ওমর আল জাবির (মিশো)

অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং

(পূর্ব প্রকাশিতের পর)

গত পর্বে OOP এর প্রাথমিক কিছু বৈশিষ্ট্য নিয়ে আলোচনা করা হয়েছিল। ওগুলো ব্যবহার করে আপনি OOP-এর অনেক সুবিধা পেতে পারেন। তবে OOP-এর প্রধান যে দুটি বৈশিষ্ট্য রয়েছে যার জন্য এটি "বিশ্ব বিখ্যাতঃ তা হল Inheritance এবং Polymorphism। এ দুটি বৈশিষ্ট্য ব্যবহার করে OOP এর পুরো শক্তিকে আপনি কাজে লাগাতে পারবেন। তবে ইনহেরিটেন্স ও পলিমরফিজমে যাওয়ার আগে আরও কিছু জানানোর বিষয় প্রয়োজন মনে করছি। আসুন আগে সেগুলো জানা যাক।

C++-এ this নামে একটি ক্লাশ পয়েন্টার রয়েছে যার ব্যবহার অনেক ব্যাপক এবং এটা টেমপোরারী অবজেক্ট এলোকেশন ও ডিফাইনের সময় এবং ঝামেলা দুইই বাঁচিয়ে দেয়। this যেই ক্লাশে যখন ব্যবহৃত হয় তখন সেই ক্লাশটিকে পয়েন্ট করে। অর্থাৎ আপনি যদি * this —> i;

লিখেন তবে যে ক্লাশে লিখেছেন সেই ক্লাশের i নামক ভেরিয়েবলকে পয়েন্ট করবে। আবার যদি এভাবে লিখেন– return *this; এবং ধরি ক্লাশটির নাম Animal এবং এর name ও i নামে দুটি মেম্বার রয়েছে তবে C++ লাইনটিকে পড়বে অনেকটা এভাবে।

```
Animal *temp;
temp—>i = i;
temp—>name=strdup (name);
return *temp;
```

একটু ভেবে দেখুনতো কত বড় সুযোগ এবং সুবিধা আপনি this থেকে পাচ্ছেন।

এবার আসুন ইনহেরিটেন্স নিয়ে জানা যাক।

ইনহেরিটেন্স এর সংজ্ঞা অনেকটা এরকম– একটি ক্লাশ এক বা একাধিক ক্লাশকে ইনহেরিট করে এবং ক্লাশগুলোর অনুমতিপ্রাপ্ত সব মেম্বার সে ব্যবহার করতে পারে বা তার নিজের হয়ে যায় এবং নিষিদ্ধ মেম্বার তার পক্ষে ব্যবহার করা সম্পূর্ণ নিষিদ্ধ, এককথায় অসম্ভব। ফলে ক্লাশের গোপনীয়তা রক্ষা পায় সে সাথে ক্লাশের নেটওয়ার্ক বা যোগাযোগ সৃষ্টি করা যায়। আরেকটি কথা আপনি যদি পলিমরফিজম ব্যবহার করতে চান তবে আপনাকে অবশ্যই প্রথমে ইনহেরিটেন্স সম্বন্ধে জানতে হবে। যেই ক্লাশ অন্যান্য ক্লাশকে ইনহেরিট করে তাকে Derived ক্লাশ বলে। যাকে ইনহেরিট করা হয় তাকে Base ক্লাশ বলে। ইনহেরিট করা হয় এভাবে–

```
class BaseClass

{.
class Derived Class: public BaseClass
```

ডিরাইভড ক্লাশ কোন বেজ ক্লাশকে দুভাবে ইনহেরিট করতে পারে। পাবলিক ও প্রাইভেটভাবে। পাবলিক ভাবে ইনহেরিট করলে বেজ ক্লাশের সকল পাবলিক মেম্বার ডিরাইভড ক্লাশের পাবলিক মেম্বারে পরিণত হয়। প্রাইভেটভাবে ইনহেরিট করলে বেজ

হয়ে যায়। বেজ ক্লাশের প্রাইভেট মেম্বার বেজ এর একান্ত ব্যক্তিগত অংশ। ইহা কোনভাবেই ডিরাইভড ক্লাশে ব্যবহার সম্ভব নয়।

গত পর্বে protected মেম্বার সম্পর্কে বলা হয়নি। এবার এর ব্যবহার বলছি। ধরুন আপনি বেজ ক্লাশের কোন মেম্বারকে বাইরে থেকে ব্যবহার করতে দিতে চাচ্ছেন না অথচ ডিরাইভড ক্লাশকে ব্যবহার করতে দিবেন। এক্ষেত্রে মেম্বারগুলো প্রটেক্টেড করে ফেলুন। প্রটেক্টেড করার ফলে মেম্বারগুলো ক্লাশে প্রাইভেটের মত নিষিদ্ধ থাকে অথচ ডিরাইভ করলে ডিরাইভড ক্লাশে ইহা পাবলিকের মত ব্যবহৃত হয়। একটা উদাহরণ দেই–

```
class Base1
{
public:
int B1;
};
class Base2
{
protected:
int B2;
};
class Derived : public Base1,
public Base2
{
public:
Derived (int i = o)
{
B1 = i;
B2 = i; // no problem
}
};
main ()
{
Derived d (50);
Base2 b;
b.B2 = 20; // Error !!
d.B1 = 10;
}
```

ব্যাপারটা কি দাড়াল দেখা যাক। d অবজেক্টে B2 পাবলিক মেম্বার। অথচ b অবজেক্টের B2 প্রাইভেট। ফলে ইহা ব্যবহার করলে Error আসে। অথচ Derived ক্লাশের যে কোন মেম্বার B2 ব্যবহার করতে পারবে।

আসুন এবার "বিশ্ব বিখ্যাত Polymorphism" প্রসঙ্গে জানা যাক। পলিমরফিজম এর শাব্দিক অর্থ বহুরূপতা। C++ এ এর অর্থ দাঁড়ায়– একটি ক্লাশ থাকে রূপ উপস্থাপক এবং অন্যান্য ক্লাশ তার সাথে যুক্ত হয়ে তার রূপ উপস্থাপন করে। যেমন ধরুন Cow, Goat, Sheep ইত্যাদি ক্লাশকে এককভাবে ব্যবাহার করতে পারেন। এক্ষেত্রে আপনি যদি তাদের লিষ্ট তৈরী করতে চান তবে তাদের জন্য আলাদা করে মেমোরী এ্যালোকেট, ফাংশন তৈরী, আলাদাভাবে সংরক্ষণ ইত্যাদি ঝামেলা করতে হবে। ফলে তাদের অস্তিত্ব সম্পূর্ণ স্বতন্ত্র হয়ে দাঁড়ায়। এর

cat) ব্যবহার করতে যাবেন তখন আপনাকে আবার তার জন্য মেমোরী এলোকেটসহ রক্ষণাবেক্ষণ ও ব্যবহারের জন্য ফাংশন তৈরী করতে হবে। এক্ষেত্রে OOP আপনার সেবায় তার অনন্য বৈশিষ্ট্য Polymorphism নিয়ে উপস্থিত। পলিমরফিজমের প্রয়োগ করতে হলে আপনাকে একটি নেটওয়ার্ক সৃষ্টি করতে হবে। ধরুন প্রধান ক্লাশটির নাম Animal। আপনি যেই ক্লাশগুলো এর সাথে ব্যবহার করবেন তা হল Cow ও Goat। এবারে আসুন নেটওয়ার্কটি তৈরী করি।

```
# include <stdio.h>
# include <stdlib.h>
# include <string.h>
class Animal
{
protected:
char * Name;
public:
Animal () { Name = NULL;}
Animal (char * n)
{
Name = strdup(n);
}
virtual void whoAmI ()
{
printf("An animal.");
}
~Animal() { delete Name;}
};
class Cow : public Animal
{
public:
Cow() : Animal () {}
Cow(Char *n)
: Animal (n) { }
virtual void whoAmI ()
{
printf("I'm a cow named%s. \n",
Name);
}
};
class Goat : public Animal
{
public:
Goat () : Animal () {}
Goat (char * n)
: Animal (n) {}
virtual void whoAmI ()
{
printf ("I'm a goat named %s. \n",
Name);
}
};
class Database
{
private:
```

```
Animal * DataB;
public:
DataBase (unsigned int max)
{
mAni = max;
nAni=0;
DataB = new Animal*
[mAni];
for (int i = 0;
i<mAni; i++)
{
DataB [i]=NULL;
}
}
unsigned int Accept
(Animal *a)
{
if(nAni == mAni)
return 0;
nAni++;
int i=0;
While(DataB [i]
i++;
DataB[i]=a;
return i+1;
}
Animal * Release
(unsigned int pen)
{
if(pen>mAni)
return NULL;
pen - - ;
if(DataB [pen])
{
Animal *temp=
DataB [pen];
DataB [pen]=NULL;
nAni - - ;
return temp;
}
else
return NULL;
}
~Database ()
{
delete DataB;
}
void List Animal ()
{
if (nAni>o)
for(int i = o;
i<mAni;i++)
if(DataB [i])
{
printf ("The Animal)
in pen %d says: ",i);
DataB[i]->whoAmI ();
}
}
}://end of class
Cow c(Milky");
Goat g("Non milky");
Cow Fc("Free");
main ()
{
Database d(3);
unsigned int Cp =
d.Accept(c);
unsigned int Gp=
d.Accept(g);
d.List Animal ();
d.Release (Cp);
d.Release (Gp);
Fc.whoAmI();
}
```

এখানে একটা লক্ষণীয় বিষয় রয়েছে। virtual নামে নতুন একটি keyword ক্লাশগুলোর who AmI নামক ফাংশানগুলোর আগে দেয়া হয়েছে। virtual এর মাধ্যমে কোন ক্লাশে একই মেম্বার একাধিকবার এসে পড়লে পৃথক করা হয়।

এখানে এর ব্যবহারের কারণ হল Animal ক্লাশে যেমন whoAmI ফাংশানটি রয়েছে তেমনি Cow এবং Goat ক্লাশেও whoAmI রয়েছে। এদের লিংক করে দেয়ার ফলে একই ক্লাশে একাধিক whoAmI এসে পড়ছে। virtual এখানে একাধিক whoAmI থেকে যেটি সবচেয়ে আগে পড়ছে সেটিকেই রাখবে। লিংক করা whoAmI এ যাচ্ছেনা। virtual এর প্রয়োগ ক্লাশের বেলায়ও রয়েছে। একাধিক ক্লাশ যখন একটি ক্লাশকে ইনহেরিট করে এবং সেই ক্লাশগুলোকে যখন অন্য কোন ক্লাশ ইনহেরিট করে তখন প্রথমোক্ত ইনহেরিট করা ক্লাশটি দুবার এসে পড়ে। এক্ষেত্রে virtual -এর প্রয়োগ করা হয়। যেমন–

```
class a
{
};
class b : virtual public a
{
};
class C : virtual
public a
{
};
class d :  public b,
           public c
{
};
```

ক্লাশ d-এ ক্লাশ a একবারই আসে এবং b ও c, a-কে একবারই পায়। অর্থাৎ b ও c একই a-কে ব্যবহার করে তবে দুভাবে। b ও c এর কোন পরিবর্তনে অপরের a-এর কোন পরিবর্তন হয় না।

পলিমরফিজমে ক্লাশ যোগ করা খুব সহজ। যেমন-cat নামের একটি ক্লাশ আপনি Animal এর সাথে সংযোগ করতে যাচ্ছেন। এভাবে লিখুন–

```
class Cat : public Animal
{
public :
Cat( ) : Animal ( ) { }
Cat (char * n)
: Animal (n) { }
virtual void whoAmI ( )
{
printf("I'm a cat
named %s.\n", Name);
}
};
```

এর ফলে Cat ক্লাশটি Animal-এর "পলিমরফিক বন্ধু"-এ পরিণত হল।

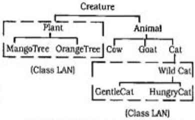
এবার ধরুন আপনি এমন কোন ক্লাশ তৈরী করতে চাচ্ছেন যা Animal এর পলিমরফিক কোন ক্লাশের পলিমরফিক হবে।

যেমন Wild Cat ক্লাশকে আপনি Cat থেকে আলাদা করতে চাচ্ছেন অথচ ইহা Cat ক্লাশেরই অন্তর্ভূক্ত থাকবে। এভাবে লিখুন

```
class Wild Cat : public cat
{
public :
WildCat () : Cat ( ) { }
WildCat (Char * n)
: Cat(n) { }
// Some codes
};
```

পলিমরফিজমকে LAN এর সঙ্গে তুলনা করা যায়। এখানে প্রধান একটি ক্লাশ থাকে যা সমস্ত ক্লাশকে ব্যবহার করতে পারে। এর অনেক শাখা ক্লাশ থাকে যা LAN এর তথ্য ধারন করে বা রূপ উপস্থাপন করে।

পলিমরফিজমকে ব্যবহার করে আপনি ক্লাশের "ইন্টারনেটঃ তৈরী করতে পারেন। এক্ষেত্রে কয়েকটি ক্লাশের ট্রিকে একটি বা একাধিক প্রধান ক্লাশের সাথে যুক্ত করে সমস্ত ক্লাশের মধ্যে যোগাযোগ স্থাপন করতে পারেন। যেমন একটি চার্ট দেখুন



পলিমরফিজমের ব্যবহারের দুটো দিক রয়েছে। প্রথমত, ক্লাশের সাথে ক্লাশের যোগাযোগ তৈরী করে বিশাল ট্রি তৈরী করতে পারেন এবং খুব সহজে রক্ষণাবেক্ষণ করতে পারেন। দ্বিতীয়ত, পলিমরফিজমের মাধ্যমে প্রোগ্রামকে খুব সহজে, কম সময়ে বর্ধিত করতে পারেন। এর মাধ্যমে ক্লাশের কোন বিশাল নেটওয়ার্কের অন্যান্য অংশে কোন প্রভাব না ফেলে যে কোন অংশ ধ্বংস করে দিতে পারেন তেমনি সংযোগও করতে পারেন। এমনিভাবে পলিমরফিজমের কোন জগতে অন্যান্য অংশকে পরিবর্তন না করেও এর যে কোন ক্ষুদ্র অংশ পরিবর্তন করা যায়। তবে একটা কথা বলা দরকার। ক্লাশের ফাংশনকে virtual করে ফেলুন। বলা তো যায়না, একই রকম ফাংশান অন্য কোন ক্লাশে থেকেও যেতে পারে। তখন পলিমরফিজমের বিশাল ক্লাশের জগতে কোথায় খুঁজে মরবেন?

পাঠকদের প্রতি

কমপিউটার বিষয়ক আপনার যেকোন লেখা, চমকপ্রদ অভিজ্ঞতা, আইডিয়া, সফটওয়্যার টিপস্ লিখে পাঠালে আমরা তা কমপিউটার জগৎ-এ প্রকাশ করতে পারলে আনন্দিত হবো। ছাপানো লেখার জন্য লেখকদের যথাযথ সম্মানী দেয়া হয়।

লেখা পাঠানোর ঠিকানা-
সম্পাদক,
মাসিক কমপিউটার জগৎ
১৪৬/১, আজিমপুর রোড, ঢাকা-১২০৫