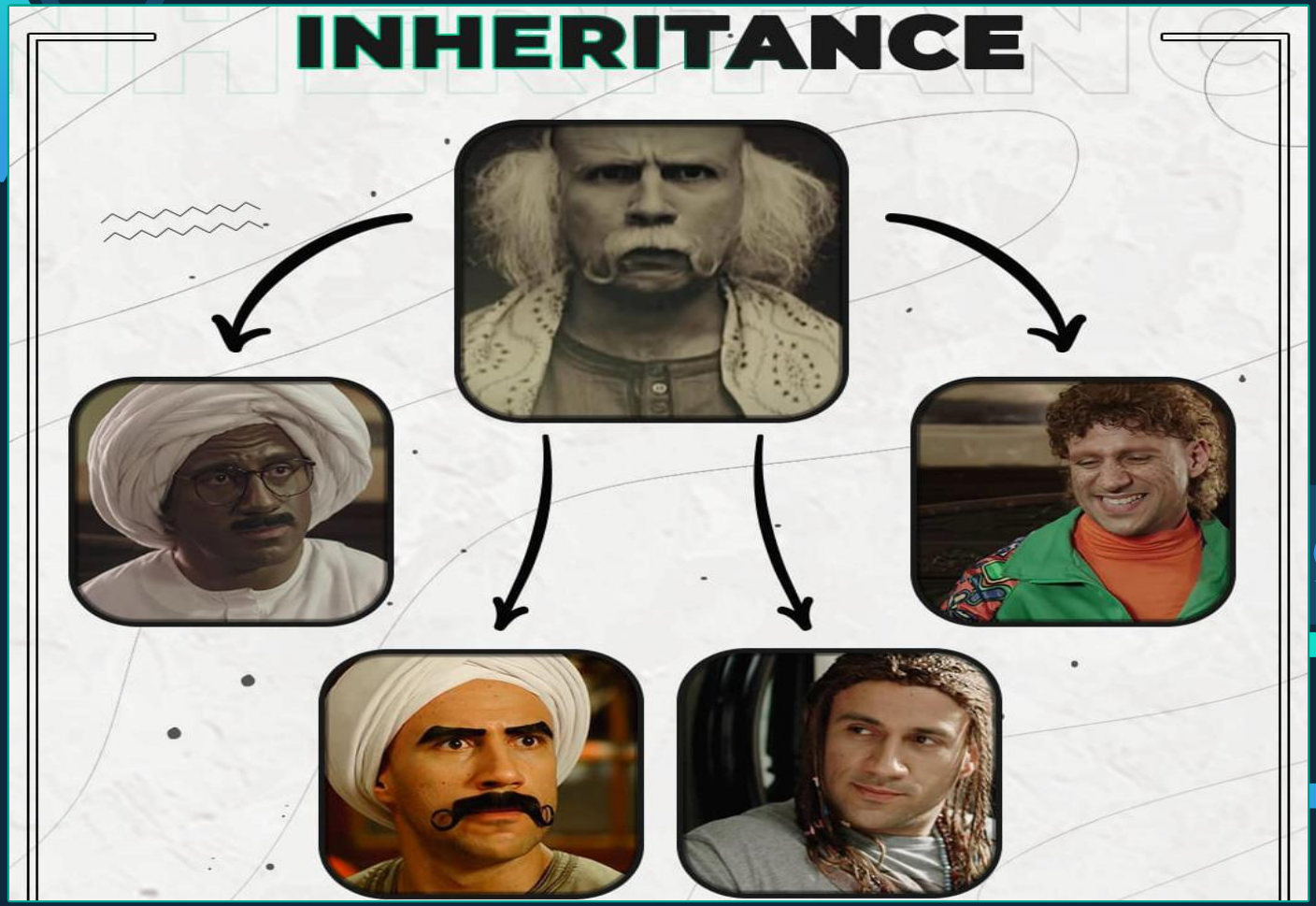
```
class Teacher :public Person {
private:
    double salary;
public:
    void set_salary(double s) {
        salary = s;
    }
    double get_salary() {
        return salary;
    }
};
```

2

What is Inheritance ?

???

What is Inheritance?



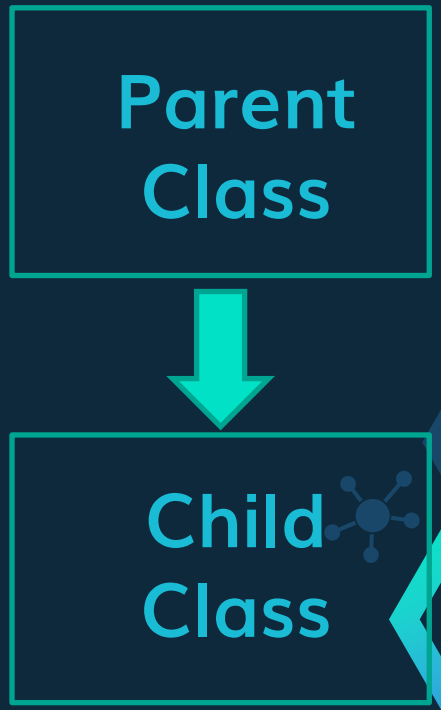
What is Inheritance?

- **Inheritance :** the **Second** OOP concepts that allows us to create **new** classes from **existing** classes, the child class will be able to inherit all parent data and methods and could **modify** the inherited parent's methods

What is Inheritance?

- **Parent Class (Base or Super Class) :** the class being inherited or extended from

- **Child Class (Derived or Sub Class) :** the class being created from parent class which will inherit all attributes and methods



3

Deriving class from existing classes

How to inherit ?

Deriving class from existing classes

```
class Parent{
    //Code
};
class Child : access_specifier Parent{
    //Code
};
```

- **Access specifier** can be : public, private, protected
- **Public** is the most common access modifier

Deriving class from existing classes

Parent Class

Public : a
Protected : b
Private : c

Public Inheritance

Child Class

Public : a
Protected : b
c : No Access

Deriving class from existing classes

Parent Class

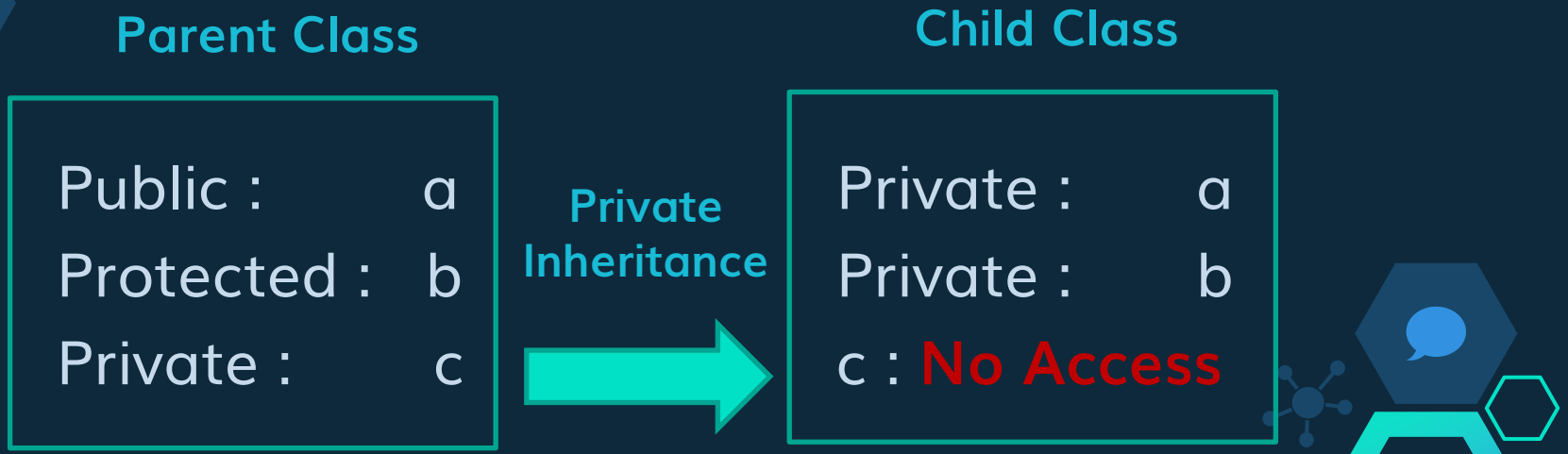
Public : a
Protected : b
Private : c

Protected Inheritance

Child Class

Protected : a
Protected : b
c : **No Access**

Deriving class from existing classes



4

Protected members

Only child can access

Protected members

```
class Parent{

protected:
    string title;
    int id;
};
```

- Protected Members Are accessible only by:
  - Parent class members
  - Child class members

Ex 1:

Create a Person class that includes :

- Attributes :
  - String name, int id
- Methods :
  - Setters, Getters

Then create Teacher class which inherit from Person

All data and methods and has :

- Attributes :
  - Float Salary
- Methods :
  - Setters, Getters
  - Get annual salary (12*salary)
  - Get raise (0.07*salary)

Then create Student class which inherit from Person

All data and methods and has :

- Attributes :
  - Float total marks
- Methods :
  - Setters, Getters

5

Constructors in Inheritance

Which one is called firstly, Parent or Child Constructor ?

Constructors in Inheritance

```
class Parent{

public:
    Parent() {
        cout << "Parent Constructor" << endl;
    }
};

class Child : public Parent{

public:
    Child() {
        cout << "Child Constructor" << endl;
    }
};


int main()
{
    Child c;
}
```

```
Microsoft Visual Studio D
Parent Constructor
Child Constructor

E:\Courses\1. Programm
ce.exe (process 12684)
Press any key to close
```

The parent constructor is executed firstly then the child constructor executed

Passing Arguments to Parameterized Constructors in Inheritance

How to pass ?

Passing Arguments to Parameterized Constructors in Inheritance

```
class Parent{
private:
    string name;
    int age;
public:
    Parent();
    Parent(string n,int a) {
        name = n;
        age = a;
    }
};

class Child : public Parent{
private:
    double expenses;
public:
    Child();
    Child(string n,int a,double e):Parent(n,a) {
        expenses = e;
    }
};
```

7

Destructors in Inheritance

Which one is called firstly, Parent or Child Destructor ?

Destructors in Inheritance

```cpp
class Parent{

public:
    ~Parent() {
        cout << "Parent Destructor" << endl;
    }
};

class Child : public Parent{

public:
    ~Child() {
        cout << "Child Destructor" << endl;
    }
};


int main()
{
    Child c;
}
```

```
Microsoft Visual Stu
Child Destructor
Parent Destructor

E:\Courses\1. Progr
ce.exe (process 155
Press any key to cl
```

The child destructor is executed firstly then the Parent destructor executed

8

Methods Overriding

Child can override on parent methods

Methods Overriding

```
class Parent{

public:
    void hello() {
        cout << "Hi i am Parent" << endl;
    }
};

class Child : public Parent{

public:
    void hello() {
        cout << "Hi i am Child" << endl;
    }
};


int main()
{
    Parent p;
    Child c;

    p.hello();
    c.hello();
}
```

```
Microsoft Visual St
Hi i am Parent
Hi i am Child

E:\Courses\1. Prog
ce.exe (process 81
Press any key to c
```

Ex 2:

Create a Person class that includes :

- **Constructors, Destructor**
- Attributes :
  - String name, int id
- Methods :
  - Setters, Getters

Then create Employee class which inherit from Person

All data and methods and has :

- **Constructors, Destructor**
- Attributes :
  - Float Salary
- Methods :
  - Setters, Getters
  - Get annual salary (12*salary)
  - Get raise (0.07*salary)

Then create Student class which inherit from Person

All data and methods and has :

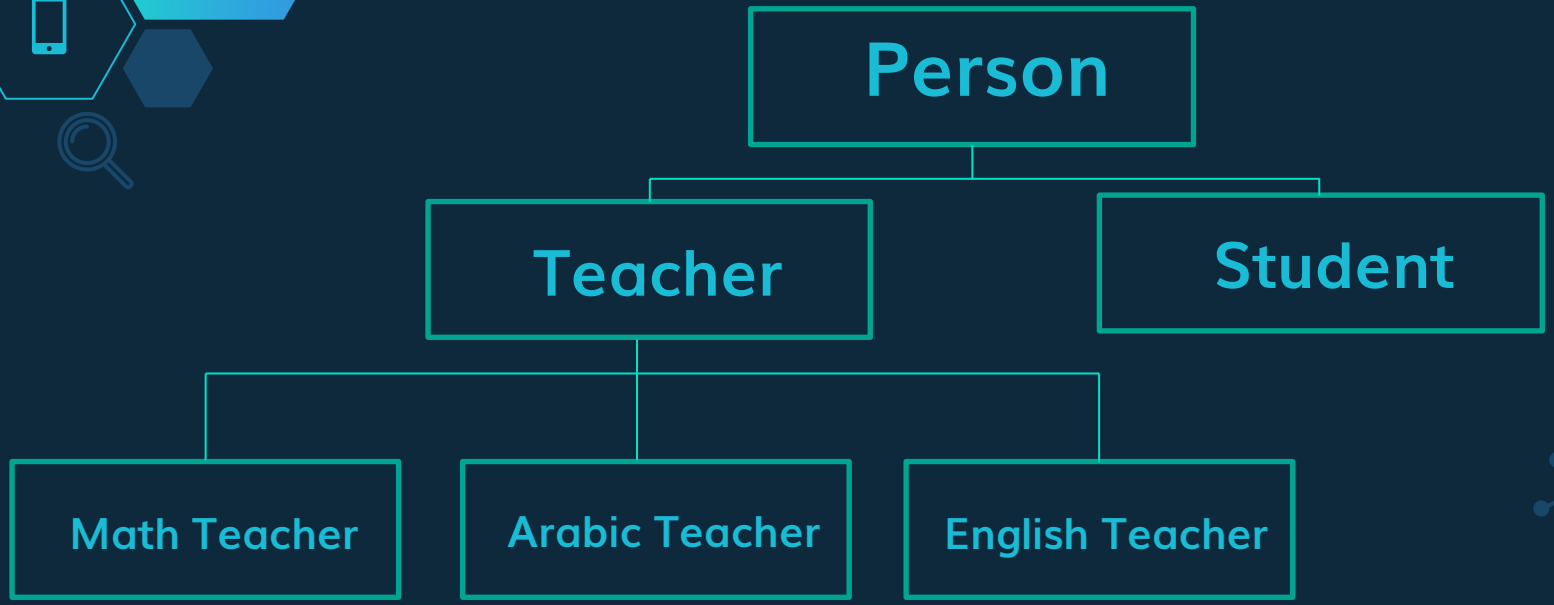
- **Constructors, Destructor**
- Attributes :
  - Float total marks
- Methods :
  - Setters, Getters

9

Inheritance Types

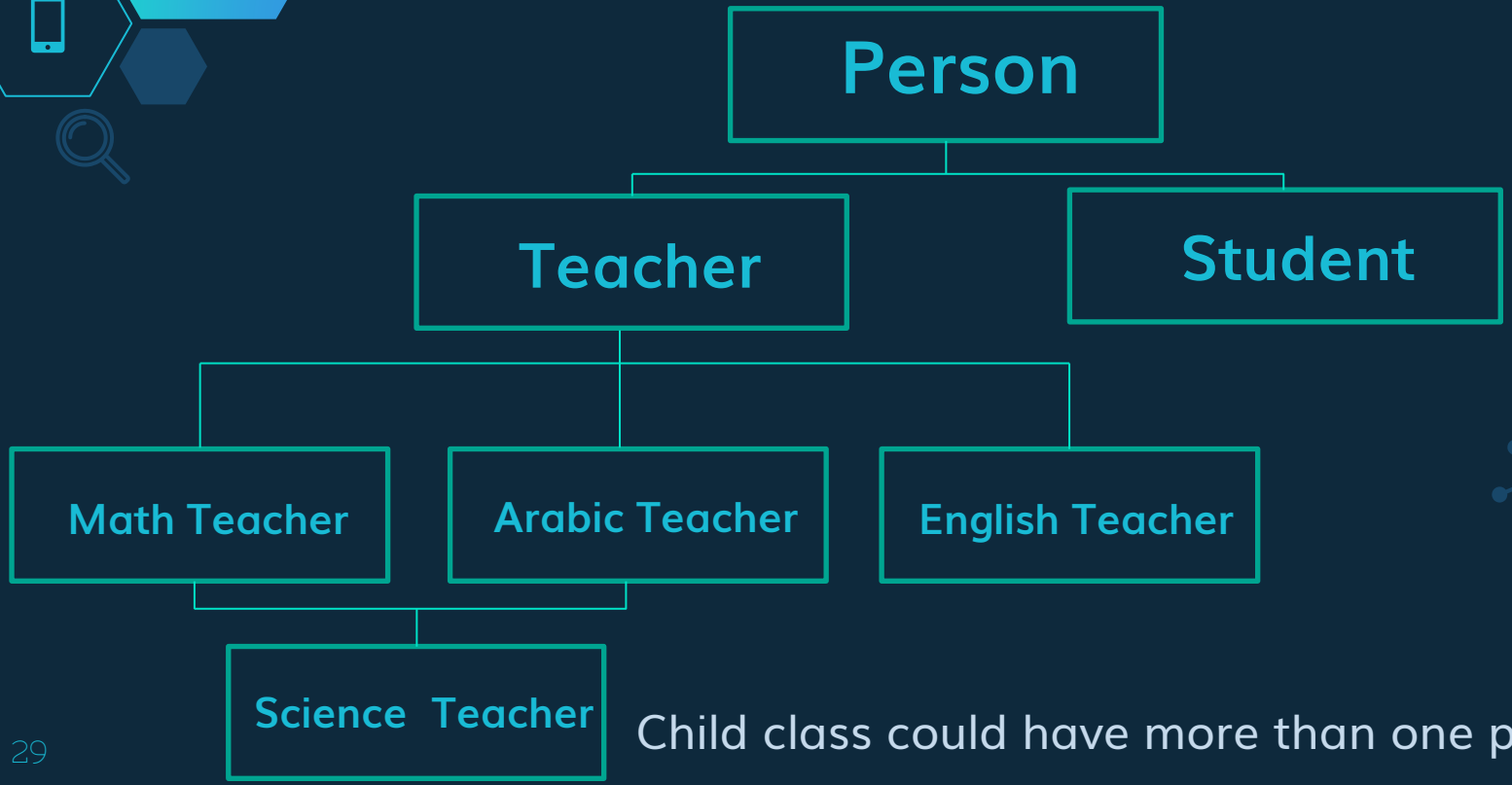
Multi-Level Inheritance , Multiple Inheritance

Multi-Level Inheritance (Single Inheritance)



Each child class has only one parent Class

Multiple Inheritance



Child class could have more than one parent Class

Multiple Inheritance

```
class Junior
{
public:
    void say_junior() {
        cout << "i am Junior" << endl;
    }
};

class Senior
{
public:
    void say_senior() {
        cout << "i am Senior" << endl;
    }
};

class Manager : public Junior, public Senior
{
public:
void say_manager() {
    cout << "i manage junior, and Senior" << endl;
}
};

int main()
{
    Manager m1;
    m1.say_junior();
    m1.say_senior();
    m1.say_manager();
}
```

```
Microsoft Visual Studio Debug Co
i am Junior
i am Senior
i manage junior, and Senior

E:\Courses\1. Programming\1.
ce.exe (process 9740) exited
Press any key to close this
```

Multiple Inheritance

Multiple Inheritance :

- **Not supported** by all programing Languages
- Other languages use **public interface** in case of more parents
- C++ **don't** have interfaces
- C++ **use** it instead of interface
- **Not recommended** to use
- **Causes The diamond problem (2 Copies of data)**

10

Inheritance Vs Composition

What is the difference ?

Inheritance Vs Composition

Public Inheritance:

- Achieve (**is a**) relationship
  - Employee is a person
  - Student is a person

Composition:

- Achieve (**has a**) relationship
  - Employee has a name
  - Student has marks

11

Static Members

All objects are related to it

Static Members

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

class Student
{
public:
    static int id;
    Student() {
        id++;
    }
    int getId() {
        return id;
    }
};

int Student::id = 0;

int main() {
    Student s1;
    cout << "S1 id = " << s1.getId() << endl;
    Student s2;
    cout << "S2 id = " << s2.getId() << endl;
    Student s3;
    cout << "S3 id = " << s3.getId() << endl;
}
```

```
Microsoft Visu
S1 id = 1
S2 id = 2
S3 id = 3

D:\Courses\1.
 17580) exited
Press any key
```

The static member are shared between all objects created from class

12

Static Methods

Could be accessed without creating an object

Static Methods

```
class Calculator {
public:
    int add(int a, int b) {
        return a + b;
    }

    static int multiply(int a, int b) {
        return a * b;
    }
};

int main()
{
    cout << Calculator::multiply(3, 5) << endl;
}
```

```
15

E:\Cours
ce.exe
Press a
```

The static method could be used without creating objects like(multiply function), but (add function) is not a static function, so it can only be used by creating objects

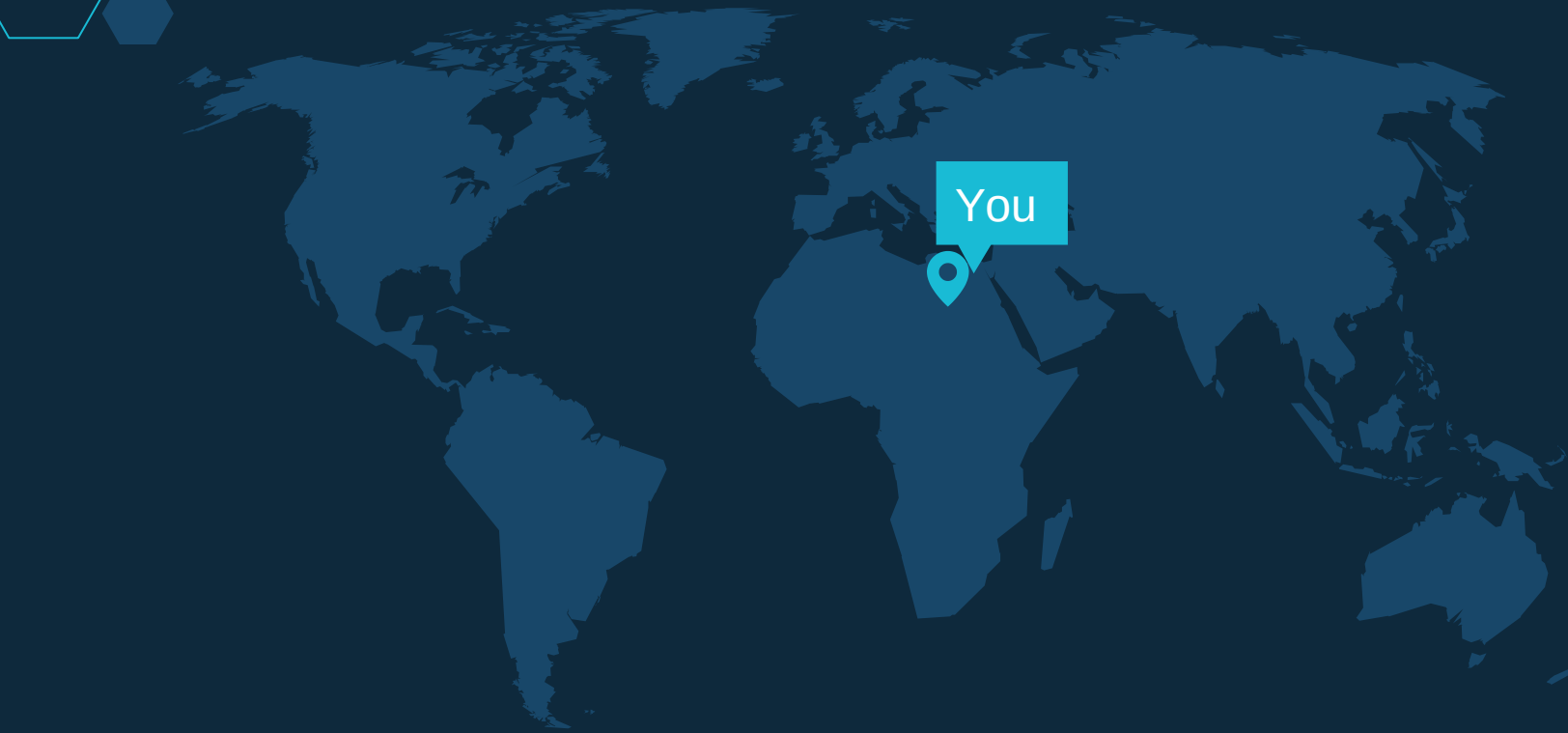
Thanks!

Any questions?

You can find me at:

- itiroutealexiti.com

Be sure that you are making a difference .




Break

OOP Revision Part 3

(Abstraction-Pointers-Polymorphism)

Agenda

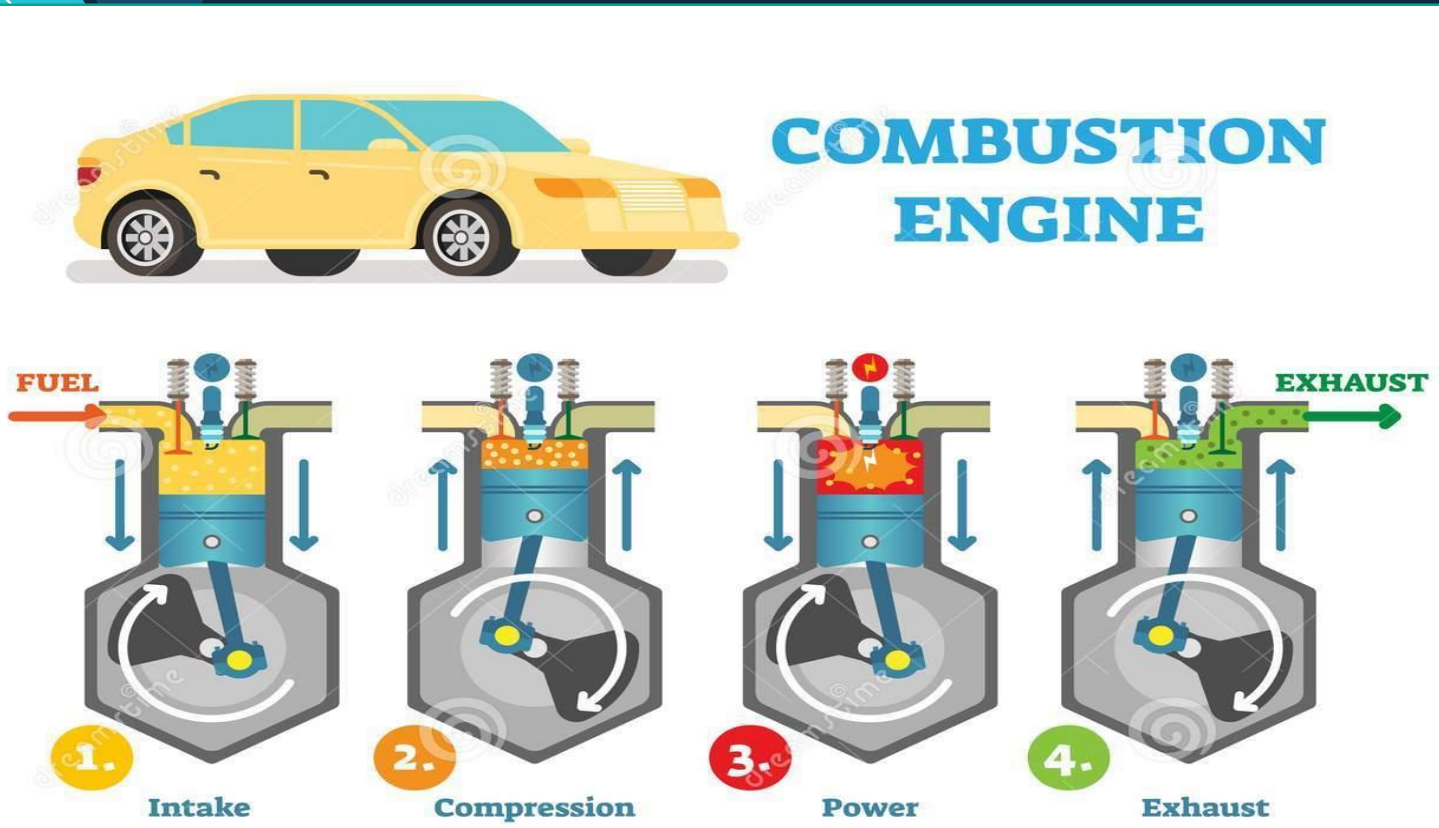
- What is Abstraction ?
- Template
- Pointers
- Declaring and Initializing Pointers
- Size of Pointers
- Storing Address in Pointer
- Accessing Data that the pointer points to
- Pointers and Arrays
- Passing Pointers to Functions
- Objects as Pointers
- Memory Types
- Heap Memory
- What is Polymorphism ?
- Exception Handling

1

What is Abstraction ?

???

What is Abstraction ?



What is Abstraction ?

- Third OOP Concept, It is an Interface describes the behavior of the class **without** any implementation of that class.
- It is declared by declaring at least **one pure** virtual function.
- Children must **override** on abstracted functions, and if they didn't override them, children will be abstracted classes.
- **Can not** take any objects from abstracted class.

What is Abstraction ?

```cpp
class Parent {
protected:
    string lname;
public:
    //abstracted function
    virtual void showName() = 0;
};

class Child : public Parent {
private:
    string fname;
public:
    void setNames(string f,string l){
        lname = l;
        fname = f;
    }
    //must declare the abstracted function
    void showName() {
        cout << fname << " " << lname;
    }
};
int main()
{
    Child c;
    c.setNames("Ibrahim", "Yasser");
    c.showName();
}
```

```
Microsoft Visual
Ibrahim Yasser
E:\Courses\1. Pr
cess 15384) exit
Press any key to
```

Ex 1:

- Create Abstracted Shape class with pure virtual function calcArea = 0;
- Then create three classes
  (Rectangle, Triangle, Circle)
  And override the calcArea function

2

Template

???

What is Template?

- **Template :** A temporary data type defined at certain time.

```
template <class t, class t1>
class MyPair
{
private:
    t key;
    t1 value;
public:
    void setKey(t n) {
        key = n;
    }
    void setValue(t1 x) {
        value = x;
    }
    t getKey() {
        return key;
    }
    t1 getvalue() {
        return value;
    }
};

int main() {
    MyPair<char,int> p;
    p.setKey('a');
    p.setValue(5);
    cout << p.getKey() << endl;
    cout << p.getvalue() << endl;

}
```

```
a
5

E:\Co
cess
Press
```

```
template <class t, class t1>
class MyPair
{
private:
    t key;
    t1 value;
public:
    void setKey(t n) {
        key = n;
    }
    void setValue(t1 x) {
        value = x;
    }
    t getKey() {
        return key;
    }
    t1 getvalue() {
        return value;
    }
};

int main() {
    MyPair<double,string> p;
    p.setKey(10.5);
    p.setValue("ahmed");
    cout << p.getKey() << endl;
    cout << p.getvalue() << endl;

}
```

```
10.5
ahmed

E:\Cour
cess 50
Press a
```

3 Pointers

???

What are Pointers ?

- **A variable:** whose value is an address
- What can be at that address ?
  - Another variable
  - A function
- So pointers **point** to variable or function , if X is an integer and it's value is 10 then you can declare a pointer that points to it.
- Pointers are used to allocate memory on **heap** , this memory doesn't even have names for variables, so the only way to access this data is via **Pointers.**

4 Declaring and Initializing Pointers

How to declare and initialize ?

Declaring and Initializing Pointers

```
Variable_type* Pointer_name;

int* x;
double* y;
char* z;
string* s;

Declaring
```

```
Variable_type* Pointer_name {nullptr};

int* x    { nullptr };
double* y { nullptr };
char* z   { nullptr };
string* s { nullptr };

Intializing
```

- Initialize pointer to 'point nowhere' using **nullptr**

Accessing Pointers Address

```cpp
int main() {

    int num = 10;
    cout << "Value of num = " << num << endl;
    cout << "Size of num = " << sizeof num << endl;
    cout << "Address of num = " << & num << endl;

}
```

```
Microsoft Visual Studio Debug Cons
Value of num = 10
Size of num = 4
Address of num = 00BFFD5C

E:\Courses\1. Programming\1. I
cess 10628) exited with code 0.
Press any key to close this wi
```

```cpp
int main() {

    int num = 10;
    int *p = &num;
    cout << "Address of num = " << &num << endl;
    cout << "Value of p = " << p << endl; //num address
    cout << "Size of p = " << sizeof p << endl;
    cout << "Address of p = " << & p << endl;
    p = nullptr; //points nowhere
    cout << "Value of p = " << p << endl; //garbage

}
```

```
Microsoft Visual Studio Debug
Address of num = 002AFCFC
Value of p = 002AFCFC
Size of p = 4
Address of p = 002AFCF0
Value of p = 00000000

E:\Courses\1. Programming\1
cess 7096) exited with code
Press any key to close this
```

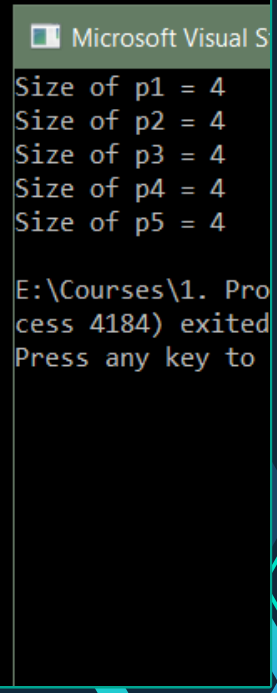
5

Size of Pointers

What is the size of what pointers have ?

Size of Pointer

```
int main() {

    int* p1{ nullptr };
    cout << "Size of p1 = " << sizeof p1 << endl;

    double* p2{ nullptr };
    cout << "Size of p2 = " << sizeof p2 << endl;

    unsigned long long* p3{ nullptr };
    cout << "Size of p3 = " << sizeof p3 << endl;

    string* p4{ nullptr };
    cout << "Size of p4 = " << sizeof p4 << endl;

    vector<string>* p5{ nullptr };
    cout << "Size of p5 = " << sizeof p5 << endl;

}
```



- Don't confuse the size of pointer, and the size of what it points to
- All pointers in the system have **the same size**, they may points to large or small types.

6

Storing Address in Pointer

How to store ?

Storing Address in Pointer

- The type of pointer must be **the same** type of what it points to
- The compiler will make sure the address stored is the same type of pointer type

```
int main() {
    int x = 10;
    double y = 3.7;

    int* p1{ nullptr };
    p1 = &x;            //ok
    p1 = &y;            //Error
}
```

1 Error | 0 Warnings | 0 Messages | Build + IntelliSense

Description	Project	File
a value of type "double *" cannot be assigned to an entity of type "int *"	ITI-Week8	ITI-Week8.cpp

Storing Address in Pointer

- Pointers are variables, so they can change
- Pointers can be null

```
int main() {
    int x = 15;
    int y = 7;

    int* p{ nullptr };
    p = &x;      //    p points to x
    p = &y;     // Now p points to y
}
```

7

Accessing Data that the pointer points to

How to access ?

Accessing Data that the pointer points to

```
int main() {
    int x = 100;
    int y = -5;

    int *p = &x;

    cout << *p << endl << endl;//100
    *p = 200;
    cout << *p << endl;          //200
    cout << x << endl << endl; //200

    p = &y;//Now p points to y
    cout << *p << endl << endl;//-5
    *p *= 5;
    cout << *p << endl;          //-25
    cout << y << endl;           //-25
}
```

```
Microso
100

200
200

-5

-25
-25

E:\Courses
cess 1332)
Press any
```

- You can access the data that the pointer points to using * operator

8

Pointers and Arrays

How to deal ?

Pointers and Arrays

- The value of an array is the address of the first element in the array
- The value of a pointer is an address
- When the pointer points to an array it's value equal the address of first element

```cpp
int main() {
    int arr[]{ 10,20,30 };
    cout << arr << endl;  // address of first element
    cout << *arr << endl; // value of first element

    int* p{ arr };
    cout << p << endl;  // address of first element
    cout << *p << endl; // value of first element
//  To print the array through the pointer:
    cout << "===== Array Values =====\n";
    cout << p[0] << endl;
    cout << p[1] << endl;
    cout << p[2] << endl;
    cout << "===== Array Addresses =====\n";
    cout << p << endl;
    cout << (p+1) << endl;
    cout << (p+2) << endl;
}
```

```
Microsoft Visual Studio Debug Co
008FFA2C
10
008FFA2C
10
===== Array Values =====
10
20
30
===== Array Addresses =====
008FFA2C
008FFA30
008FFA34

E:\Courses\1. Programming\1.
cess 7192) exited with code 0
Press any key to close this w
```

9

Passing Pointers to Functions

How to pass ?

Passing Pointers to Functions

- Pass by reference with pointer parameters
- The function parameter can be a pointer or address of a variable

```
void double_salary(int* s) {
    *s *= 2;
}

int main() {
    int salary = 1000;
    cout << salary << endl;
    double_salary(&salary);
    cout << salary << endl;

}
```

```
Micro
1000
2000

E:\Cours
cess 862
Press an
```

10

Objects as Pointers

How to access ?

Objects as Pointers

Note 1:

- Any class has an internal pointer called this, created automatically

Note 2:

- To access data of object pointer use " -> "Operator instead of " . " Operator

```
class Person {
private:
    string name;
    int age;
public:
    Person(){}
    Person(string name, int age) {
        this->name = name;
        this->age = age;
    }
    void setName(string name) {
        this->name = name;
    }
    void setAge(int age) {
        this->age = age;
    }
    string getName() {
        return name;
    }
    int getAge() {
        return age;
    }
};

int main() {
    Person* p = new Person;
    p->setName("Ahmed");
    p->setAge(30);
    cout << p->getName() << endl;
    cout << p->getAge() << endl;
}
```

```
Ahmed
30

E:\Cour
cess 11
Press a
```

11

Memory Types

How many types ?

Memory Types

Stack	Heap
A **static** memory allocated at **compile** time	A **dynamic** memory allocated at **run** time
Variables are stored **continuously** in memory at **compile** time	Variables are stored **randomly** in memory at **run** time
Access to memory is **faster**, as variables are already arranged **automatically**	Access to memory is **slower**, as variables are not arranged and must be accessed **manually** through **pointers**
Size is **fixed** and **limited**	Size is **dynamic** and **larger**, (need to **delete** unused variables)

12

Heap Memory

How to allocate heap ?

Heap Memory

- We use pointer to allocate new memory at heap at run time, using **new** keyword

```
int main() {

    int* p{ nullptr };
    p = new int;         // allocate an int in heap
    cout << p << endl; // address of new int
    cout << *p << endl;// value of new int(garbage)
    *p = 10;
    cout << *p << endl;// value of new int(10)

}
```

```
Microsoft
00628A40
-842150451
10

E:\Courses\1
cess 3864) e
Press any ke
```

Heap Memory

- We deallocate variable storage, using **delete** keyword

```
int main() {
    int* p{ nullptr };
    p = new int;         // allocate an int in heap

    // code ...

    delete p;            // Frees the allocated storage
}
```

Heap Memory

- We use **new[ ]** to allocate storage for array in heap
- And we use **[ ]delete** to delete array from heap

```
int main() {
    int* p { nullptr };
    int size = 0;

    cout << "Enter Array size\n";
    cin >> size;

    p = new int[size]; // allocate array in heap

    // code ...

    delete [] p;       // Frees the allocated storage
}
```

13

What is Polymorphism ?

???

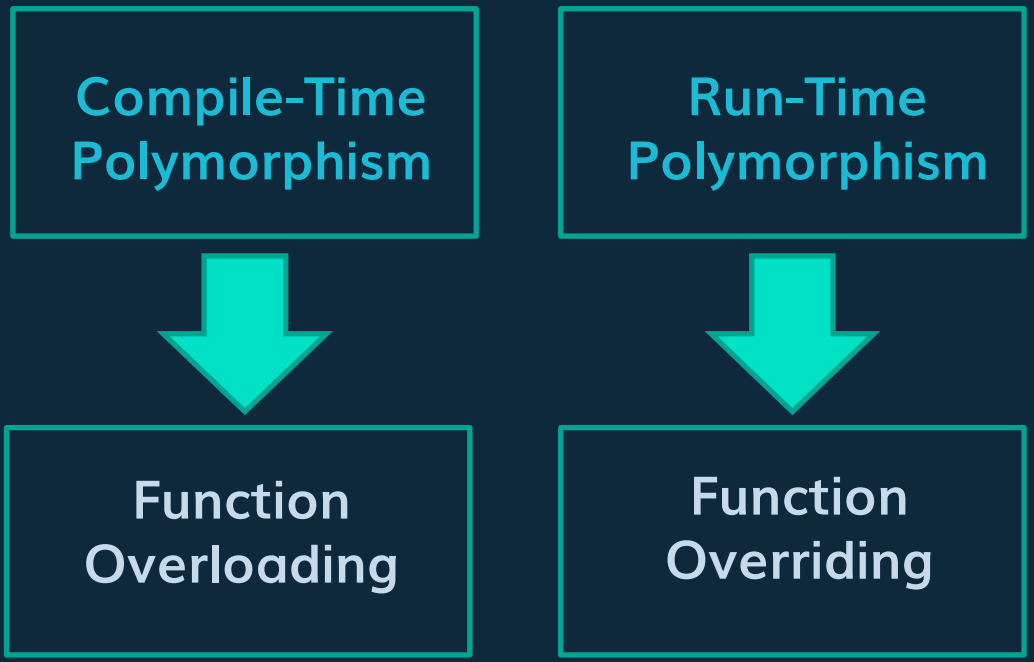
What is Polymorphism ?

- The fourth OOP concept that refers to the ability of object to take multiple forms, so objects with common parent may have the same name of function but with different behaviors at run time .

Polymorphism

Example 2

What is Polymorphism ?



Difference between overloading and overriding

- Overloading:
  - A rewritten function with the same name of another function but with different signature (Number of parameters, Type of parameters) and its processing happened in compile time (Compile Time Polymorphism).
- Overriding:
  - A rewritten function in child class with the same name of another function in parent class and with the same signature (Number of parameters, Type of parameters) and its processing happened in run time (Run Time Polymorphism).

14

Exception Handling

To deal with errors

Exception Handling

- What happens if y = 0 ?
  - Crash ?
  - It depends!

```
int divide(int x, int y) {
    return x / y;
}

int main() {
    cout << divide(5, 0) << endl;
}
```

Exception Handling

- What happens if y = 0 ?
  - Crash ?
  - It depends!

```
int divide(int x, int y) {
    if (y == 0) {
        // what to do ?
    }
    else {
        return x / y;
    }
}

int main() {

    cout << divide(5, 0) << endl;
}
```

Exception Handling

- Exception:
  - An object or primitive type where error occurred

- Try Block:
  - The block that contain the code that would cause error

- Catch Block:
  - The block that handle the error

Exception Handling

- C++ standard library has exception class you can use it to handle errors , #include < exception >

- And you can inherit from it many other exception classes to create your own exception class by editing what() function, and through polymorphism there will be only one exception class called when it's error occurred

```
const char* what() const throw()
{
    return "Can not divide by zero";
}
```

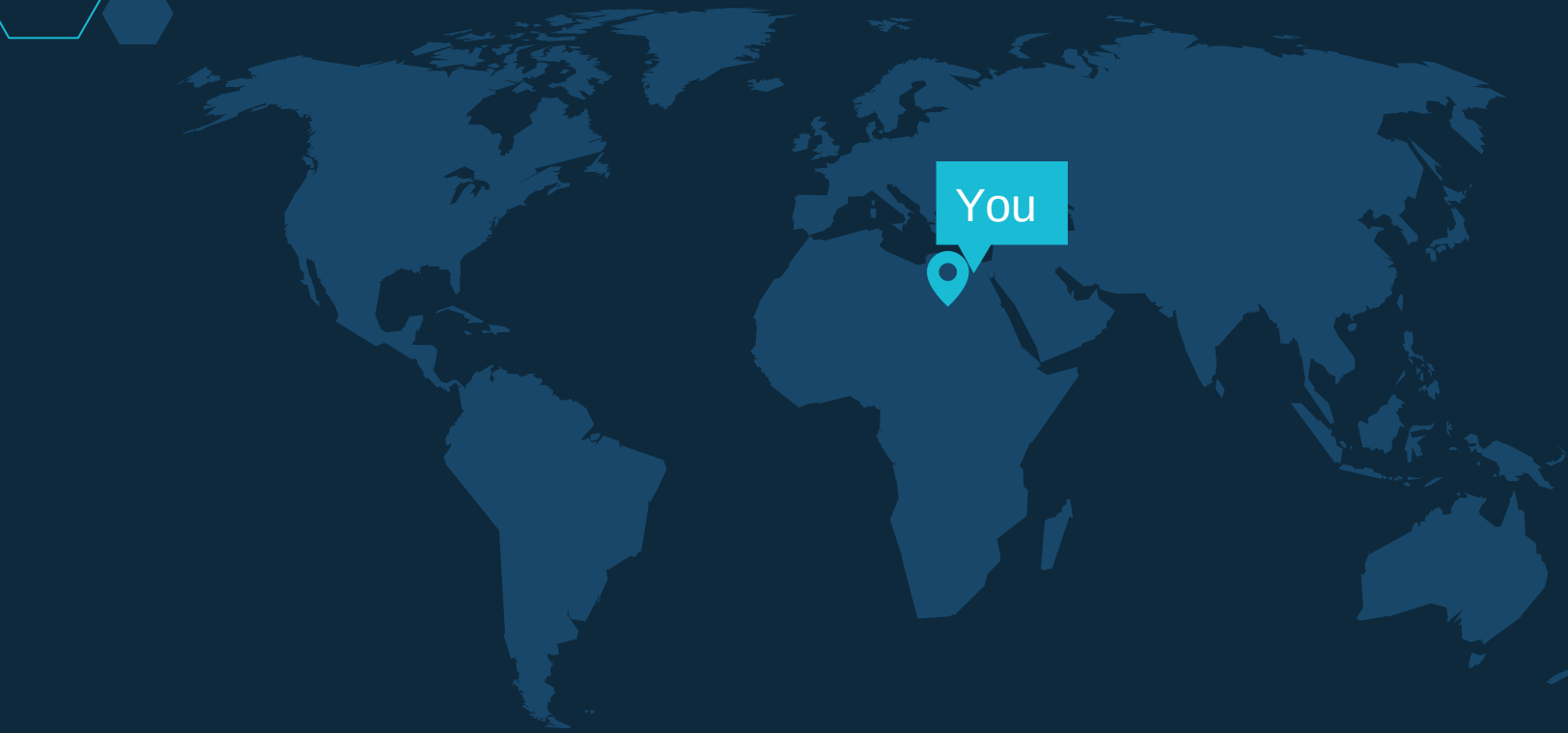
Thanks!

Any questions?

You can find me at:

- itiroutealexiti.com

Be sure that you are making a difference .




Break