Name : Mukul Kumar

Enrollment number : 201B162

Batch : B5

Subject : OOP

Assignment number : 1

Semester : 2

Q1. Explain following concepts by writing C++ programs for each:

1. Friend function
2. Friend class
3. Static data member and static member function
4. Constant member function
5. Default parameters
6. Array of objects
7. Reference variable
8. Passing object to a function

Answer

1. **Friend Function** Like friend class, a friend function can be given a special grant to access private and protected members.

**Code:**

#include <iostream>

using std::cin;

using std::cout;

using std::endl;

using std::string;

class Student{

string name;

int roll;

public:

void set\_data()

{

cout << "Input student name : ";

cin >> name;

cout << "Input student roll number : ";

cin >> roll;

}

void get\_data()

{

cout << name << endl;

cout << roll << endl;

}

friend void friend\_function(Student&);

};

void friend\_function(Student &st){

st.roll = 10;

}

int main()

{

Student st1;

st1.set\_data();

friend\_function(st1);

st1.get\_data();

return 0;

}

1. **Friend Class** A friend class can access private and protected members of other class in which it is declared as friend. It is sometimes useful to allow a particular class to access private members of other class.

**Code:**

#include <iostream>

using std::cin;

using std::cout;

using std::endl;

using std::string;

class Student{

string name;

int roll;

public:

void set\_data(){

cout<<"Input your name : ";

cin>>name;

cout<<"Input your roll : ";

cin>>roll;

}

friend class friend\_class;

};

class friend\_class{

public:

void display(Student st)

{

cout<<"name : "<<st.name<<endl;

cout<<"roll : "<<st.roll<<endl;

}

};

int main()

{

Student s1;

friend\_class fc;

s1.set\_data();

fc.display(s1);

return 0;

}

1. **Static data member and static member function** The static member function is similar to the normal member function of a class, but the only difference is that it can access only static member (data or functions) declared in the same class. In other words, we cannot access non-static members inside the definition of a static member function.

**Code:**

#include <iostream>

using std::cin;

using std::cout;

using std::endl;

class ABC{

static int x;

public:

static void static\_function();

static void get\_data(){

cout<<x;

}

};

void ABC::static\_function(){

x=10;

}

int main()

{

ABC ab;

ab.static\_function();

ab.get\_data();

return 0;

}

1. **Constant member function** The const member functions are the functions which are declared as constant in the program. The object called by these functions cannot be modified. It is recommended to use const keyword so that accidental changes to object are avoided. A const member function can be called by any type of object.

**Code:**

#include <iostream>

using std::cin;

using std::cout;

using std::endl;

class ABC{

const int x= 10;

public:

const void display(){

cout<<x;

}

};

int main()

{

ABC ac;

ac.display();

return 0;

}

1. **Default parameters** A default argument is a value provided in a function declaration that is automatically assigned by the compiler if the caller of the function doesn’t provide a value for the argument with a default value.

**Code:**

#include <iostream>

using std::cin;

using std::cout;

using std::endl;

class ABC{

int x,y;

public:

void get\_data(int x=0,int y=0){

this->x=x;

this->y=y;

}

void display(){

cout <<x<<endl;

cout <<y<<endl;

}

};

int main()

{

ABC ab;

ab.get\_data(10);

ab.display();

return 0;

}

1. **Array of objects** The array of type class contains the objects of the class as its individual elements. Thus, an array of a class type is also known as an array of objects. An array of objects is declared in the same way as an array of any built-in data type.

**Code:**

#include <iostream>

using std::cin;

using std::cout;

using std::endl;

class ABC

{

int x;

public:

void set\_data()

{

cin >> x;

}

void display()

{

cout << x << endl;

}

};

int main()

{

ABC ab[3];

for (int i = 0; i < 3; i++)

{

ab[i].set\_data();

ab[i].display();

}

return 0;

}

1. **Reference variable** A referencevariable is an alias, that is, another name for an already existing variable. Once a reference is initialized with a variable, either the variable name or the reference name may be used to refer to the variable.

**Code:**

#include <iostream>

using std::cin;

using std::cout;

using std::endl;

class ABC{

int x;

public:

void get\_data()

{

cout<<x;

}

friend void set\_data(ABC &ref)

{

cin>>ref.x;

}

};

int main()

{

ABC ab;

set\_data(ab);

ab.get\_data();

return 0;

}

1. **Passing object to a function** In C++ we can pass class's objects as arguments and also return them from a function the same way we pass and return other variables. No special keyword or header file is required to do so.

**Code:**

#include <iostream>

using std::cin;

using std::cout;

using std::endl;

class ABC

{

public:

int x;

void set\_data()

{

cin >> x;

}

};

void display(const ABC &ab)

{

cout << ab.x << endl;

}

int main()

{

ABC obj1;

obj1.set\_data();

display(obj1);

return 0;

}

Q2. Suppose you have to simulate “**coin tossing**”. For each toss of the coin, the program should print Heads or Tails. Let the program toss the coin **100 times** and count the number of times each side of the coin appears. Print the results. The program should call a separate function flip that takes no arguments and returns 0 for tails and 1 for heads. [Note: If the program realistically simulates the coin tossing, then each side of the coin should appear approximately half the time.]

Code:

#include <iostream>

#include <time.h>

#include <stdlib.h>

using std::cin;

using std::cout;

using std::endl;

bool headtails();

int main()

{

int count = 0;

srand(time(0));

for (int i = 0; i < 100; i++)

{

if (headtails() == 0)

{

cout << "H"<<endl;

count++;

}

else

cout << "T"<<endl;

}

cout << "No of Heads : " << count << endl;

cout << "No of Tails : " << 100 - count << endl;

return 0;

}

bool headtails()

{

bool ht;

ht = 0 + (rand() % (1 - 0 + 1));

return ht;

}



Q3. Suppose you have to develop small software to assist the primary students. Write a program that will help an elementary school student learn multiplication. Use the rand function to produce two positive one-digit integers. The program should then prompt the user with a question, such as

How much is 6 times 7?

The student then inputs the answer. Next, the program checks the student’s answer. If it’s correct, display the message "Very good!" and ask another multiplication question. If the answer is wrong, display the message "No. Please try again." and let the student try the same question repeatedly until the student finally gets it right. A separate function should be used to generate each new question. This function should be called once when the application begins execution and each time the user answers the question correctly.

For making your software more interesting, program should display various comments for each answer as follows:

Possible responses to a correct answer:

**Very good!**

**Excellent!**

**Nice work!**

**Keep up the good work!**

Possible responses to an incorrect answer:

**No. Please try again.**

**Wrong. Try once more.**

**Don't give up!**

**No. Keep trying.**

Use random-number generation to choose a number from 1 to 4 that will be used to select one of the four appropriate responses to each correct or incorrect answer. Use a **switch** statement to issue the responses.

To make you software more sophisticate, system should monitor the student’s performance over a period of time. The decision to begin a new topic is often based on the student’s success with previous topics. System should count the number of correct and incorrect responses typed by the student. After the student types 10 answers, your program should calculate the percentage that is correct. If the percentage is lower than 75%, display "Please ask your teacher for extra help.", then reset the program so another student can try it. If the percentage is 75% or higher, display "Congratulations, you are ready to go to the next level!", then reset the program so another student can try it. System should support various difficulty levels. At a difficulty level of 1, the program should use only single-digit numbers in the problems; at a difficulty level of 2, numbers as large as two digits, and so on.

#include <iostream>

#include <math.h>

#include <time.h>

using std::cin;

using std::cout;

using std::endl;

void question(int, int &, int &);

int answer(int, int);

void correct();

void incorrect();

int main()

{

int count\_true = 0, count\_false = 0, level = 1, total\_count = 0,total=0;

srand(time(0));

while (1)

{

int a, b, userinp;

int &refa = a;

int &refb = b;

question(level, refa, refb);

label1:

cout << "What is " << a << " times " << b << "?" << endl;

cin >> userinp;

++total\_count;

++total;

if(userinp == -1){

return 0;

}

if ((userinp == answer(a, b)) && total\_count <= 11)

{

correct();

count\_true++;

}

else

{

incorrect();

count\_false++;

if (total\_count < 10)

{

goto label1;

}

}

if (total\_count > 10 && count\_true >= 8)

{

cout << "Congratulations, you are ready to go to the next level!\n" << endl;

level++;

total\_count = 0;

count\_true = 0;

}

else if (total\_count > 10 && count\_true < 8)

{

cout << "Please ask your teacher for extra help." << endl;

count\_true = 0, count\_false = 0, total\_count = 0, level = 1;

}

}

return 0;

}

void question(int level, int &a, int &b)

{

a = pow(10, level - 1) + (rand() % ((int)pow(10, level) - (int)pow(10, (level - 1)) + 1));

b = pow(10, level - 1) + (rand() % ((int)pow(10, level) - (int)pow(10, (level - 1)) + 1));

}

int answer(int a, int b)

{

return a \* b;

}

void correct()

{

int a;

a = (1 + (rand() % (4 - 1 + 1)));

switch (a)

{

case 1:

cout << "Very good!" << endl;

break;

case 2:

cout << "Excellent!" << endl;

break;

case 3:

cout << "Nice work!" << endl;

break;

case 4:

cout << "Keep up the good work!" << endl;

break;

}

}

void incorrect()

{

int a;

a = (1 + (rand() % (4 - 1 + 1)));

switch (a)

{

case 1:

cout << "No. Please try again." << endl;

break;

case 2:

cout << "Wrong. Try once more." << endl;

break;

case 3:

cout << "Don't give up!" << endl;

break;

case 4:

cout << "No. Keep trying." << endl;

break;

}

}

