BATCH

RECURSION

C AND C++

PROGRAMMING MASTERCLASS

AWESH ISLAM
BUET, CSE

C++ Class-05

SHAROARE HOSAN EMON
BME , BUET

আমাদের সবগুলো ক্লাস দেখার জন্য ভিজিট করো

https://www.hsccrackers.com/

SCAN ME

Overloading

Constructor Overloading

Reasons:

1)Flexibility

2)To Support Arrays

3)Copy Constructor

Constructor Overloading

To provide option for both initialising and not initialising.

```cpp
#include<iostream>
using namespace std;
class samp{
    int n;
public:
    samp(){
        n = 0;
    }
    samp(int x){
        n = x;
    }
    void set_n(int x){
        n = x;
    }
    int get_n(){
        return n;
    }
};
int main(){
    samp a;
    samp b(10);
    cout<<a.get_n()<<endl;
    cout<<b.get_n()<<endl;
}
```

Constructor Overloading

This creates an opportunity for uninitialised array.

```
#include<iostream>
using namespace std;
class samp{
    int n;
public:
    samp(){
        n = 0;
    }
    samp(int x){
        n = x;
    }
    void set_n(int x){
        n = x;
    }
    int get_n(){
        return n;
    }
};
int main(){
    samp a[5];
    samp b[5] = {1,2,3,4,5};
    for(int i = 0 ; i < 5;i++){
        cout<<"a["<<i<<"].n = "<<a[i].get_n()<<endl;
    }
    for(int i = 0 ; i < 5;i++){
        cout<<"b["<<i<<"].n = "<<b[i].get_n()<<endl;
    }
}
```

Constructor Overloading

Dynamically allocated array is uninitialised.

```
#include<iostream>
using namespace std;
class samp{
    int n;
public:
    samp(){
        n = 0;
    }
    samp(int x){
        n = x;
    }
    void set_n(int x){
        n = x;
    }
    int get_n(){
        return n;
    }
};
int main(){
    samp *p;
    p = new samp[5];
    for(int i = 0;i < 5;i++){
        (p+i)->set_n(i+1);
    }
    for(int i = 0;i < 5;i++){
        cout<<(p+i)->get_n()<<endl;
    }
}
```

Constructor Overloading

Flexibility

```cpp
#include<iostream>
using namespace std;
class date{
    int day;
    int month;
    int year;
public:
    date(int day,int month,int year){
        this->day = day;
        this->month = month;
        this->year = year;
    }
    date(char * str){
        sscanf(str, "%d%*c%d%*c%d",&day,&month,&year);
    }
    void show(){
        cout<<day<<"/"<<month<<"/"<<year<<endl;
    }
};
int main(){
    date a(1,12,2023);
    date b("30/06/2002");
    date c("1-1-2025");
    a.show();
    b.show();
    c.show();
}

```

Copy Constructor

3 Times when copy constructor is called:

1) When an object is used to initialise another in a declaration statement
2) When an object is passed as a parameter to a function
3) When a temporary object is created for use as a return value by a function

```
25  int main(){
26      samp y;
27      samp x = y;
28      func(y);
29      y = func2();
30  }
```

Problem 1

```
1  #include<iostream>
2  #include<cstring>
3  using namespace std;
4  #define size 100
5  class strtype{
6      char *p;
7      int len;
8  public:
9      strtype(char *ptr){
10         cout<<"Constructing..."<<endl;
11         p = new char[size];
12         if(!p){
13             cout<<"Allocation Error"<<endl;
14             exit(1);
15         }
16         if(strlen(ptr) > size){
17             cout<<"String is too big to copy"<<endl;
18         }
19         strcpy(p, ptr);
20         len = strlen(p);          ⚠ Implicit conversion loses integer precision: 'size_t' (aka 'unsigned long') to 'int'
21     }
22     ~strtype(){
23         cout<<"Destructing..."<<endl;
24         delete [] p;              ≡ Thread 1: signal SIGABRT
25     }
26     void show(){
27         cout<<"The string is: "<<p<<"--Length: "<<len<<endl;
28     }
29 };
30 int main(){
31     strtype str("This is a test string");   ⚠ ISO C++11 does not allow conversion from string literal to 'char *'
32     strtype str2 = str;
33 }
34
```

Problem 2

```
30  void func(strtype x){
31
32  }
33  int main(){
34      strtype str("This is a test string");
35      func(str);
36  }
```

Problem 3

```
30  strtype func(){
31      strtype s("String from function");
32      return s;
33  }
34  int main(){
35      strtype str("This is a test");
36      str = func();
37  }
```

Solution: Copy Constructor

```
23      strtype(const strtype &str){
24          cout<<"Copy Constructor..."<<endl;
25          p = new char[size];
26          if(!p){
27              cout<<"Allocation Error"<<endl;
28              exit(1);
29          }
30          strcpy(p, str.p);
31          len = strlen(p);
32      }
```

Default Arguments

```cpp
#include<iostream>
using namespace std;
void f(int a = 0,int b = 0){
    cout<<"A: "<<a<<" B: "<<b<<endl;
}
int main(){
    f();
    f(10);
    f(10,99);
}

```

```
A: 0 B: 0
A: 10 B: 0
A: 10 B: 99
Program ended with exit code: 0
```

Function Overloading

```cpp
#include<iostream>
using namespace std;
void rect_area(double length,double width){
    cout<<"Area is: "<<length*width<<endl;
}
void rect_area(double length){
    cout<<"Area is: "<<length*length<<endl;
}
int main(){
    rect_area(10, 10);
    rect_area(5);
}
```

```
Area is: 100
Area is: 25
Program ended with exit code: 0
```

Ambiguity

```
1  #include<iostream>
2  using namespace std;
3  void f(double x){
4      cout<<"this is from 1st f: "<<x<<endl;
5  }
6  void f(float y){
7      cout<<"this is from 2nd f: "<<y<<endl;
8  }
9  int main(){
10     double x = 10.3;
11     float y = 10.5;
12     f(x);
13     f(y);
14     f(10);
15 }
16
```

Call to 'f' is ambiguous

Ambiguity

```
1  #include<iostream>
2  using namespace std;
3  int func(int a,int b){
4      return a+b;
5  }
6  int func(int a,int &b){
7      return a-b;
8  }
9  int main(){
10     int a = 10;
11     int b = 20;
12     cout<<func(a,b)<<endl;
13 }
14
```

Call to 'func' is ambiguous

Ambiguity

```
1  #include<iostream>
2  using namespace std;
3  int func(int a,int b = 0){
4      return a+b;
5  }
6  int func(int a){
7      return a;
8  }
9  int main(){
10     int a = 10;
11     cout<<func(a)<<endl;
12 }
13
```

Call to 'func' is ambiguous

Operator Overloading

Rules:

1) Doesn’t change the original meaing adds extra meaning
2) Can’t change the precedence
3) Can’t change the number of arguments

Binary Operator +

```cpp
#include<iostream>
using namespace std;
class coord{
    int x,y;
public:
    coord(){

    }
    coord(int x,int y){
        this->x = x;
        this->y = y;
    }
    void show(){
        cout<<"X: "<<x<<" Y: "<<y<<endl;
    }
    coord operator+(coord &obj){
        coord temp;
        temp.x = x + obj.x;
        temp.y = y + obj.y;
        cout<<"(x,y) = "<<"("<<x<<","<<y<<")"<<endl;
        cout<<"(obj.x,obj.y) = "<<"("<<obj.x<<","<<obj.y<<")"<<endl;
        return temp;
    }
};
int main(){
    coord a(10,20),b(20,30);
    a.show();
    (a+b).show();
}
```

Binary Operator +

```
16      coord operator+(coord &obj){
17          coord temp;
18          temp.x = x + obj.x;
19          temp.y = y + obj.y;
20          return temp;
21      }
22      coord operator+(int i){
23          coord temp;
24          temp.x = x + i;
25          temp.y = y + i;
26          return temp;
27      }
28  };
29  int main(){
30      coord a(10,20),b(20,30);
31      a.show();
32      (a+b).show();
33      (a+10).show();
34      (b+9).show();
35  }
```

Binary Operator -

```
28      coord operator-(coord &obj){
29          coord temp;
30          temp.x = x - obj.x;
31          temp.y = y - obj.y;
32          return temp;
33      }
34  };
35  int main(){
36      coord a(10,20),b(20,30);
37      a.show();
38      (a-b).show();
39  }
```

Binary Operator =

```
34      coord operator=(coord &obj){
35          coord temp;
36          temp.x = obj.x;
37          temp.y = obj.y;
38          return temp;
39      }
40
41  };
42  int main(){
43      coord a(10,20),b(20,30);
44      b = a;
45      a.show();
46      b.show();
47  }
```

Logical Operator ==

```
40      bool operator==(coord &obj){
41          return (x == obj.x) && (y == obj.y);
42      }
43  };
44  int main(){
45      coord a(10,20),b(10,20);
46      if(a == b) cout<<"Yes"<<endl;
47      else cout<<"No"<<endl;
48  }
```

Unary Operator ++x

```
43      coord operator++(){
44          coord temp;
45          temp.x = ++x;
46          temp.y = ++y;
47          return temp;
48      }
49  };
50  int main(){
51      coord a(10,20),b(10,20);
52      (++a).show();
53      a.show();
54  }
```

Unary Operator x++

```
43      coord operator++(){
44          coord temp;
45          temp.x = ++x;
46          temp.y = ++y;
47          return temp;
48      }
49      coord operator++(int notused){
50          coord temp;
51          temp.x = ++x;
52          temp.y = ++y;
53          return temp;
54      }
55  };
56  int main(){
57      coord a(10,20),b(10,20);
58      (++a).show();
59      (a++).show();
60      a.show();
61  }
```

Unary Operator -

```
55        coord operator-(){
56            coord temp;
57            temp.x = -x;
58            temp.y = -y;
59            return temp;
60        }
61    };
62    int main(){
63        coord a(10,20),b(10,20);
64        (-a).show();
65    }
```

Friend Operator Functions +

```
55      friend coord operator+(coord &a,coord &b);
56  };
57  coord operator+(coord &a,coord &b){
58      coord temp;
59      temp.x = a.x + b.x;
60      temp.y = a.y + b.y;
61      return temp;
62  }
63  int main(){
64      coord a(10,20),b(10,20);
65      (a+b).show();
66  }
```

Friend Operator Functions +

```
49      friend coord operator+(coord &a,coord &b);
50      friend coord operator+(int a,coord &b);
51      friend coord operator+(coord &b,int a);
52  };
53  coord operator+(coord &a,coord &b){
54      coord temp;
55      temp.x = a.x + b.x;
56      temp.y = a.y + b.y;
57      return temp;
58  }
59  coord operator+(int a,coord &b){
60      coord temp;
61      temp.x = a + b.x;
62      temp.y = a + b.y;
63      return temp;
64  }
65  coord operator+(coord &b,int a){
66      coord temp;
67      temp.x = a + b.x;
68      temp.y = a + b.y;
69      return temp;
70  }
71  int main(){
72      coord a(10,20),b(10,20);
73      (a+b).show();
74      (a+10).show();
75      (10+a).show();
76  }
```

Print Operator <<

```
53      friend ostream &operator<<(ostream &os,coord obj);
54  };
55
56  ostream &operator<<(ostream &os,coord obj){
57      os<<"("<<obj.x<<","<<obj.y<<")"<<endl;
58      return os;
59  }
```

Closer look: Assignment Operators

```
30      void operator=(strtype &obj){
31          cout<<"Assignment Operator..."<<endl;
32          p = new char[size];
33          if(!p){
34              cout<<"Allocation Error"<<endl;
35              exit(1);
36          }
37          strcpy(p, obj.p);
38          len = strlen(p);
39      }
40  };
41  int main(){
42      strtype str("This is a test string");
43      strtype str2("This is a second string");
44      str2 = str;
45  }
```