**Labsheet-3: Constructors & Destructors**

***Objectives***

* Recognize the need for constructor and destructor.
* Explore type of constructors.
* Describe Constructor and Destructor.
* Visit the examples of constructors and destructors.

**Constructor**

**W**hen an object is created all the members of the object are allocated memory spaces. Each object has its individual copy of member variables. However, the data members are not initialized automatically. If left uninitialized these members contain garbage values. Therefor it is important that the data members are initialized to meaning values at the time of object creation.

**Constructor** is a special member function which is used for initializing objects i.e. member variables of an object. It is special because it has the same name as that of the class, it is declared without any return types and is automatically invoked when an object of its class is created.

Syntax rules for creating a constructor:

1. Its name must be the same as that of the class.
2. It is declared with no return type (not even void).
3. It may not be static.
4. It may not be virtual.
5. It should have public or protected access within the class (only in rare circumstances should it be declared private).

**Types of Constructors:**

1. **Default Constructor:**

This constructor doesn’t take any arguments but simply initializes each data member of an object. It is also known as non-parameterized constructor.

1. **Parameterized Constructor:**

This constructor takes arguments and initializes the object’s member variables with the parameters passed to it.

1. **Copy Constructor:**

This constructor is used when newly created object is to be initialized using pre-existing ones. With this kind of constructor all the member variables of the existing object will be copied to the member variables of the newly created object.

**Destructor**

Destructor is another special member function that is called when an object of its class goes out of existence (i.e. when the object’s scope finishes). Its name is again, the same as that of the class (and it begins with a tilde ‘~’ sign). The primary usage of the destructor is to release memory space occupied by the object.

Syntax rule for creating a destructor:

1. Its name must the same as that of the class except that the first character of its name must be a tilde (~).
2. It is declared with no return type (not even a void).
3. It may be not static.
4. It takes no arguments and therefore cannot be overloaded.
5. It should have public access within the class.

The following code illustrate the use of destructor against dynamic objects:

|  |
| --- |
| #include<iostream>  #include<conio.h>  using namespace std;  class Calculator{  public: int \* num1, \* num2;  Calculator(int ident1, int ident2){  num1 = new int; num2 = new int;  \*num1 = ident1; \*num2 = ident2;  }  ~Calculator(){  delete num1, num2;  }  int add(){  return \*num1 + \*num2;  }  };  int main()  {  Calculator my\_obj(2, 3);  int result = my\_obj.add();  cout << result;  getch();  return 0;  } |

Output: **5**

The destructor function is automatically called, without you knowing, when the program no longer needs the object. If you defined a destructor function as in the above code, it will be executed. If you didn’t define a destructor function, C++ supplies you one, which program users uses unknown to you. However, this default destructor will not destroy dynamic objects.

**Assignments:**

1. **Define a class to represent a bank account. Include the following members:**

**Data Members:**

1. **Name of the account holder**
2. **Account Number**
3. **Balance Amount in the Account**

**Member Functions:**

1. **Open the account.**
2. **Deposit and Withdraw money**
3. **Display account information**

**Write a program to test this class for 10 customers. Make use of all three types of constructors (whenever appropriate).**

|  |
| --- |
| #include<iostream>  #include<conio.h>  using namespace std;  class Account{  char name[100];  int accountNo;  double balance;  public:  Account(); //Default constructor  void createAccount(int); //Function Prototype  void deposit();  void withdraw();  void display();  };  Account :: Account(){ balance = 0.0;}  void Account :: createAccount(int num){  accountNo = num;  cout << "Account Holder's name : ";  cin >> name;  cout << "---Account Created--- " << endl;  cout << "Account name : " << name << endl;  cout << "Account number : " << accountNo << endl;  cout << "Total Balance: " << balance << endl;  }  void Account :: deposit(){  double money;  cout <<" Amount to be deposited : ";  cin >> money;  balance += money;  cout<< "\t\t\t---Cash deposited successfully---\n";  }  void Account :: withdraw(){  double money;  cout << " Amount to be withdrawn :: ";  cin >> money;  cout << endl << "\t\t\t---Cash withdrawn successfully---"<< endl;  balance -= money;  }  void Account :: display(){  cout << " NAME: " << name;  cout << "\n ACCOUNT NUMBER: " << accountNo;  cout << "\n Total Amount: " << balance;  }  int main(){  Account A[10];  int choice, count = 0, acc;  do{  cout << "\t\t\t\*\*\*Menu\*\*\*\n";  cout << "\t\t\t1. Create Account.\n";  cout << "\t\t\t2. Deposit.\n";  cout << "\t\t\t3. Withdraw.\n";  cout << "\t\t\t4. Details.\n";  cout << "\t\t\t5. Exit.\n";  cout << "Enter your choice: ";  cin >> choice;  switch(choice){  case 1:  A[count].createAccount(count);  count++;  break;  case 2:  cout << "Enter Account Number: ";  cin >> acc;  if( acc < count)  A[acc].deposit();  else  cout << "Account number NOT FOUND!!" ;  break;  case 3:  cout << "Enter Account Number: ";  cin >> acc;  if( acc < count)  A[acc].withdraw();  else  cout << "Account number NOT FOUND!!" ;  break;  case 4:  cout << " Account Number: ";  cin >> acc;  A[acc].display();  break;  case 5: return 0;  default:  cout << "Choose Valid option. " <<endl;  }  } while(1);  getch();  return 0;  } |

**Possible Output:::**

\*\*\*Menu\*\*\*

1. Create Account.

2. Deposit.

3. Withdraw.

4. Details.

5. Exit.

Enter your choice: **1**

Account Holder's name : **Dhananjaya**

---Account Created---

Account name : Dhananjaya

Account number : 0

Total Balance: 0

\*\*\*Menu\*\*\*

1. Create Account.

2. Deposit.

3. Withdraw.

4. Details.

5. Exit.

Enter your choice: **2**

Enter Account Number: **0**

Amount to be deposited : **2222**

---Cash deposited successfully---

\*\*\*Menu\*\*\*

1. Create Account.

2. Deposit.

3. Withdraw.

4. Details.

5. Exit.

Enter your choice: **3**

Enter Account Number: **0**

Amount to be withdrawn :: **1111**

---Cash withdrawn successfully---

\*\*\*Menu\*\*\*

1. Create Account.

2. Deposit.

3. Withdraw.

4. Details.

5. Exit.

Enter your choice: **4**

Account Number: **0**

NAME: Dhananjaya

ACCOUNT NUMBER: **0**

Total Amount: 1111

\*\*\*Menu\*\*\*

1. Create Account.

2. Deposit.

3. Withdraw.

4. Details.

5. Exit.

Enter your choice: 2

Enter Account Number: 1

Account number NOT FOUND!!

\*\*\*Menu\*\*\*

1. Create Account.

2. Deposit.

3. Withdraw.

4. Details.

5. Exit.

Enter your choice: 5

1. **Imagine a tollbooth at a bridge. Cars passing by the booth are expected to pay a five-rupee toll. Mostly they do, but sometimes a car goes by without paying. The tollbooth keeps track of the number of cars that have gone by and of the total amount of money collected.**

**Model this tollbooth with a class (say TollBooth). The two data members may be a type of insigne dint (to hold the numbers of cars) and a type double to hold the amount of money collected. A constructor may initialize both these to 0. A member function (say payinCar()) increments the car total and adds 5 to the cash total, while another member function (say noPayCar()) increments the car total but adds nothing to the cash total. Finally a member function display() displays the two total.**

**[Hint: Allow user to press a particular key to indicate a paying car and another key to indicate nonpaying car]**

|  |
| --- |
| #include<iostream>  #include<conio.h>  using namespace std;  class TollBooth{  unsigned int total\_cars;  unsigned int total\_amounts;  public:  TollBooth()  {  total\_amounts = 0;  total\_cars = 0;  }  void payingCar();  void noPayingCar();  void display();  };  void TollBooth :: payingCar(){  ++total\_cars;  total\_amounts += 5;  }  void TollBooth :: noPayingCar(){  ++total\_cars;  }  void TollBooth :: display()  {  cout << "\nTotal Cars: " << total\_cars;  cout << "\nTotal Amount: Rs " << total\_amounts;  }  int main()  {  TollBooth Kalanki;  int choice;  do{  cout << "\*\*\*Menu\*\*\*";  cout << "\n1. Add Paying Car";  cout << "\n2. Add Non-Payin Car";  cout << "\n3. Display Detail";  cout << "\n4. Exit\n";  cout << "Enter your choice: ";  cin >> choice;  switch(choice){  case 1: Kalanki.payingCar();  break;  case 2: Kalanki.noPayingCar();  break;  case 3: Kalanki.display();  break;  case 4: return 0;  default: cout << "Wrong Entry" << endl;  }  }while(1);  getch();  } |

**Possible Output**

\*\*\*Menu\*\*\*

1. Add Paying Car

2. Add Non-Payin Car

3. Display Detail

4. Exit

Enter your choice: **1**

\*\*\*Menu\*\*\*

1. Add Paying Car

2. Add Non-Payin Car

3. Display Detail

4. Exit

Enter your choice: **1**

\*\*\*Menu\*\*\*

1. Add Paying Car

2. Add Non-Payin Car

3. Display Detail

4. Exit

Enter your choice: **2**

\*\*\*Menu\*\*\*

1. Add Paying Car

2. Add Non-Payin Car

3. Display Detail

4. Exit

Enter your choice: **1**

\*\*\*Menu\*\*\*

1. Add Paying Car

2. Add Non-Payin Car

3. Display Detail

4. Exit

Enter your choice: **2**

\*\*\*Menu\*\*\*

1. Add Paying Car

2. Add Non-Payin Car

3. Display Detail

4. Exit

Enter your choice: **3**

Total Cars: 5

Total Amount: Rs 15

\*\*\*Menu\*\*\*

1. Add Paying Car

2. Add Non-Payin Car

3. Display Detail

4. Exit

Enter your choice: 4

**Conclusion**

A **constructor** is a member function of a class, having a same name as its class and which is automatically called when the object of the class is created. It is used to initialize the member variables with desire values. Generally, constructor are **public** function unless there is a strong reason not to make it public.

A constructor may have parameters or arguments which can be provided at the time of creation of objects. A constructor which doesn’t take arguments is called **default** **constructor** or **non-parameterized** constructor whereas a constructor which take arguments is call **parameterized** **constructor**. **Copy** **constructor** is called whenever an instance of same type is assigned to another instance of same class. A destructor is a member function that is automatically called when an object is no more required. It is also a member function much like constructor but with opposite properties.

Thus, In this chapter we explore the world of constructor and destructor. We were able to experience the difference between constructor and destructor by evaluating them from examples. We also recognize the different type of constructor. We also cover some problems relating to construct and destructor.