ওমর আল জাবির (মিশো)

# অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং

C 3 C++ এর মধ্যে পার্থক্য কি ? C হল একটি হাইলেভের প্রোগ্রামিং ল্যাঙ্গুয়েজ এবং C++ হল হাইলেভেল প্রোগ্রামিং ল্যাঙ্গুয়েজসহ অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং ল্যাঙ্গুয়েজ। সহজ উত্তর। কিন্তু যদি প্রশ্ন করি অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং কি? তবে নিশ্চয়ই কপালে ভাঁজ করে কিছুক্ষণ মাথা চুলকোবেন। অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং এর সংজ্ঞা দিতে গেলে বিশাল বক্তৃতা দিতে হবে। যেহেতু কমপিউটার জগৎ জুন ১৯৯৩ সংখ্যায় এ নিয়ে বিস্তারিত আলোচনা হয়েছে তাই আমি পুনরায় এ নিয়ে আলোচনা করব না। এবারে আমরা সরাসরি প্রোগ্রামিং-এ চলে যাব।

অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং এর প্রধান বৈশিষ্ট্য বা ভিত্তি হল class। মূলতঃ OOP মানেই হল class। কেননা OOP যত বৈশিষ্ট্য রয়েছে তা কোন না কোনভাবে ক্লাশ দ্বারা ব্যবহৃত হয়। class কেও একক সংজ্ঞা দিয়ে বোঝান যাবেনা। তবে বলা যায় class হল এমন একটি প্যাকেট (ধরে নিন) যার মধ্যে তার নিজস্ব ব্যাক্তিগত, সংরক্ষিত ও উন্মুক্ত কিছু অংশ রয়েছে এবং এই একটি প্যাকেট থেকে হুবুহ একই রকম ভিন্ন ভিন্ন নামে অসংখ্য প্যাকেট সৃষ্টি করা যায়। এক্ষেত্রে বি প্রথম প্যাকেটটি class এবং সৃষ্ট বাকি প্যাকেটগুলো তার অবজেক্ট। যেমন একটি উদাহরণ দেই–

```
class Animal
    {
            //It's member(s)
    };
Animal Cat, Rat ;
```

এখানে Animal একটি ক্লাশ এবং Cat ও Rat উভয়েই তার অবজেক্ট। Animal ক্লাশের যাই থাকুক না কেন Cat ও Rat উভয়েই তা একই নামে ব্যবহার করতে পারবে। উভয়েই স্বতন্ত্র অবজেক্ট। কারও কোন পরিবর্তনে অপরের পরিবর্তন সাধিত হয় না।

এবার আসি ক্লাশের ফাংশন ও ডাটার কথায়। OOP এর একটি চমৎকার বৈশিষ্ট্য হল ক্লাশের ফাংশন ও ডাটাকে তিনটি বৈশিষ্ট্যে রাখা যায়। Public বা উন্মুক্ত, Private বা ব্যক্তিগত এবং Protected অর্থাৎ সংরক্ষিত। পাবলিক অংশ বাইরের এবং ক্লাশের যেকোন অংশের ফাংশন ও ভেরিয়েবল ব্যবহার করতে পারবে। মনে রাখবেন ক্লাশের সকল ফাংশন ও ভেরিয়েবলকে তার Member বলা হয়। প্রাইভেট ক্লাশের একান্ত ব্যক্তিগত অংশ। এ অংশ কোনভাবেই ক্লাশের বাইরে থেকে ব্যবহার করা যাবে না। প্রোটেক্টেড ক্লাশের সংরক্ষিত অংশ। প্রোটেক্টেড মেম্বারের ব্যবহার একটু জটিল এবং এ নিয়ে বলতে গেলে OOP এর বেশ কিছু অংশ জানতে হবে তাই এ নিয়ে পরে আলোচনা হবে। এবার দেখি কিভাবে ক্লাশের মেম্বারকে এ তিনটি বৈশিষ্ট্যে রাখা যায়।

```
class        Animal
    {
            public :
            int i ;
            private :
            char j ;
            protected :
            long k ;
    };
```

এই ক্লাশে i, j ও k যথাক্রমে public, private ও protected ভেরিয়েবল।

প্রতিটি ক্লাশে একটি ফাংশন (একাধিকও থাকতে পারে) থাকে যা ক্লাশের নামে হয়ে থাকে। যখন ক্লাশ থেকে কোন অবজেক্ট সৃষ্টি হয় তখন তা নিজে নিজেই কল হয়। এধরনের ফাংশনকে Constructor বলে। কন্সট্রাক্টর ক্লাশকে ইনশিয়ালাইজ ক্লাশের জন্য মেমোরী এলোকেট ইত্যাদি প্রারম্ভিক কাজ করে থাকে। যেমন–

```
class animal
    {
            public :
            Animal ( )
            {
             printf("Object has
successfully created. In");
            };
void main ( )
{
                    Animal a;
}
```

যখন a অবজেক্টটি সৃষ্টি হয় তখন Animal ( ) কন্সট্রাক্টরটি নিজে নিজেই কল হয়। ফলস্বরূপ a অবজেক্টটি সৃষ্টি হয়েই দেখাবে যে সে ঠিকমত সৃষ্টি হয়েছে।

কন্সট্রাক্টরকে বিভিন্ন আরগুমেন্ট দেওয়া যায়। মনে রাখবেন কন্সট্রাক্টরকে যদি কোন আরগুমেন্ট না দেওয়া হয় তবে তাকে ডিফল্ট কন্সট্রাক্টর বলে। উপরের উদাহরণে Animal ( ) একটি ডিফল্ট কন্সট্রাক্টর। আরগুমেন্ট দিয়েও কন্সট্রাক্টরকে ডিফল্ট করা যায়।

যেমন–

```
class Animal
    {
            public :
            Animal (int X = 0)
            {
            printf ("x = %d" , x ) ;
            };
    };
Animal Cat, Rat (10) ;
```

এখানে Cat এ ডিফল্ট কন্সট্রাক্টর এবং Rat এ Animal (10) এই কন্সট্রাক্টরটি কল হয়েছে। এভাবে দেওয়ার সুবিধা হল আপনি একইসাথে ডিফল্ট ও আরগুমেন্টসহ কন্সট্রাক্টর পাচ্ছেন।

অবজেক্ট যে আরগুমেন্ট দেওয়া হয় তা অনুযায়ী কন্সট্রাক্টর কল হয়। যেমন –

```
Animal Cat, Rat (10), Bat ('a') ;
```

এখানে Cat এর ডিফল্ট, Rat এর Animal (int) এবং Bat এর Animal (char) কন্সট্রাক্টর কল হয়েছে। কন্সট্রাক্টরকে যত খুশি আরগুরমেন্ট দেওয়া যায় তবে মনে রাখবেন কন্সট্রাক্টর এর কোন return type এবং return value বা string কিছুই থাকবে না। যেমন–

```
int    Animal (int x = 0)
{
       return x;
}
```

এখানে int Animal এবং return i উভয়েই ভুল। এমনকি void-ও দেওয়া যাবে না।

কন্সট্রাক্টর এর মত ক্লাশে একটি মাত্র ডেস্ট্রাক্টর থাকে যার নামও ক্লাশের নামে হয়ে থাকে। তবে নামের আগে একটি – চিহ্ন দিতে হয়। ডেস্ট্রাক্টরের কোন রিটার্নটাইপ এবং আরগুমেন্ট থাকবেনা। যেমন–

```
class Animal
    {
            public :
            Animal ( ) { }        / /
constructor
    - Animal ( ) { }// destructor
    };
```

ডেস্ট্রাক্টর যখন অবজেক্টের প্রয়োজন শেষ হয়ে যায় বা মুছে ফেলা হয় তখন কল হয়। যেমন–

```
void main ( )
{
        Animal Cat ;
        i f ( kbhit ( ) )
        {
                Animal Rat ;
                 // whatever
        }
}
```

এক্ষেত্রে main শেষে Cat এর এবং if শেষে Rat এর ডেস্ট্রাক্টর কল হয়। ডেস্ট্রাক্টরে ক্লাশের শেষের কিছু কাজ যেমন মেমোরী ফ্রী ইত্যাদি করা হয়ে থাকে।

এবারে ক্লাশের ফাংশন স্বন্ধে আলোচনা করা যাক। ক্লাশের ফাংশন দুভাবে তৈরী করা যায়। ভিতরে এবং ক্লাশের বাইরে। যদি ভিতরে ডিক্লেয়ার করতে চান তবে সাধারণ ফাংশন যেভাবে করা হয় সেভাবেই ডিফাইন করতে পারেন। প্রোটোটাইপের প্রয়োজন নেই। যদি ক্লাশের বাইরে ডিফাইন করতে চান তবে সাধারণ নিয়মে ক্লাশের ভিতরে প্রোটোটাইপ লিখুন। এরপর একটু ঝামেলা পোহাতে আসুন। ক্লাশের বাইরে ফাংশন ডিক্লেয়ার করার যে জটিল স্ট্রাকচার রয়েছে তা হল–

```
<রিটার্ন টাইপ > < স্পেস > < ক্লাশের নাম > :
: < ফাংশানের নাম > (< আরগুমেন্ট>)
      {
              < ফাংশনের কার্যকলাপ >
      }
```

একটা উদাহরণ দেই–

```
int Animal :: Hello (int i)
              {
              printf ("I am an animal. In");
                      return i ;
              }
```

বাইরে ফাংশন ডিক্লেয়ার করার নিয়মটি খুব সাবধানে মেনে চলবেন। একটু নড়চড় হলেই সমস্ত ফাংশনে ভুল দেখা দিবে। ফাংশনকে কল করা বেশ সহজ।

struct এর অন্তর্ভুক্ত ভেরিয়েবলকে যেভাবে ব্যবহার করা হয় ফাংশানকেও অনুরূপভাবে কল করা যায়। যেমন –

```
void main ( )
      {
              Animal cat ;
              Animal *Rat ;
              cat. Hello (10) ;
              Rat-> Hello (20) ; //
because it is dlynamic.
      }
```

friend নামে OOP-এর এক অনন্য অসাধারণ বৈশিষ্ট্য রয়েছে। এর মাধ্যমে OOP এর অনেক অসুবিধা যেমনি দূর হয়েছে তেমনি অসংখ্য নতুন সুবিধা OOP উপস্থাপন করেছে। friend এর কর্মক্ষেত্র অনেক বড় এবং এর ব্যবহার ব্যাপক হলেও এর ব্যবহারকে প্রধানত তিনভাগে ভাগ করা যায়।

(১) ক্লাশে

(২) ফাংশনে এবং

(৩) ভেরিয়েবলে।

ক্লাশে friend এক অসাধারণ বৈশিষ্ট্য উপস্থাপন করেছে। আপনি এখন ইচ্ছা করলেই দুটো ক্লাশকে একটি ক্লাশে পরিণত করতে পারেন আবার একই সাথে দুটি ক্লাশকেও আলাদাভাবে ব্যবহার করতে পারেন। যেমন দেখুন–

```
        class Man ; // Declare
        class Human
              {
                    public :
                    friend class Man;
                    // Human class mem-
        ber
              };
        class Man
              {
                      // whatever
              };
```

Human ক্লাশটি Man ক্লাশকে ফ্রেণ্ড হিসেবে নিয়েছে ফলে Human ইচ্ছা করলেই Man এর ব্যক্তিগত সংরক্ষিত ও উন্মুক্ত অংশ ব্যবহার করতে পারবে। অর্থাৎ Man ক্লাশের সমস্ত অংশ Human ক্লাশের অন্তর্ভুক্ত হয়ে গেছে।

এখানে একটা লক্ষ্যণীয় বিষয় আছে তা হল আপনি যেই ক্লাশকে friend হিসেবে ব্যবহার করবেন তাকে অবশ্যই আগে থেকে declare করে রাখতে হবে। প্রথম লাইনটি বলে যে– "হ্যা, Man ক্লাশ উপস্থিত আছে"। অনেকটা স্কুলের present sir এর মত।

C++ নিজেই কিছু সুবিধা দেয়। যেমন আপনি ক্লাশকে এভাবে ডিক্লেয়ার করতে পারেন।

```
    Animal a = 50;
```

এই প্রক্রিয়াকে automatic conversion বলে। এক্ষেত্রে কি হয় দেখা যাক। a = 50 এর জায়গায় Animal (50) করা হয়। আবার দেখা যাক–

```
    Animal b = 50 + 50 ;
```

এক্ষেত্রে Animal (50 + 50) কন্সট্রাক্টরটি কল হয়। আপনাদেরকে এ নিয়ে বলা প্রয়োজন ছিল তাই এটা বর্ণনা করা হল। এর যা নিয়ে আলোচনা করা হবে তা বুঝতে হলে আপনাদের আগে এই অটোমেটিক কনভারশন বুঝতে হবে।

OOP এর আরেকটি অত্যন্ত চমকপ্রদ বৈশিষ্ট্য নিয়ে না বললে অসম্পূর্ণ থেকে যাবে। তা হল operator। operator OOP এর এক অনন্য বৈশিষ্ট্য। এর ব্যবহার প্রোগ্রামিং-এ এক নতুন ধারার সৃষ্টি করেছে। আপনি কি কখনও C-এ কোন কোড এভাবে লিখতে পারবেন?

```
    int i ;
    void Hello ( ) ;
    Hello ( ) + i;
```

অস্বাভাবিক এবং অসম্ভব। কিন্তু C++ এই অসম্ভবকে সম্ভব করেছে operator নামের ছোট অথচ এক বিশাল সুবিধার ভাণ্ডার দিয়ে। আপনি ইচ্ছা করলেই C++ এ এভাবে লিখতে পারবেন।

```
    class Animal ;
    int i = 10 ;
    Animal + i ;
```

অথবা,

```
    Animal + = " Hi! " ;
```

অবাককর নয় কি? অন্তত C ব্যবহারকারীদের কাছে নিশ্চয়ই পরিষ্কার হয়ে গেছে যে এর দ্বারা কি বিশাল প্রোগ্রামিং সুবিধা পাওয়া সম্ভব। তবে আসুন দেখি কিভাবে এগুলো সম্ভব করা যায়।

operator হচ্ছে +, –, *, /, = ইত্যাদি। ক্লাশের সাথে এগুলো ব্যবহার করা যায় operator নামের একটি keyword এর মাধ্যমে। একটি উদাহরণ দেই–

```
        class Integer
              {
                        public :
                        int i ;
                        Integer ( int x = 0)
                         {
                                 i = x ;
                         }
                        Integer operator + (
        Integer x)
                        {
                         i + = x.i ;
                        }
              };
        void main (void)
              }
                        Integvar i1 = 10;
                        Integer i2 = i1 + 5 ;
                        Integer i3 = i1 + i2;
              }
```

উপরে উদাহরণে i1=10 লাইনটিতে অটোমেটিক কনভারশন হয়েছে। i1 + 5 লাইনটিতে 5 এর বেলায় 5 একটি অবজেক্টে পরিণত হয়েছে। অর্থাৎ লাইনটি এরকম হয়ে গেছে।

```
    i1 + x (5);
```

i1 + i2 এই লাইনটিতে operator সহজেই তার কাংখিত আরগুমেন্টটি পেয়ে গেছে। অর্থাৎ যেহেতু operator এর আরগুমেন্ট হিসেবে একটি অবজেক্ট চাওয়া হয়েছে তাই i2 সরাসরি ব্যবহৃত হয়েছে। আপনি যদি এভাবে লিখতেন–

```
            Integer operator + (int X)
              {
                        i + = x ;
              }
```

তবে i1 + 5 লাইনটিই কেবল কাজ করবে। পরেরটি কাজ করবেনা। এবার আরেকটি উদাহরণ দেখি। উপরের ক্লাশটি এভাবে কি ব্যবহার করা যাবে?

```
    Integer i5 = 5 + i1 ;
```

এক্ষেত্রে দেখা যায় 5 একটি ক্লাশ এবং + তার অপারেটর। ফলস্বরূপ error। তাছাড়া মনে রাখবেন অটোমেটিক কনভারশন কেবল ক্লাশের কোন অপারেটরের ডান দিকে হতে পারে। অর্থাৎ 5 + i1 যদি হয়ও তবুও 5 কে C++ কোন অবজেক্টে পরিণত করতে পারবেনা। কেননা ইহা + এর বাদিকে রয়েছে। এক্ষেত্রে উপায় হচ্ছে friend ব্যবহার।

```
        class Animal
              {
                        public :
                        int i;
                        Animal (int x = 0)
                        {
                         i = x;
                        }
                        friend Animal opera-
        tor + (Animal aa1, Animal aa2) ;
              };
        void main ( )
              {
                        Animal  a1 = 10;
                        Animal  a2 = a1 + 5 ;
        // okay!
                        Animal  a3 = 5 + a2 ;
        // Now it is fine !
              }
        Animal operator + (Animal aa1,
        Animal aa2)
              {
                        Animal aa = aa1.i +
        aa2.I ;
                        return aa;
              }
```

এবারে 5+ a2 কাজ করবে। কেননা friend ব্যবহার করাতে অটোমেটিক কনভারশন বাঁ দিকেও করতে পারছে।

এখানে aa1 5 নিচ্ছে এবং aa2, a2 কে নিচ্ছে। আপনি কোন অবজেক্টকে কোন টাইপ হিসেবে নিতে পারেন। এভাবে লিখুন–

```
        Animal operator float ( )
              {
                        float f ;
                        f = (float) i ;
                        return f ;
              }
        void main ( )
              {
                        Animal x (10) ;
                        float f = float (x) ;
              }
```

শেষের লাইনটিতে x ক্লাশের float রূপটি নেওয়া হয়েছে। সবশেষে ব্যবহার করা যায় এমন একটি ক্লাশ দিয়ে যাচ্ছি।

```
        class Animal
              {
                        private :
                        char *Name ;
                        public :
                        Animal (char *n =
        NULL)
                        {
                                       s t r c a t
        (Name, n) ;
                        }
                        void whoAmI (void) ;
                        void MyNewName
        (char *n = NULL) ;
                        Animal operator =
        (char *n)
              {

        memset (name, O, strlen
        (Name));

        memcpy (Name, n, strlen (n));
              }
                        Animal operator +
        (char *n)

        strcat (Name, n);
                                    }
              };
        void Animal :: Who AmI ( )
              {
                        printf ("My name is %
        s \n", Name)'
              }
        void Animal :: my New Name
        (char *n = NULL)
              {
                        memcpy (Name, n,
        strlen (n));
                        printf ("My new name
        : %s \n", Name);
              }
        void main ( )
              {
                        Animal Cat ("Cat");
                        Animal Rat = " Rat" ;
                        Animal Bat = "Bat " ;
                                     Bat  +  "
        Man " ;

        Bat.WhoAmI ( ) ;
                        Rat.My New Name
        ("Why cats kill us (SOB!SOB!) " );
              }
```

আগামী পর্বে OOP যে সব বৈশিষ্ট্যের জন্য বিখ্যাত তা নিয়ে আলোচনা করার আশা রাখছি। (চলবে)

কাগজের দুষ্প্রাপ্যতা জন্য এ সংখ্যা কমপিউটার জগৎ প্রকাশে অপ্রত্যাশিত বিলম্বের জন্য আমরা আন্তরিকভাবে দুঃখিত।

স.ক.জ.