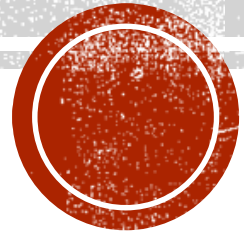
OBJECT ORIENTED PROGRAMMING...

Working with Classes



RECAP LAST WEEK – Q1

Which of the following programming language(s) is/are considered as Object-Oriented programming languages?

1. Python
2. C
3. Java
4. Pascal
5. Basic

RECAP LAST WEEK – Q1 ANSWER

Which of the following programming language(s) is/are considered as Object-Oriented programming languages?

1. **Python**
2. C
3. **Java**
4. Pascal
5. Basic

RECAP LAST WEEK – Q2

In "Procedural Oriented Programming",

1. adding new data and function is easy.
2. there is NO proper way for hiding data, so it is less secure.
3. overloading is possible.
4. program is divided into small parts called objects.
5. data is more important than functions.

RECAP LAST WEEK – Q2 ANSWER

In "Procedural Oriented Programming",

1. adding new data and function is easy.
2. **there is NO proper way for hiding data, so it is less secure.**
3. overloading is possible.
4. program is divided into small parts called objects.
5. data is more important than functions.

DATA TYPES

DATA TYPES

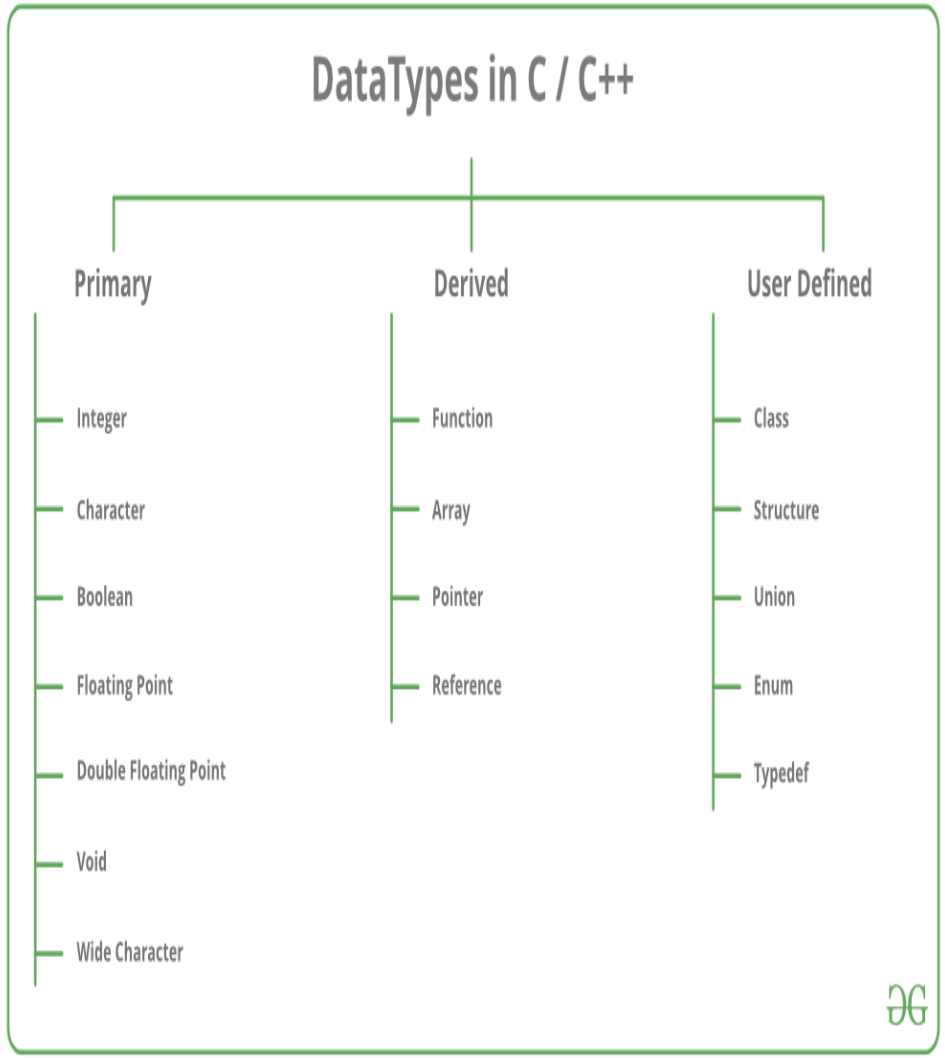
- The type of data the variables can store.
- The compiler allocates some memory for the variable at the declaration based on the data-type.
- Every data type requires a different amount of memory.

Example:

- Integer: Keyword (int) - Requires 4 bytes of memory space and ranges from -2147483648 to 2147483647.
- Character: Keyword (char) - Requires 1 byte of memory space and ranges from -128 to 127 or 0 to 255.

DATA TYPES

- **Build-in/Primitive Data Types**:
  - Built-in or predefined data types
  - Can be used directly by the user to declare variables.
  - Example: int, char , float, bool, void etc.
- **Derived Data Types:**
  - Derived from the primitive or built-in datatypes
  - Example: Function, Array, Pointer, Reference
- **User-Defined Data Types**:
  - Defined by user.
  - Example: Class, Structure, Enum, Typedef



DERIVED DATA TYPES

Derived from the primitive or built-in datatypes.

- **Function -** a block of code or program segment that is defined to perform a specific well-defined task. All the lines of code are put together inside a single function and this can be called anywhere required

  Syntax: ReturnType FunctionName(parameters)

- **Array -** a collection of items stored at continuous memory locations. Represent many instances in one variable

  Syntax: DataType ArrayName[size_of_array];

- **Pointers -** a symbolic representation of addresses. A pointer points to an address which holds the data

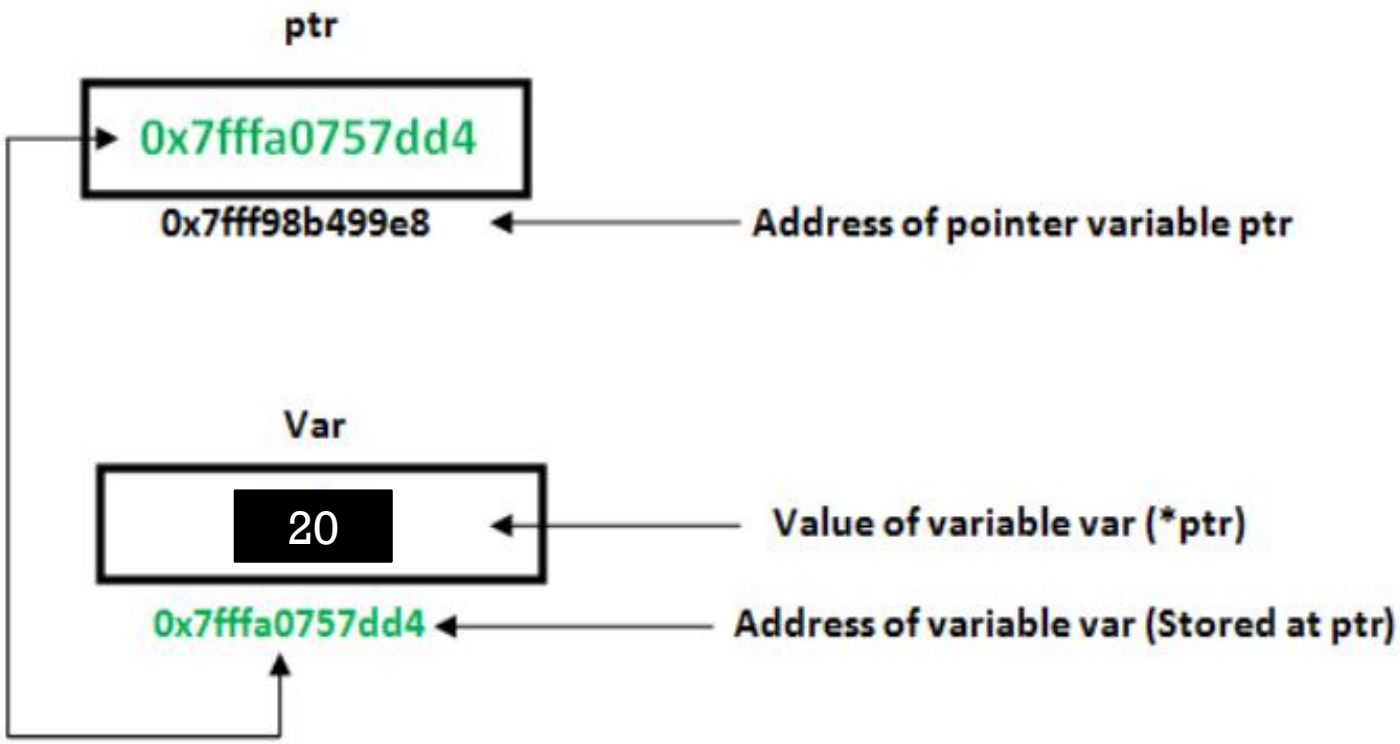
  Syntax: datatype *var_name;

- **References -** an alternative name for an existing variable. Declared as a reference by putting '&'

ACTIVITY: WRITE THE OUTPUT

```
Value at ptr = 0x6ffdfc
Value at var = 20
Value at *ptr = 20
20 0x6ffdfc
Value of var = 40
```

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

void geeks()
{
    int var = 20;

    int* ptr;

    ptr = &var;

    cout << "Value at ptr = " << ptr << "\n";
    cout << "Value at var = " << var << "\n";
    cout << "Value at *ptr = " << *ptr << "\n";

        int &ref = var;
        cout << ref <<" "<<&ref<< endl;

        ref = 40;
        cout<< "Value of var = " <<var<<endl;
}

int main()
{
    geeks();
     return 0;
}
```


ptr
0x7fffa0757dd4
0x7fff98b499e8
Address of pointer variable ptr
Var
20
Value of variable var (*ptr)
0x7fffa0757dd4
Address of variable var (Stored at ptr)

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

int main()
{
    int i = 10; // simple or ordinary variable.
    int* p = &i; // single pointer
    int** pt = &p; // double pointer
    int*** ptr = &pt;   // triple pointer

    cout << "i = " << i << "\t"
        << "p = " << p << "\t"
        << "pt = " << pt << "\t"
        << "ptr = " << ptr << '\n';

    int a = 5;
    int& S = a;
    int& S0 = S;
    int& S1 = S0;

    cout << "a = " << a << "\t"
        << "S = " << S << "\t"
        << "S0 = " << S0 << "\t"
        << "S1 = " << S1 << '\n';
    return 0;
}
```

USER-DEFINED DATA TYPES

- Variables of simple data types (float, char, and int) represent one item of information.
  - float: Price of sugar, char: a character, int: number of students
- But just as groceries are organized into bags, employees into departments, and words into sentences…..
- Organize simple variables into more complex entities.
- In C/C++ we define *structure to get this done.*

EXERCISE

Write a simple C structure to store details of a student (Name, age, height & weight) and print these student details on the console

```
#include <stdio.h>

struct Student {
    char name[50];
    int age;
    float height;
    float weight;
};

int main() {
      struct Student s1 = {“Mala", 20, 175.5, 65.0};
          printf("Name: %s\n", s1.name);
          printf("Age: %d\n", s1.age);
          printf("Height: %.2f cm\n", s1.height);
          printf("Weight: %.2f kg\n", s1.weight);
    return 0;
}
```

STRUCTURES

- A user-defined data type in C/C++
- Store a group of data (similar data types or non-similar data types)
- Creates a data type that can be used to group items of possibly different types into a single type
- The 'struct' keyword is used to create a structure.
- Structure members can be initialized with declaration in C++
- Contain two types of members:
  - Data Member: Variables of different/similar data types in C++.
  - Member Functions: Can include functions inside a structure declaration.

```cpp
#include<iostream>      //include the header file
using namespace std;    //use names for objects and variables

struct Student{
    string name = " ";
    int age =0;
    float height = 0.0;
    float weight = 0.0;
};

int main()
{
    Student S1;
    S1.name="Mala";
    S1.age=10;
    S1.height=7.5;
    S1.weight=23;

    cout<<S1.name<<endl;
    cout<<S1.age<<endl;
    cout<<S1.height<<endl;
    cout<<S1.weight<<endl;
}
```

STRUCTURES

- A structure is an aggregate of data types built using elements of other types

```
struct date{
  int day;
  char month[10];
  int year;
} ;
```

- Each structure definition must end with a semicolon ; (structure termination)

- The data items in a structure are called the *members* of the structure.
- Has three members:
  - *day*-an integer,
  - *month*-a character array,
  - *year*-an integer.

[ACTIVITY] Declare a structure variable called 'today'

```
date today;
```

STRUCTURE SYNTAX

Structure Name

Creates a new data type

```
struct date{
        int day;
        char month[10];
        int year;
        } ;
```

Structure members

Structure Termination

STRUCTURES

- Defining a Structure Variable
  - Structure variables are declared like variables of other types
  - Example:

    date today, dateArray[10], *datePtr, &dateRef = dateObject;
  - Arrays of structures can also be defined.

    date dateArray[10];
  - The variable name must be used to distinguish one variable from another (date1, date2)
- In C - need to include the keyword struct in defining structure variables. But in C++ it is optional.

Example C

Struct date today; (In C++ the keyword struct is not necessary)

ACCESSING MEMBERS OF THE STRUCTURE

- Member access operators:
  - Dot operator (.)
- Syntax

  Name of the structure variable. member name;
- Structure members are treated just like other variables.

[ACTIVITY] Print member day of the variable 'today'

```
cout <<"Day"<< today.day<<endl;
```

EXERCISE 1: INITIALIZING STRUCTURE VARIABLES

```
#include<iostream>
using namespace std;
//defining the structure date
struct date{
        int day;
        char month[10];
        int year;
}; // end of structure definition
int main()
{
    date today={28,“November",2024};
        cout<<"Day:"<<today.day
                <<" Month:"<<today.month
                <<"Year:"<<today.year<<endl;
}
```

ACTIVITY

```
#include<iostream>
#include<string.h>

using namespace std;

//defining the structure date
struct date{
    int day;
    char month[10];
    int year;
}yesterday,d1; // end of structure definition


int main()
{
    yesterday.day=27;
    strcpy(yesterday.month,"November");
    yesterday.year=2024;

        d1=yesterday;

        cout<<"YESTERDAY"<<endl
                <<"Day:"<<yesterday.day
                <<" Month:"<<yesterday.month
                <<" Year:"<<yesterday.year<<endl;

    //Display values of d1
        cout<<"YESTERDAY = D1"<<endl
                    <<"Day:"<<d1.day
                    <<" Month:"<<d1.month
                    <<" Year:"<<d1.year<<endl;

        date today={28,"November",2024};

        cout<<"TODAY"<<endl
            <<"Day:"<<today.day
            <<" Month:"<<today.month
            <<" Year:"<<today.year<<endl;

        d1=today;
        cout<<"D1 = TODAY"<<endl
                    <<"Day:"<<d1.day
                    <<" Month:"<<d1.month
                    <<" Year:"<<d1.year<<endl;
}//end of main
```

STRUCTURES...

- In standard C: Functions are **not** allowed to be defined within a structure; Hold only data
- In C++ : Both Data & Functions can be declared in the structure.
- The members of a structure are visible to all functions that are within the scope of the structure (by Default members are PUBLIC).

ACTIVITY

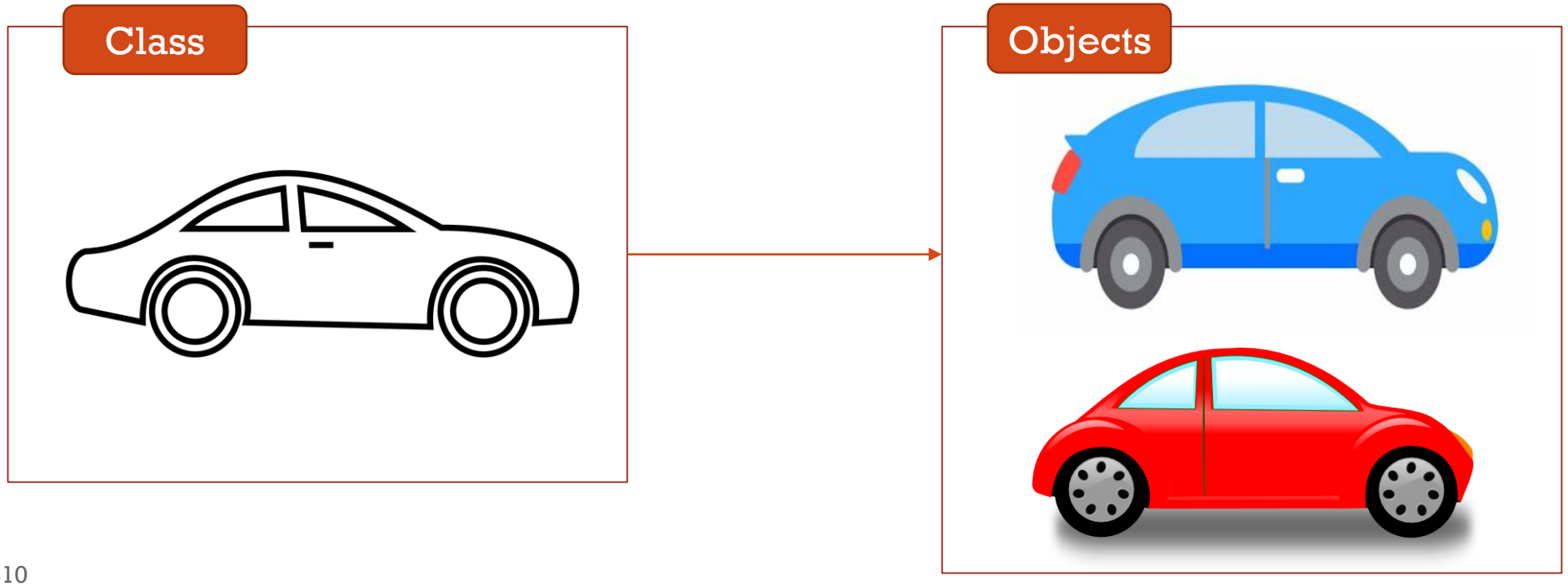
```
2   #include<string.h>
3   using namespace std;
4
5   //defining the structure date
6   struct date{
7       int day;
8       char month[10];
9       private:
10      int year=2024;
11
12      public:
13      void printYear()
14      {
15          cout<<"Year: "<<year<<endl;
16      }
17  }; // end of structure definition
18
19  int main()
20  {
21      date today;
22      today.day=28;
23      strcpy(today.month,"November");
24
25      cout<<"Day:"<<today.day
26          <<" Month:"<<today.month;
27          today.printYear();
28  }
```

STRUCTURES VS CLASSES

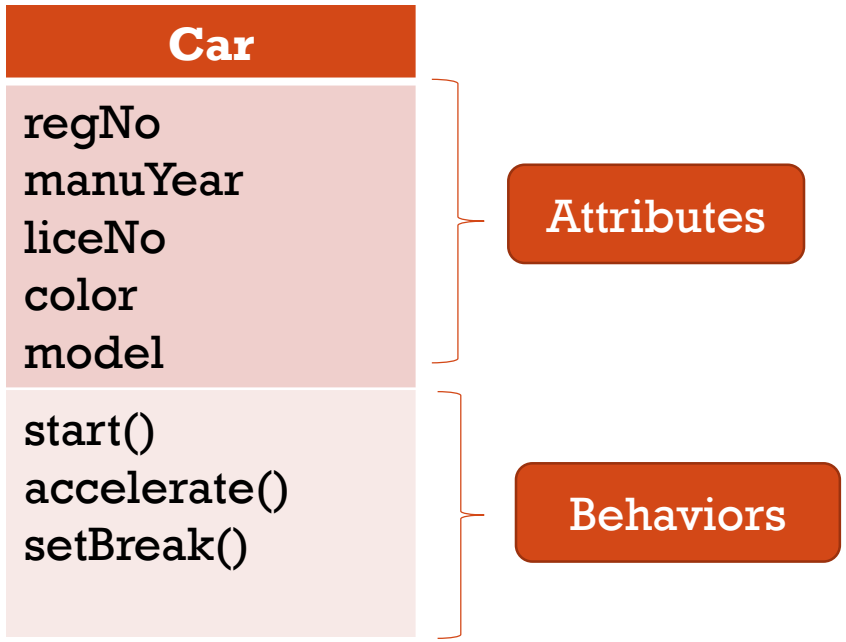
- Classes in C++ are a natural evolution of the C notion of structures.
- Both are user-defined data types.
- Both hold a bunch of different data types.
- A class is a user-defined blueprint or prototype from which objects are created.
- A class combines the fields and methods(member function which defines actions) into a single unit.
- The syntactical distinction between structures and classes in C++ is minimal…
- Classes in C++: by default members are PRIVATE.

WORKING WITH CLASSES

- Two main building blocks of Object Orientation.
- A class is a **template** for objects, and an object is an **instance** of a class.



CLASSES IN C++




```
class car
{
public:
        int regNo;
        int manuYear;
        string liceNo;
        string color;
        string model;

        void start();
        int accelerate();
        void setBreak();
};
```

Access specifier

Data Members

Member Functions

Class Body

End of the class Definition

CLASS MEMBERS

Functions that are declared within a class are called **member functions**

Variables that are declared within a class are called **data members**

CREATING OBJECTS

- An object is created from a class.
- To create an object, specify the class name, followed by the object name.
  - Example: car myRedCar;
- To access the class attributes/member functions, use the dot syntax (.) on the object.
  - Example: myRedCar.regNo;

```
Int main()
{
car myRedCar;
car myBlueCar={456,2018,"CP345","Blue","B"};
}
```

DECLARE OBJECTS

Syntax:

```
ClassName ObjectName;
```

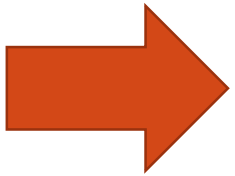
DOT OPERATOR

- Whenever you refer to a member of a class from code outside the class definition, you must prefix the member's name with the *dot operator* and the name of the object it belongs to.
- The member may be either data or a function.

objectName.memberName

ACTIVITY

```
1   #include<iostream>
2   using namespace std;
3
4   class car
5   {
6       int regNo;
7       int manuYear;
8       string liceNo;
9       string color;
10      string model;
11
12      void start();
13      int accelerate();
14      void setbreak();
15  };
16
17  int main()
18  {
19      car myRedCar ={234,2017,"Wp4567","Red","C"};
20          car myBlueCar={456,2018,"CP345","Blue","B"};
21
22      cout<<"My Red Car: "<<myRedCar.liceNo<<" "<<endl;
23      cout<<"My Blue Car: " <<myBlueCar.liceNo<<" "<<endl;
24  }
```

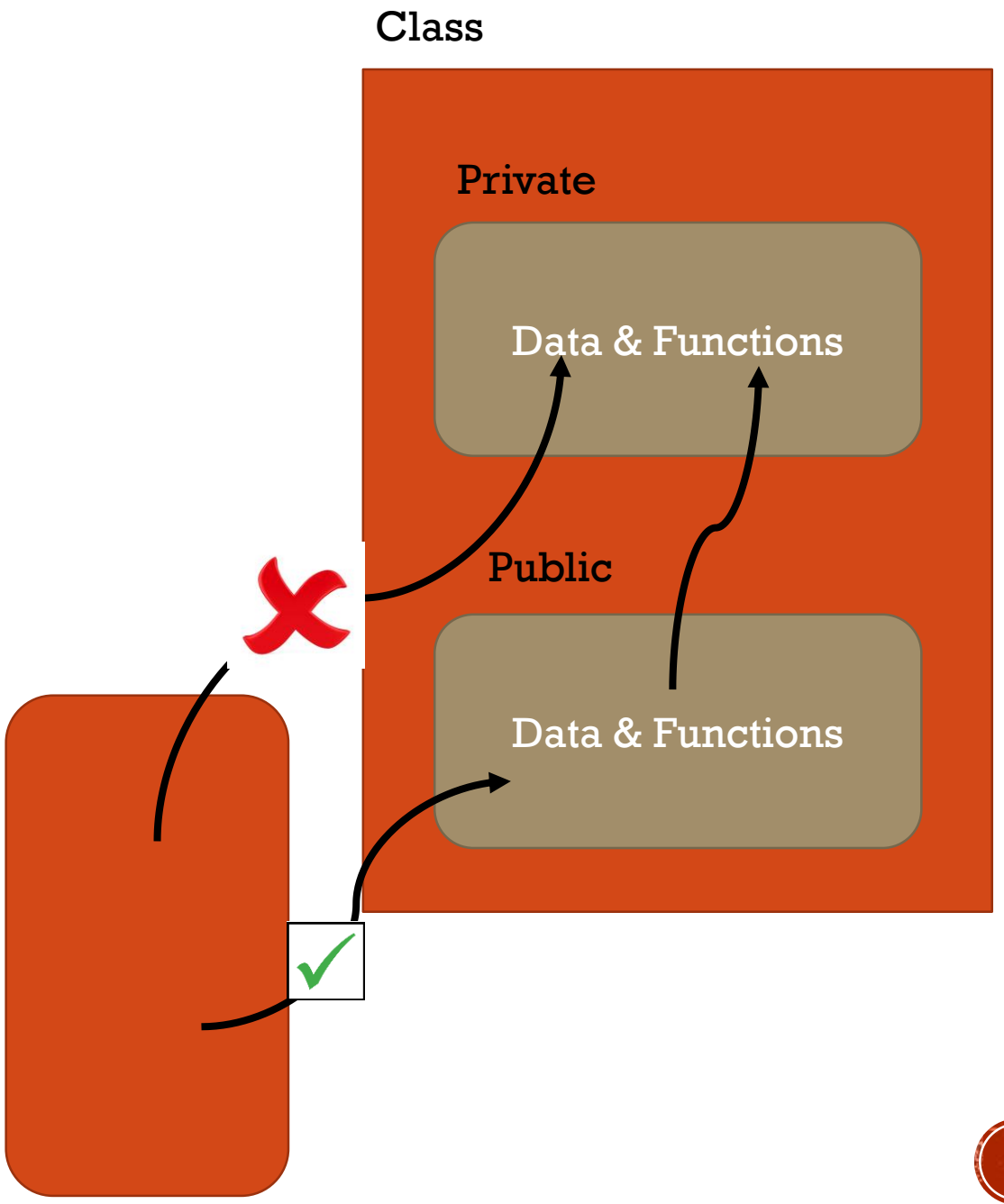


Compilation Error

The default access model for classes is private so all members after the class header and before the first member access specifier are private

ACCESS SPECIFIERS

- Define how the members (attributes and methods) of a class can be accessed.
- *Data hiding:* Key feature of object-oriented programming
  - The data is concealed/wrapped within a class so that it cannot be accessed mistakenly directly or by functions outside the class.
  - Private data or functions can only be accessed from within the class.
  - Public data or functions are accessible from outside the class.
- Hide data from parts of the program that don't need to access it.
- Data in one class is hidden from other classes.

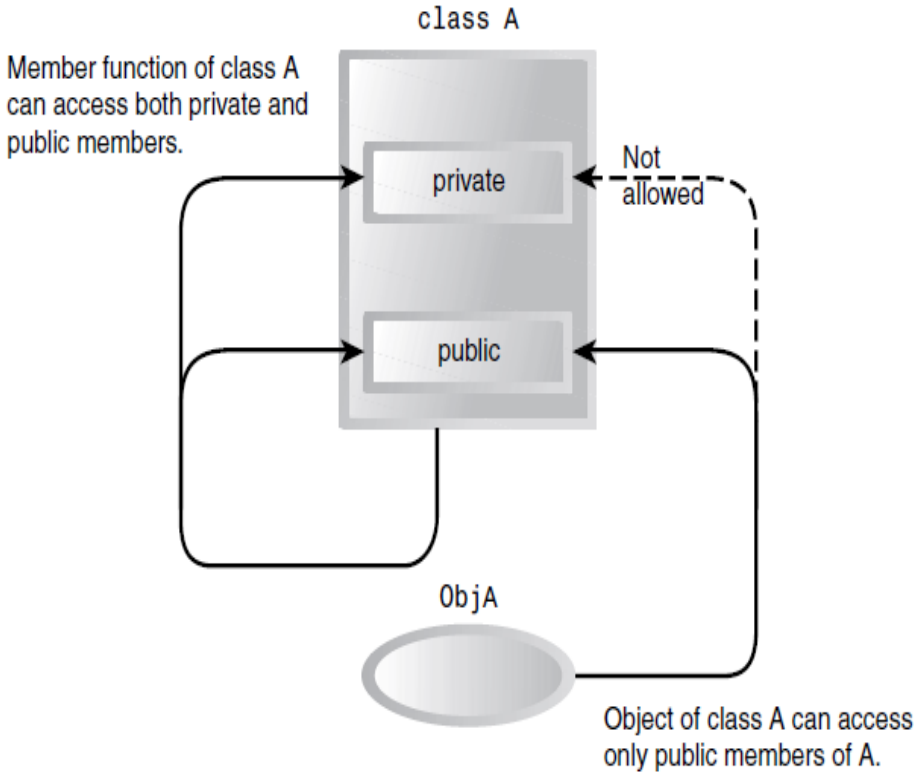


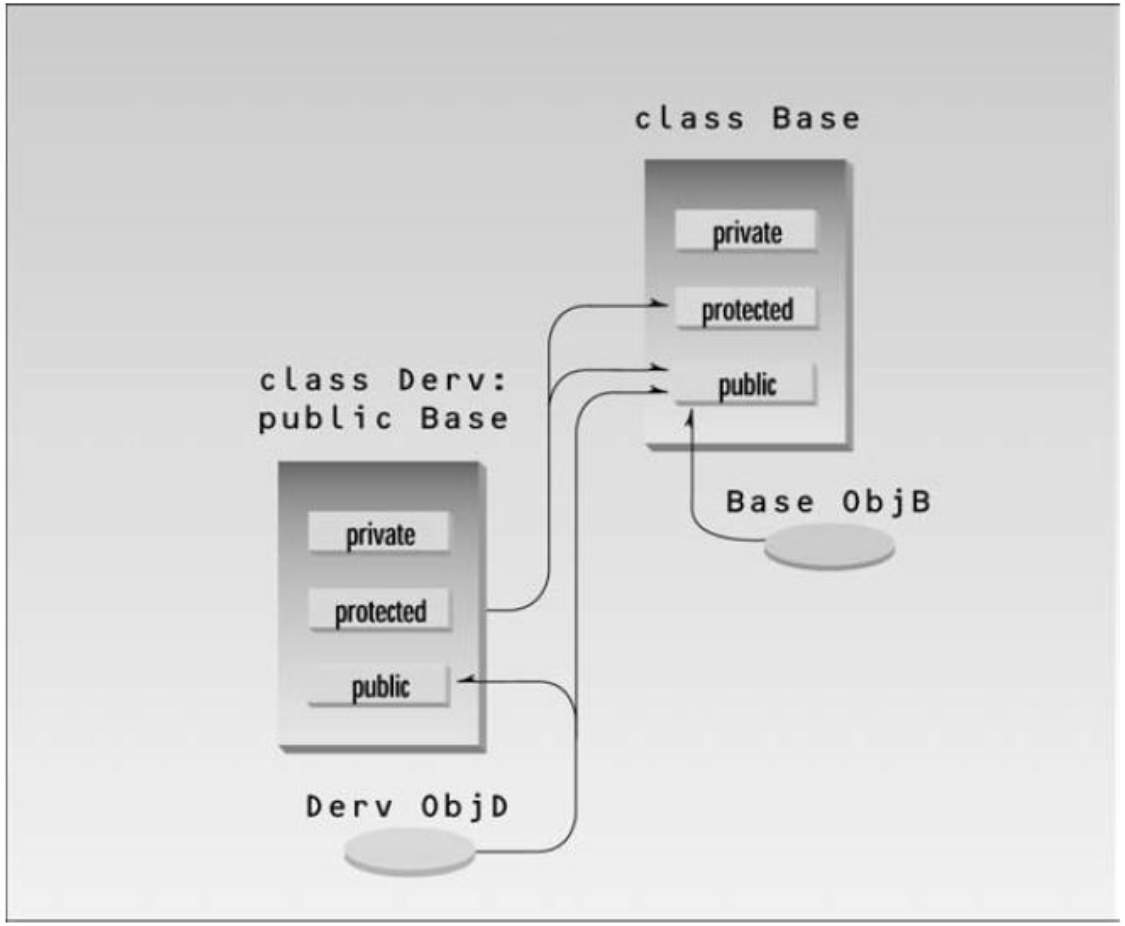
ACCESS SPECIFIERS

Keyword **public:** makes subsequent members public (can be accessed by non-member functions)

Keyword **private:** makes subsequent members private (cannot be accessed by non-member functions)

Keyword **protected:** Somewhat similar to the keyword private.

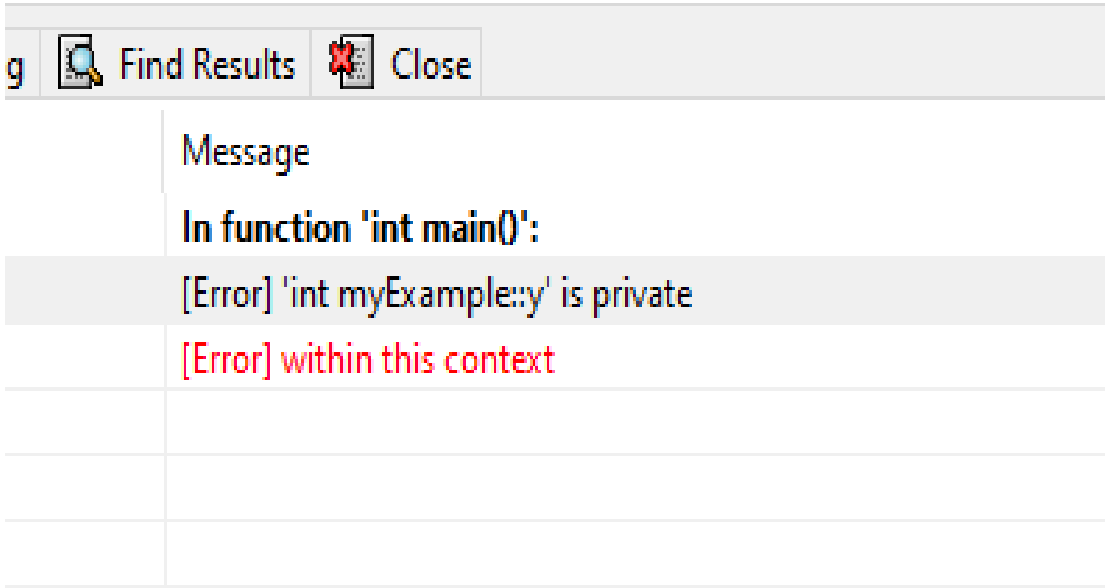

class A
Member function of class A can access both private and public members.
private
public
Not allowed
ObjA
Object of class A can access only public members of A.


class Base
private
protected
public
Base ObjB
class Derv: public Base
private
protected
public
Derv ObjD

ACCESS SPECIFIERS-EXAMPLE

```
class myExample
{
 public:   // Public access specifier
   int x;  // Public attribute
 private:  // Private access specifier
   int y;  // Private attribute
};

int main() {
 myExample E1;
 E1.x = 3;
 E1.y = 4;
}
```

OUTPUT ?



DEFINING CLASSES

```
class ClassName{          //class Name
AccessSpecifier:          //public, private or protected
Datamembers;              //Variables/Attributes
MemberFunctions(){}       //Methods to access member functions/Behavior
//////////////////////////////////////////////////
AccessSpecifier:          //public, private or protected
Datamembers;              //Variables/Attributes
MemberFunctions(){}       //Methods to access member functions/Behavior
//////////////////////////////////////////////////
};                                  //end of the class Name
```

ACTIVITY

- Define a class to create a new datatype called Date (Should include day, month and year)
- Define two functions called
  - setDate to initialize values by passing parameters
  - getDate to print the value (Date) on the console
- Create a variable called "today" of data type Date
- Set the value to 13/November/2025
- Display the value on the console

CLASS: DATE

```
1   #include<iostream>
2   using namespace std;
3
4   class date
5   {
6       int day;
7       string month;
8       int year;
9
10      void setDate(int D,string M,int Y){
11          day = D;
12          month = M;
13          year = Y;
14      }
15
16      void getDate(){
17          cout<<"Day:"<<day<<" Month:"<<month<<" Year:"<<year<<endl;
18      }
19  }; // end of class definition
20
```

CLASS AS A DATA TYPE

- Once a class has been declared, it can be used as object/array declarations

Example:

```
Car carObj,                  // object of type Car
    Car carArray[10];        // array of Car objects
      Car *newCar;           // Object pointer
```

The class name becomes the new type specifier.

DATA MEMBERS AND MEMBER FUNCTIONS

Accessing Data members / Member functions

- DOT (.) Operator
  - Example: MyObject.variableX; MyObject.func();
- Refer to a member of a class via a pointer to an object, (->) Operator

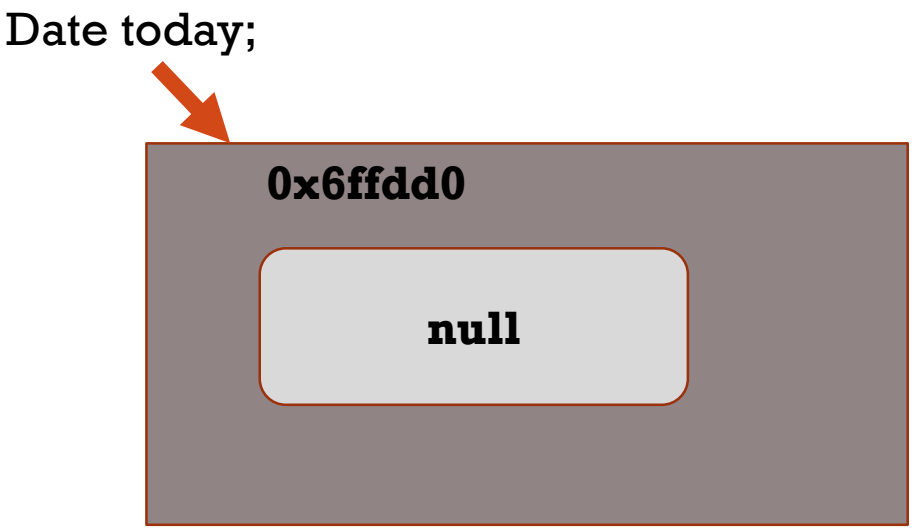
A *pointer* variable, or simply a *pointer*, contains the location, or address in the memory, of a memory cell

ACTIVITY

- Define a class to create a new datatype called Date (Should include day, month and year)
- Define two functions called
  - setDate to initialize values by passing parameters
  - getDate to print the value (Date) on the console
- Create a variable called "today" of data type Date
- Set the value to 13/November/2025
- Display the value on the console
- **Create a pointer variable called ptrDate to assign the value of "today"**
- **Display the value of ptrDate**

ACCESS OPERATOR (->): EXAMPLE

```
int main()
{
  date today;
  today.setDate(5,“December",2024);

            date *ptrDate;
            ptrDate=&today;

            cout<<"CHECK POINTER"<<endl;
            ptrDate->getDate();
}
```



DATA MEMBERS AND MEMBER FUNCTIONS

Accessing Data members / member functions

- DOT (.) Operator
  - Example: MyObject.variableX; MyObject.func();
- Refer to a member of a class via a pointer to an object, (->) Operator
- Accessing a data member depends on the access modifier of the data member.

Defining Member functions in classes

- Inside class definition
- Outside class definition
  - use the scope resolution :: operator along with class name and function name.

```
4  class Date{
5      int day;
6      string month;
7      int year;
8
9      public:
10     void setDate(int,string,int);
11     void getDate();
12 };
13
14 void Date::setDate(int d,string m,int y){
15     day=d;
16     month=m;
17     year=y;
18 }
19
20 void Date::getDate(){
21     cout<<"Day: "<<day<<" Month: "<<month<<" Year: "<<year<<endl;
22 }
```

DEFINING MEMBER FUNCTIONS

- The scope resolution operator (::) can be used to define member functions outside the class definition.
- Member functions defined inside a class definition are automatically treated as inline functions.
- Member functions that contain more than one statement should be defined outside their class definition.
- Members of a class have class scope – data members can be referred to directly inside the member functions from that class.