ਆਬਜੈੱਕਟ ਓਰੀਐਂਟਡ ਪ੍ਰੋਗਰਾਮਿੰਗ ਇੰਨ ਸੀ++

(ਬਾਰ੍ਹਵੀਂ ਸ਼੍ਰੇਣੀ ਲਈ ਪਾਠ-ਪੁਸਤਕ)

ਪੰਜਾਬ ਸਕੂਲ ਸਿੱਖਿਆ ਬੋਰਡ

ਸਾਹਿਬਜ਼ਾਦਾ ਅਜੀਤ ਸਿੰਘ ਨਗਰ

ਸੰਪਾਦਕੀ ਕਮੇਟੀ

ਸ੍ਰੀਮਤੀ ਪੂਜਾ ਅਰੋੜਾ, ਸਰਕਾਰੀ ਸੀਨੀਅਰ ਸਕੈਡਰੀ ਸਕੂਲ, ਸਹੌੜਾ (ਐਸ.ਏ.ਐਸ.ਨਗਰ)

ਸ੍ਰੀਮਤੀ ਸੁਖਵਿੰਦਰ ਕੌਰ, ਸਰਕਾਰੀ ਸੀਨੀਅਰ ਸਕੈਡਰੀ ਸਕੂਲ, ਸਹੌੜਾ (ਐਸ.ਏ.ਐਸ.ਨਗਰ)

ਪੁਨਰ ਮੁਲਾਂਕਣ ਅਤੇ ਤਸਦੀਕ ਕਰਤਾ

ਸ੍ਰੀ ਗਗਨਦੀਪ ਸਿੰਘ, ਸਰਕਾਰੀ ਮਾਡਲ ਸੀਨੀਅਰ ਸਕੈਡਰੀ ਸਕੂਲ, 3 ਬੀ 1, ਐਸ.ਏ.ਐਸ.ਨਗਰ

ਚੇਤਾਵਨੀ

1. ਕੋਈ ਵੀ ਏਜੰਸੀ-ਹੋਲਡਰ ਵਾਧੂ ਪੈਸੇ ਵਸੂਲਣ ਦੇ ਮੰਤਵ ਨਾਲ ਪਾਠ-ਪੁਸਤਕਾਂ ਦੇ ਜਿਲਦ-ਸਾਜੀ ਨਹੀਂ ਕਰ ਸਕਦਾ। (ਏਜੰਸੀ-ਹੋਲਡਰਾਂ ਨਾਲ ਹੋਏ ਸਮਝੌਤੇ ਦੀ ਧਾਰਾ ਨੰ.7 ਅਨੁਸਾਰ)

2. ਪੰਜਾਬ ਸਕੂਲ ਸਿੱਖਿਆ ਬੋਰਡ ਦੁਆਰਾ ਛਪਾਈਆਂ ਅਤੇ ਪ੍ਰਕਾਸ਼ਿਤ ਪਾਠ-ਪੁਸਤਕਾਂ ਦੇ ਜਾਅਲੀ ਨਕਲੀ ਪ੍ਰਕਾਸ਼ਨਾਂ (ਪਾਠ-ਪੁਸਤਕਾਂ) ਦੀ ਛਪਾਈ, ਪ੍ਰਕਾਸ਼ਨ, ਸਟਾਕ ਕਰਨਾ, ਜਮ੍ਹਾਂ ਕਰਨਾ, ਜਮ੍ਹਾਂ-ਖੋਰੀ ਜਾਂ ਵਿਕਰੀ ਆਦਿ ਕਰਨਾ ਭਾਰਤੀ ਦੰਡ ਪ੍ਰਣਾਲੀ ਦੇ ਅੰਤਰਗਤ ਫੌਜਦਾਰੀ ਜੁਰਮ ਹੈ।

ਸਕੱਤਰ, ਪੰਜਾਬ ਸਕੂਲ ਸਿੱਖਿਆ ਬੋਰਡ, ਵਿੱਦਿਆ ਭਵਨ, ਫੇਜ-8 ਸਾਹਿਬਜਾਦਾ ਅਜੀਤ ਸਿੰਘ ਨਗਰ- 16●●62 ਰਾਹੀਂ ਪ੍ਰਕਾਸ਼ਿਤ ।

ਮੁੱਖ ਬੰਧ

ਪੰਜਾਬ ਸਕੂਲ ਸਿੱਖਿਆ ਬੋਰਡ, ਰਾਜ ਦੀ ਸਕੂਲ-ਸਿੱਖਿਆ ਨੂੰ ਸਮੇਂ ਦੀਆਂ ਲੋੜਾਂ ਤੇ ਵੰਗਾਰਾਂ ਦੇ ਪ੍ਰਸੰਗ ਵਿੱਚ ਢਾਲਣ ਤੇ ਨਵਿਆਉਣ ਲਈ ਨਿਰੰਤਰ ਯਤਨਸ਼ੀਲ ਹੈ।

ਅਸੀਂ ਇਤਿਹਾਸ ਦੇ ਉਸ ਕਾਲ-ਖੰਡ ਵਿੱਚੋਂ ਗੁਜ਼ਰ ਰਹੇ ਹਾਂ ਜਿੱਥੇ ਤੇਜ਼ੀ ਨਾ੧ ਪਰਿਵਰਤਨ ਵਾਪਰ ਰਹੇ ਹਨ। ਵਿਕਾਸ ਦੀ ਤੋਰ ਤਿਖੇਰੀ ਹੈ। ਇਸ ਵਿਕਸਤ ਸੰਸਾਰ ਨਾ੧ ਇਕਸੁਰ ਹੋਣ ਲਈ, ਜਿੱਥੇ ਗਿਆਨ ਦੀਆਂ ਤੰਦਾ ਵਿਸਤਰਿਤ ਹੋ ਗਈਆਂ ਹਨ, ਸੂਚਨਾ ਤਕਨਾ੧ਜੀ ਤੇ ਕੰਪਿਊਟਰ-ਸਿੱਖਿਆ ਨੂੰ ਸਿੱਖਿਆ ਦਾ ਅਹਿਮ ਅੰਗ ਬਣਾਉਣਾ ਜ਼ਰੂਰੀ ਹੋ ਗਿਆ ਹੈ।

ਇਸੇ ਮਨੋਰਥ ਨਾ੧ ਕੰਪਿਊਟਰ-ਤਕਨੀਕ ਦੀ ਸਿਖਲਾਈ ਹਿੱਤ ਇਹ ਪਾਠ-ਪੁਸਕਤ ਤਿਆਰ ਕੀਤੀ ਗਈ ਹੈ ਤੇ ਇਸ ਨੂੰ ਕੰਪਿਊਟਰ ਦੀ ਵੈੱਬਸਾਈਟ 'ਤੇ ਉਪਲੱਭਧ ਕਰਵਾਉਣ ਦਾ ਉਪਰਾਲਾ ਕੀਤਾ ਗਿਆ ਹੈ। ਨਿਸ਼ਚੇ ਹੀ ਇਹ ਸੁਵਿਧਾ ਕੰਪਿਊਟਰ ਸਾਇੰਸ, ਵੋਕੇਸ਼ਨਲ ਗਰੁੱਪ ਦੇ ਵਿਦਿਆਰਥੀਆਂ ਲਈ ਲਾਹੇਵੰਦ ਤੇ ਰੋਚਕ ਸਾਬਿਤ ਹੋਵੇਗੀ।

ਇਹ ਪਾਠ-ਸਮੱਗਰੀ ਕੰਪਿਊਟਰ ਸਿੱਖਿਆ ਦੇ ਵਿਦਵਾਨਾਂ, ਤਜਰਬੇਕਾਰ ਅਧਿਆਪਕਾਂ ਅਤੇ ਸਿੱਖਿਆ ਬੋਰਡ ਦੇ ਵਿਸ਼ਾ-ਮਾਹਿਰਾਂ ਦੇ ਸਾਂਝੇ ਉੱਦਮ ਸਦਕਾ ਤਿਆਰ ਕੀਤੀ ਗਈ ਹੈ। ਇਸ ਲਈ ਖੇਤਰ ਦੇ ਵਿਦਵਾਨ ਤੇ ਸਹਿਯੋਗੀ ਅਧਿਆਪਕ ਸਾਡੇ ਧੰਨਵਾਦ ਦੇ ਪਾਤਰ ਹਨ। ਇਸ ਨੂੰ ਹੋਰ ਬਿਹਤਰ, ਹੋਰ ਉਪਯੋਗੀ ਤੇ ਹੋਰ ਸੰਚਾਰਮਈ ਬਣਾਉਣ ਲਈ ਖੇਤਰ ਵਿੱਚੋਂ ਆਏ ਮੁੱਲਵਾਨ ਸੁਝਾਵਾਂ ਦਾ ਸਦਾ ਸਵਾਗਤ ਹੈ।

ਚੇਅਰਪਰਸਨ

ਪੰਜਾਬ ਸਕੂਲ ਸਿੱਖਿਆ ਬੋਰਡ

ਵਿਸ਼ਾ ਸੂਚੀ

ਪਾਠ 1

OOP ਕਾਨਸੈਪਟ

ਇਸ ਪਾਠ ਵਿਚ ਅਸੀਂ ਸਾਫਟਵੇਅਰ ਕੁਆਲਿਟੀ ਫੈਕਟਰਜ਼ (ਇੰਟਰਨਲ ਅਤੇ ਐਕਸਟਰਨਲ) ,ਜੇਨਰੀਸਿਟੀ (Genercity) ਅਤੇ ਓਵਰਲੋਡਿੰਗ (Overloading), ਆਬਜੈੱਕਟ ਓਰੀਐਂਟਿਡ ਫਿਲਾਸਫੀ, ਆਬਜੈੱਕਟ ਓਰੀਐਂਟਿਡ ਡਿਜ਼ਾਇਨ ਅਤੇ ਟੱਪ-ਡਾਊਨ (Top Down) ਡਿਜ਼ਾਈਨ ਬਾਰੇ ਜਾਣਾਗੇ।

1.1 ਸਾਫਟਵੇਅਰ ਕੁਆਲਿਟੀ ਫੈਕਟਰਜ਼

ਆਬਜੈੱਕਟ ਓਰੀਐਂਟਿਡ ਤਕਨੀਕ ਦੇ ਨਾਲ ਅਸੀਂ ਸਾਫਟਵੇਅਰ ਕੁਆਲਿਟੀ ਫੈਕਟਰਜ਼ (ਬਾਹਰੀ ਅਤੇ ਅੰਦਰੂਨੀ) ਤੇ ਜ਼ੋਰ ਪਾ ਸਕਦੇ ਹਾਂ।

1.1.1 **ਬਾਹਰੀ ਫੈਕਟਰਸ**—— ਇਹ ਸਿਰਫ ਐਂਡ-ਯੂਜ਼ਰ ਨੂੰ ਦਿਖਾਈ ਦਿੰਦੇ ਹਨ ਜਿਵੇਂ ਕਿ

* Correctness
* Robustness
* Extendibility
* Reusability
* Compatibility
* Efficiency
* Portability
* Ease of Use
* Functionality

1.1.2 **ਅੰਦਰੂਨੀ ਫੈਕਟਰਸ**—— ਇਹ ਸਿਰਫ ਸਾਫਟਵੇਅਰ ਬਣਾਉਣ ਵਾਲੇ ਨੂੰ ਦਿਖਾਈ ਦਿੰਦੇ ਹਨ ਜਿਵੇਂ ਕਿ

* Modularity
* Flexibility
* Reusability

ਅੰਦਰੂਨੀ ਤੱਤ, ਸਿਰਫ਼ ਕੰਪਿਊਟਰ ਦੇ ਮਾਹਰ ਹੀ ਰਿਸੀਵ ਕਰਦੇ ਹਨ। ਬਾਹਰੀ ਤੱਤ ਇਕ ਦੂਜੇ ਨਾਲ ਸੰਬੰਧਿਤ ਹੁੰਦੇ ਹਨ। ਨਿਰਸੰਦੇਹ ਇਹ ਯੂਜਰ ਨੂੰ ਦਿਖਾਈ ਦਿੰਦੇ ਹਨ ਪਰ ਫਿਰ ਵੀ ਬਾਹਰੀ ਤੱਤ ਅੰਦਰੂਨੀ ਕੁਆਲਿਟੀ ਤੱਤਾਂ ਤੇ ਨਿਰਭਰ ਹੁੰਦੇ ਹਨ।

1.2 ਮੇਜਰ ਐਕਸਟਰਨਲ ਫੈਕਟਰਸ (ਮੁੱਖ ਬਾਹਰੀ ਤੱਤ)

1.2.1 **Correctness.** Correctness ਸਾਫਟਵੇਅਰ ਦੀ ਉਹ ਸਮਰੱਥਾ ਹੈ ਜਿਸ ਦੇ ਵਿਚ ਸਾਫਟਵੇਅਰ ਦਿੱਤੀ ਹੋਈ ਪਰਿਭਾਸ਼ਾ ਦੇ ਅਨੁਸਾਰ ਕੰਮ ਕਰਦਾ ਹੈ।

* Correctness ਪਾਉਣ ਲਈ ਇਕ ਨਿਸ਼ਚਿਤ ਪਰਿਭਾਸ਼ਾ ਹੋਣੀ ਜ਼ਰੂਰੀ ਹੈ।
* Correctness ਸ਼ਰਤੀਆ ਹੁੰਦੀ ਹੈ। ਜੇਕਰ ਹੇਠਲੀ ਲੇਅਰ ਠੀਕ ਹੋਵੇ, ਜਿਸ ਦਾ ਅਰਥ ਹੈ ਕਿ ਜੇਕਰ ਸ਼ੁਰੂਆਤ ਵਿਚ ਹੀ ਸਾਫਟਵੇਅਰ ਦਿੱਤੀ ਹੋਈ ਪਰਿਭਾਸ਼ਾ ਦੇ ਅਨੁਸਾਰ ਬਣਨਾ ਸ਼ੁਰੂ ਹੋਵੇ ਤਾਂ ਅਖ਼ੀਰ ਵਿਚ Correctness ਜ਼ਰੂਰ ਪ੍ਰਾਪਤ ਹੋ ਜਾਂਦੀ ਹੈ।

1.2.2 **Robustness.** Robustness ਸਾਫਟਵੇਅਰ ਸਿਸਟਮ ਦੀ ਉਹ ਸਮਰਥਾ ਹੈ ਜਿਸ ਦੇ ਅਨੁਸਾਰ ਸਾਫਟਵੇਅਰ ਸਿਸਟਮ ਅਸਾਧਾਰਨ (abnormal) ਹਾਲਾਤਾਂ ਵਿਚ ਵੀ ਕੰਮ ਕਰ ਸਕੇ, Robustness ਦੱਸਦੀ ਹੈ ਕਿ ਪਰਿਭਾਸ਼ਾ ਦੇ ਬਾਹਰ ਕੀ ਹੋ ਰਿਹਾ ਹੈ।

1.2.3 **Extendibility.** ਇਹ ਸਾਫਟਵੇਅਰ ਸਿਸਟਮ ਦੀ ਉਹ ਵਿਸ਼ੇਸ਼ਤਾ ਹੈ ਜਿਸ ਦੇ ਅਨੁਸਾਰ ਸਾਫਟਵੇਅਰ ਬਦਲਾਵ ਨੂੰ ਅਸਾਨੀ ਨਾਲ ਧਾਰਨ ਕਰ ਲੈਂਦਾ ਹੈ।

1.2.4 **Reusability.** ਇਹ ਸਾਫਟਵੇਅਰ ਐਲੀਮੈਂਟਸ ਦੀ ਉਹ ਸਮਰਥਾ ਹੈ ਜਿਸ ਦੇ ਅਨੁਸਾਰ ਇਕ ਪ੍ਰੋਗਰਾਮ ਦੇ ਸਾਫਟਵੇਅਰ ਐਲੀਮੈਂਟਸ ਨੂੰ ਅਲਗ-ਅਲਗ ਸਾਫਟਵੇਅਰ ਐਪਲੀਕੇਸ਼ਨਾਂ ਬਣਾਉਣ ਲਈ ਵਰਤਿਆ ਜਾ ਸਕਦਾ ਹੈ।

1.2.5 **Compatibility.** ਇਸ ਵਿਚ ਸਾਫਟਵੇਅਰ ਐਲੀਮੇਂਟਸ ਨੂੰ ਅਸਾਨੀ ਨਾਲ ਹੋਰ ਐਲੀਮੇਂਟਸ ਨਾਲ ਜੋੜਿਆ ਜਾ ਸਕਦਾ ਹੈ।

1.2.6 **Portability.** ਇਸ ਵਿਚ ਸਾਫਟਵੇਅਰ ਨੂੰ ਅਸਾਨੀ ਨਾਲ ਅਲਗ-ਅਲਗ ਸਾਫਟਵੇਅਰ ਅਤੇ ਹਾਰਡਵੇਅਰ ਵਾਤਾਵਰਣ ਵਿੱਚ ਵਰਤਿਆ ਜਾ ਸਕਦਾ ਹੈ।

1.2.7 **Efficiency.** ਇਹ ਸਾਫਟਵੇਅਰ ਸਿਸਟਮ ਦੀ ਉਹ ਸਮਰੱਥਾ ਹੈ ਜਿਸ ਦੇ ਅਨੁਸਾਰ ਸਾਫਟਵੇਅਰ ਸਿਸਟਮ ਅਲਗ-ਅਲਗ ਹਾਰਡਵੇਅਰ ਰਿਸੋਰਸਾਂ 'ਤੇ ਬਹੁਤ ਹੀ ਘੱਟ ਬਦਲਾਵ ਦੀ ਮੰਗ ਕਰਦਾ ਹੋਇਆ ਕੰਮ ਕਰ ਸਕਦਾ ਹੈ।

1.2.8 **Ease of Use.** ਇਹ ਸਾਫਟਵੇਅਰ ਸਿਸਟਮ ਦਾ ਉਹ ਗੁਣ ਹੈ ਜਿਸ ਨਾਲ ਯੂਜ਼ਰ ਆਸਾਨੀ ਨਾਲ ਸਾਫਟਵੇਅਰ ਨੂੰ ਵਰਤਣਾ ਸਿੱਖਦੇ ਹਨ ਅਤੇ ਇਨ੍ਹਾਂ ਸਾਫਟਵੇਅਰਸ ਨੂੰ ਸਮੱਸਿਆਵਾਂ ਦਾ ਹੱਲ ਲੱਭਣ ਲਈ ਵਰਤਦੇ ਹਨ।

1.3 ਮੇਜਰ ਇੰਟਰਨਲ ਫੈਕਟਰਸ (ਮੁੱਖ ਅੰਦਰੂਨੀ ਤੱਤ)

1.3.1 ਮਾਡੂਲੈਰਿਟੀ (Modularity)

ਮਾਡੂਲੈਰਿਟੀ ਵਿਚ ਪ੍ਰੋਗਰਾਮ ਨੂੰ ਛੋਟੇ ਛੋਟੇ ਯੂਨਿਟ ਜਾਂ ਮਾਡਿਊਲਜ਼ ਵਿਚ ਵੰਡਿਆ ਜਾਂਦਾ ਹੈ ਜਿਵੇਂ ਕਿ C ਵਿਚ ਫੰਕਸ਼ਨਜ਼ ਬਣਾਉਣ ਲਈ ਵਰਤਿਆ ਜਾਂਦਾ ਹੈ। ਮਾਡਿਊਲਜ਼ ਵਿਚ ਵੰਡ ਕੇ ਅਸੀਂ ਇਕ ਵੱਡੇ ਪ੍ਰੋਗਰਾਮ ਨੂੰ ਅਸਾਨੀ ਨਾਲ ਲਿਖ ਸਕਦੇ ਹਾਂ ਤੇ ਕਿਸੇ ਗੁੰਝਲਦਾਰ ਸਮੱਸਿਆ ਦਾ ਹੱਲ ਵੀ ਅਸਾਨੀ ਨਾਲ ਲਿਖ ਸਕਦੇ ਹਾਂ। ਇਹ ਸਾਡੇ ਇਕ ਲੰਬੇ ਪ੍ਰੋਗਰਾਮ ਨੂੰ ਸੋਹਣੀ ਅਤੇ ਅਸਾਨ ਦਿੱਖ ਵੀ ਦਿੰਦਾ ਹੈ। ਅਸਲ ਵਿਚ ਮਾਡੂਲੈਰਿਟੀ ਦਾ ਮੁੱਖ ਲਾਭ "ਦੁਬਾਰਾ ਵਰਤਣਾ" (Reusability) ਹੈ। ਜਦੋਂ ਅਸੀਂ ਇਕ ਗੁੰਝਲਦਾਰ ਪ੍ਰੋਗਰਾਮ ਨੂੰ ਅਲਗ-ਅਲਗ ਮਾਡਿਊਲਜ਼ ਵਿਚ ਵੰਡ ਦਿੰਦੇ ਹਾਂ ਤਾਂ ਅਸੀਂ ਇਨ੍ਹਾਂ ਮਾਡਿਊਲਜ਼ ਨੂੰ ਕਈ ਤਰੀਕਿਆਂ ਨਾਲ ਵਰਤ ਸਕਦੇ ਹਾਂ। ਕਿਸੇ ਹੋਰ ਪ੍ਰੋਗਰਾਮ ਵਿਚ ਜਦੋਂ ਸਾਨੂੰ ਉਸੇ ਤਰ੍ਹਾਂ ਦੀ ਸਮੱਸਿਆ ਦਾ ਸਾਹਮਣਾ ਕਰਨਾ ਪੈਂਦਾ ਹੈ ਤਾਂ ਅਸੀਂ ਇਨ੍ਹਾਂ ਮਾਡਿਊਲਜ਼ ਦੀ ਵਰਤੋਂ ਉਸ ਪ੍ਰੋਗਰਾਮ ਵਿਚ ਵੀ ਕਰ ਸਕਦੇ ਹਾਂ।

1.3.2 Reusability

ਇਸ ਵਿਚ ਬਹੁਤ ਸਾਰੇ ਪ੍ਰੋਗਰਾਮਰ ਇਕੋ ਲਿਖਤੀ ਕਲਾਸ ਆਫ਼ ਡਾਟਾ ਨੂੰ ਵਰਤ ਸਕਦੇ ਹਨ। ਇਹ ਇਕ ਸਮਾਂ ਬਚਾਉਣ ਦੀ ਵਧੀਆ ਤਕਨੀਕ ਹੈ। ਇਸ ਦੇ ਨਾਲ ਹੀ ਇਕ ਪ੍ਰੋਗਰਾਮਰ ਮੌਜੂਦਾ ਕਲਾਸ ਵਿਚ ਹੋਰ ਵਿਸ਼ੇਸ਼ਤਾਵਾਂ ਵੀ ਪਾ ਸਕਦਾ ਹੈ। (ਲੋੜ ਮੁਤਾਬਿਕ)

1.4 ਜੈਨਰੀਸਿਟੀ ਅਤੇ ਓਵਰਲੋਡਿੰਗ (Genericity and Overloading)

ਜੈਨਰੀਸਿਟੀ:- ਜੈਨਰੀਸਿਟੀ ਆਬਜੈੱਕਟ ਓਰੀਐਂਟਿਡ ਪ੍ਰੋਗਰਾਮਿੰਗ ਦਾ ਇਕ ਮੁੱਖ ਮੰਤਵ ਹੈ। ਜੈਨਰਿਕ ਪ੍ਰੋਗਰਾਮ ਇਕ ਵਾਰ ਹੀ ਲਿਖੇ ਤੇ ਕੰਪਾਇਲ ਕੀਤੇ ਜਾਂਦੇ ਹਨ ਪਰ ਅਲਗ-ਅਲਗ ਡਾਟਾ ਟਾਈਪਸ ਦੇ ਨਾਲ ਵਰਤੇ ਜਾ ਸਕਦੇ ਹਨ। ਅਸੀਂ ਜੈਨਰੀਸਿਟੀ ਬਾਰੇ ਹੋਰ ਜਾਣਕਾਰੀ ਪਾਠ-7 ਵਿਚ ਹਾਸਿਲ ਕਰਾਂਗੇ।

ਓਵਰਲੋਡਿੰਗ:- ਜੇਕਰ ਅਸੀਂ ਇਕ ਫੰਕਸ਼ਨ ਨਾਮ ਦੀਆਂ ਇਕ ਤੋਂ ਜ਼ਿਆਦਾ ਪਰਿਭਾਸ਼ਾਵਾਂ ਦੱਸਦੇ ਹਾਂ ਜਾਂ ਫਿਰ ਇਕ ਆਪਰੇਟਰ ਦੀਆਂ ਇਕ ਤੋਂ ਜ਼ਿਆਦਾ ਪਰਿਭਾਸ਼ਾਵਾਂ ਦੱਸਦੇ ਹਾਂ ਤਾਂ ਅਸੀਂ ਓਪਰੇਟਰ ਓਵਰਲੋਡਿੰਗ ਜਾਂ ਫੰਕਸ਼ਨ ਓਵਰਲੋਡਿੰਗ ਕਰਦੇ ਹਾਂ। ਓਵਰਲੋਡਿੰਗ ਬਾਰੇ ਅਸੀਂ ਅਗਲੇ ਪਾਠਾਂ ਵਿਚ ਵਿਸਥਾਰ ਨਾਲ ਪੜ੍ਹਾਂਗੇ।

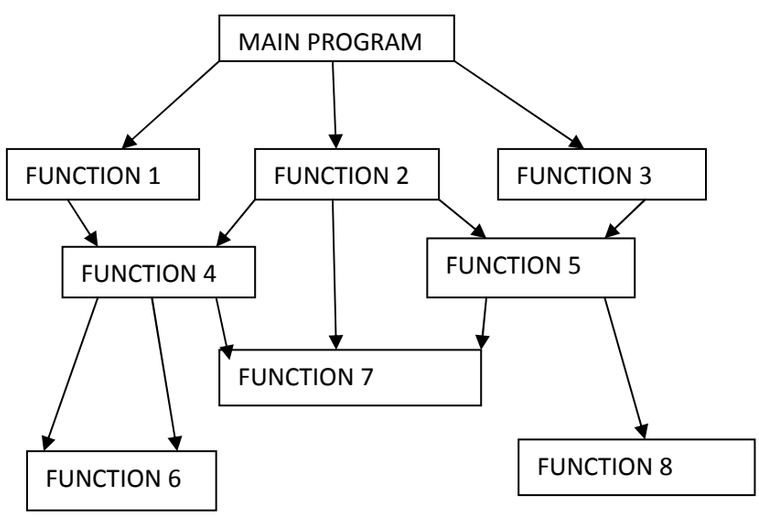
1.5 ਆਬਜੈੱਕਟ ਓਰੀਐਂਟਿਡ ਪ੍ਰੋਗਰਾਮਿੰਗ ਫਿਲਾਸਫੀ

ਫਿਲਾਸਫੀ ਦਾ ਮਤਲਬ ਹੈ ਕਿਸੇ ਚੀਜ਼ ਬਾਰੇ ਡੂੰਘੀ ਸਮਝ ਜਾਂ ਜਾਣਕਾਰੀ।

OOP (ਆਬਜੈੱਕਟ ਓਰੀਐਂਟਿਡ ਪ੍ਰੋਗਰਾਮਿੰਗ) ਨੂੰ ਸਮਝਣਾ: OOP ਦਾ ਕੀ ਮਤਲਬ ਹੈ ?

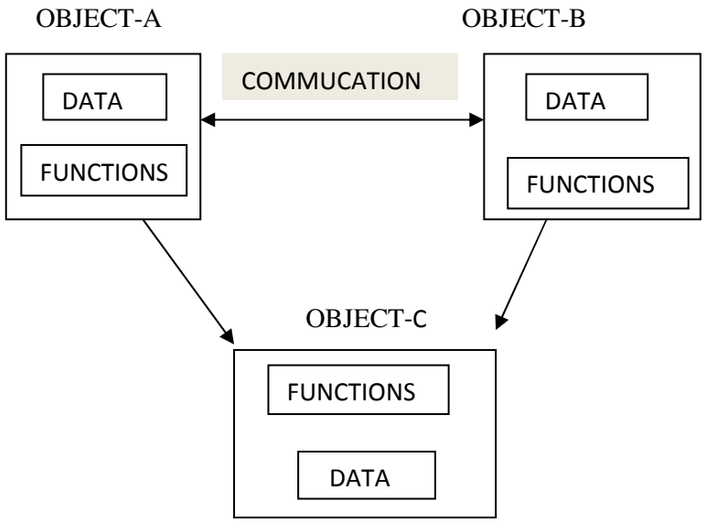
ਅਸੀਂ ਇਸ ਦਾ ਪੂਰਾ ਨਾਂ ਜਾਣਦੇ ਹਾਂ ਪਰ ਅਸਲ ਵਿਚ ਇਹ ਹੈ ਕੀ ?ਇਹ ਇਕ ਪ੍ਰੋਗਰਾਮਿੰਗ ਸਟਾਈਲ ਹੈ ਜਿਸ ਦੀ ਸੋਚ ਅਸਲ ਦੁਨੀਆਂ ਦੇ ਆਬਜੈਕਟ ਉੱਤੇ ਆਧਾਰਿਤ ਹੈ। ਇਹੀ ਕਾਰਨ ਹੈ ਕਿ ਇਸ ਦਾ OOP ਨਾਮ ਪਿਆ। ਸਾਧਾਰਨ ਤੌਰ ਤੇ ਜਦੋਂ ਅਸੀਂ ਪ੍ਰੋਗਰਾਮਿੰਗ ਬਾਰੇ ਸੋਚਦੇ ਹਾਂ ਤਾਂ ਅਸੀਂ ਪ੍ਰੋਗਰਾਮ ਦੀ ਫੰਕਸ਼ਨੇਲਟੀ (ਕੰਮ) ਬਾਰੇ ਸੋਚਦੇ ਹਾਂ। ਜਿਵੇਂ ਕਿ ਇਹ ਪ੍ਰੋਗਰਾਮ ਵਿਦਿਆਰਥੀਆਂ ਦੀ ਡਿਗਰੀ (Degree) ਕੈਲਕੁਲੇਟ ਕਰਦਾ ਹੈ। ਇਸ ਤਰ੍ਹਾਂ ਅਸੀਂ ਪ੍ਰੋਗਰਾਮ ਨੂੰ ਉਸਦੇ ਕੰਮ ਦੇ ਅਨੁਸਾਰ ਵੰਡ ਦਿੰਦੇ ਹਾਂ। ਅਸੀਂ ਵਿਦਿਆਰਥੀਆਂ ਦੇ ਨੰਬਰ ਪ੍ਰਾਪਤ ਕਰਨ ਲਈ ਵੀ ਇਕ ਫੰਕਸ਼ਨ ਬਣਾ ਸਕਦੇ ਹਾਂ, ਫਿਰ ਅਸੀਂ ਉਨ੍ਹਾਂ ਦੇ ਫਾਈਨਲ ਨੰਬਰ ਜੋੜਨ ਲਈ ਇਕ ਪ੍ਰੋਗਰਾਮ ਬਣਾ ਸਕਦੇ ਹਾਂ, ਫਿਰ ਵਿਦਿਆਰਥੀਆਂ ਦੇ ਗਰੇਡ ਪ੍ਰਾਪਤ ਕਰਨ ਲਈ ਇਕ ਪ੍ਰੋਗਰਾਮ ਬਣਾ ਸਕਦੇ ਹਾਂ ਤੇ ਇਸੀ ਤਰ੍ਹਾਂ ਇਹ ਪ੍ਰਕ੍ਰਿਆ ਚਲਦੀ ਰਹੇਗੀ। ਇਸ ਤਰ੍ਹਾਂ ਦੀ ਆਦਤ ਸਾਡੇ ਦਿਮਾਗ਼ ਤੇ ਅਸਰ ਪਾਉਂਦੀ ਹੈ ਤੇ ਅਸੀਂ ਹੋਰ ਕਿਸੇ ਚੀਜ਼ ਬਾਰੇ ਨਹੀਂ ਸੋਚ ਸਕਦੇ। ਇਸ ਤਰ੍ਹਾਂ ਦੇ ਲੱਛਣ Procedural Programmes ਵਿਚ ਪਾਏ ਜਾਂਦੇ ਹਨ ਜਿਵੇਂ ਕਿ Pascal / C Programmers.

ਹੁਣ ਅਸੀਂ ਜਾਣਾਂਗੇ ਕਿ ਆਬਜੈੱਕਟ ਓਰੀਐਂਟਿਡ ਫਿਲਾਸਫੀ ਕਿਵੇਂ ਕੰਮ ਕਰਦੀ ਹੈ ?ਸਭ ਤੋਂ ਪਹਿਲਾਂ ਅਸੀਂ ਪ੍ਰੋਗਰਾਮ ਦੇ ਉਦੇਸ਼ (Objective) ਬਾਰੇ ਫੈਸਲਾ ਕਰਦੇ ਹਾਂ। ਪ੍ਰੋਗਰਾਮ ਤੋਂ ਅਸੀਂ ਕੀ ਹਾਸਿਲ ਕਰਨਾ ਚਾਹੁੰਦੇ ਹਾਂ ਇਹ ਪਤਾ ਲਗਾਉਂਦੇ ਹਾਂ। ਅਸੀਂ ਆਪਣੀ ਸੋਚ ਆਬਜੈਕਟ ਤੇ ਆਧਾਰਿਤ ਰੱਖਦੇ ਹਾਂ ਨਾ ਕਿ ਫੰਕਸ਼ਨੈਲਟੀ 'ਤੇ।



Typical Structure of Procedure-Oriented Programs

OOP ਡਾਟਾ ਨੂੰ ਪ੍ਰੋਗਰਾਮ ਡੇਵਲਪਮੈਂਟ ਦੇ ਇਕ Critical Elements ਦੇ ਰੂਪ ਵਿਚ ਲੈਂਦਾ ਹੈ ਤੇ ਇਸ ਨੂੰ ਸਿਸਟਮ ਵਿਚ ਕਿਧਰੇ ਵੀ ਆਜ਼ਾਦੀ ਨਾਲ ਘੁੰਮਣ ਦੀ ਆਗਿਆ ਨਹੀਂ ਦਿੰਦਾ। OOP ਸਾਨੂੰ ਆਗਿਆ ਦਿੰਦਾ ਹੈ ਕਿ ਅਸੀਂ ਪ੍ਰੋਗਰਾਮ ਨੂੰ ਕਈ Entities, ਜਿਨ੍ਹਾਂ ਨੂੰ ਆਬਜੈੱਕਟ ਕਿਹਾ ਜਾਂਦਾ ਹੈ, ਵਿਚ ਵੰਡ ਦਿੰਦੇ ਹਾਂ ਤੇ ਫਿਰ ਡਾਟਾ ਤੇ ਫੰਕਸ਼ਨ ਇਨ੍ਹਾਂ ਆਬਜੈੱਕਟ ਸ ਨੂੰ ਮੁੱਖ ਰੱਖਦੇ ਹੋਏ ਬਣਾਂਦੇ ਹਾਂ। ਆਬਜੈੱਕਟ ਓਰੀਐਂਟਿਡ ਪ੍ਰੋਗਰਾਮ ਵਿਚ ਡਾਟਾ ਤੇ ਫੰਕਸ਼ਨ ਦਾ ਪ੍ਰਬੰਧ ਹੇਠ ਲਿਖੇ ਚਿੱਤਰ ਵਿਚ ਦਿਖਾਇਆ ਗਿਆ ਹੈ।



Organization of Data and Function in OOP

1.6 ਬੇਸਿਕ ਕਾਨਸੈਪਟ (Concept) ਆਫ OOP

OOP ਵਿਚ ਹੇਠਾਂ ਲਿਖੇ Concepts ਸ਼ਾਮਿਲ ਹਨ——

1. ਆਬਜੈੱਕਟਸ (Objects)
2. ਕਲਾਸਾਂ (classes)
3. ਡਾਟਾ ਐਬਸਟਰੈਕਸ਼ਨ ਤੇ ਐਨਕੇਪਸੂਲੇਸ਼ਨ (Data Abstraction and Encapsulation)
4. ਇਨਹੈਰੀਟੈਂਸ (Inheritance)
5. ਪੋਲੀਮੋਰਫਿਜਮ (Polymorphism)
6. ਡਾਈਨਾਮਿਕ ਬਾਈਂਡਿੰਗ (Dynamic Binding)
7. ਮੈਸੇਜ ਪਾਸਿੰਗ (Message Passing)

ਇਨ੍ਹਾਂ ਕਾਨਸੈਪਟ ਬਾਰੇ ਅਸੀਂ ਅਗਲੇ ਪਾਠਾਂ ਵਿਚ ਵਿਸਥਾਰ ਨਾਲ ਜਾਣਾਂਗੇ।

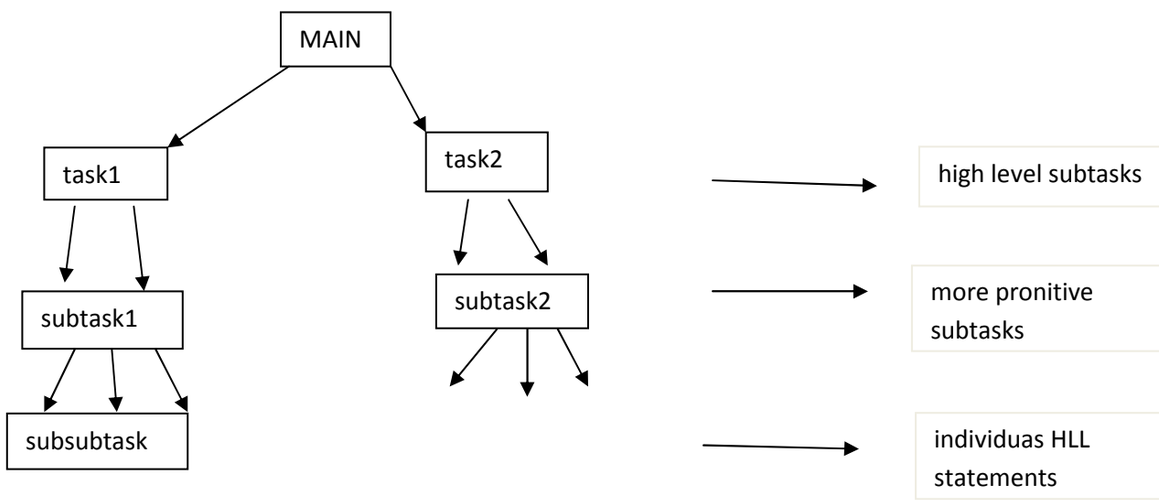
1.7 ਟਾਪ ਡਾਊਨ ਡਿਜ਼ਾਈਨ ਤੇ ਫੰਕਸ਼ਨਲ ਡਿਕੰਪੋਜੀਸ਼ਨ (Top-Down Design and Functional Decomposition)

1 ਇਕ ਸਮੱਸਿਆ (Problem) ਨੂੰ ਹਾਈ- ਲੈਵਲ ਸਬ-ਸਮੱਸਿਆਵਾਂ (Sub-problems) ਵਿਚ ਦਿਖਾਉ।

2 ਹਰ ਇਕ ਹਾਈ ਲੈਵਲ Sub-problem ਲਈ।

2.1 ਇਸ ਨੂੰ ਥੋੜ੍ਹੀ ਜਿਹੀ ਹੋਰ ਅਸਾਨ ਸਬ-ਸਮੱਸਿਆ (Sub-problem) ਵਿਚ ਦਿਖਾਉ।

3 ਇਹ ਪ੍ਰਕ੍ਰਿਆ ਉਸ ਸਮੇਂ ਤੱਕ ਚਲਦੀ ਰਹਿਦੀ ਚਾਹੀਦੀ ਹੈ ਜਦ ਤੱਕ ਕਿ ਸਾਰੀਆਂ Sub-problems ਪ੍ਰੋਗਰਾਮਿੰਗ ਸਟੇਟਮੈਂਟ ਵਿਚ ਨਾ ਬਦਲ ਜਾਣ। ਨੋਟ—— ਹਰ ਸਬ-ਟਾਸਕ (Sub-task) ਇਕ ਸਬ-ਪ੍ਰੋਗਰਾਮ (Sub-program) ਬਣ ਜਾਂਦਾ ਹੈ।



ਯਾਦ ਰੱਖਣ ਯੋਗ ਗੱਲਾਂ (Points to be Remember) :

1. ਸਾਫਟਵੇਅਰ ਕੁਆਲਿਟੀ ਫੈਕਟਰਜ਼ ਦੋ ਤਰ੍ਹਾਂ ਦੇ ਹੁੰਦੇ ਹਨ——ਬਾਹਰੀ ਤੇ ਅੰਦਰੂਨੀ।
2. ਬਾਹਰੀ ਫੈਕਟਰਜ਼ ਸਿਰਫ਼ ਐਂਡ ਯੂਜਰ ਨੂੰ ਦਿਖਾਈ ਦਿੰਦੇ ਹਨ।
3. ਅੰਦਰੂਨੀ ਫੈਕਟਰਜ਼ ਸਿਰਫ਼ ਸਾਫਟਵੇਅਰ ਬਣਾਉਣ ਵਾਲੇ ਨੂੰ ਦਿਖਾਈ ਦਿੰਦੇ ਹਨ।
4. Correctness ਪਾਉਣ ਲਈ ਇਕ ਨਿਸ਼ਚਿਤ ਪਰਿਭਾਸ਼ਾ ਹੋਣੀ ਜ਼ਰੂਰੀ ਹੈ।
5. ਮਾਡੂਲੈਰਿਟੀ ਵਿਚ ਪ੍ਰੋਗਰਾਮ ਨੂੰ ਛੋਟੇ-ਛੋਟੇ ਯੂਨਿਟ ਜਾਂ ਮਾਡਿਊਲਜ਼ ਵਿਚ ਵੰਡਿਆ ਜਾਂਦਾ ਹੈ।
6. ਆਬਜੈੱਕਟ ਓਰੀਐਂਟਿਡ ਪ੍ਰੋਗਰਾਮਿੰਗ ਉਹ ਸਟਾਈਲ ਹੈ ਜਿਸ ਦੀ ਸੋਚ ਅਸਲ ਦੁਨੀਆਂ ਦੇ ਆਬਜੈਕਟ ਉੱਤੇ ਅਧਾਰਿਤ ਹੈ।

ਅਭਿਆਸ
(Exercise)

1. ਖਾਲੀ ਥਾਵਾਂ ਭਰੋ——

1. ..... ਸਾਫਟਵੇਅਰ ਬਣਾਉਣ ਵਾਲੇ ਨੂੰ ਦਿਖਾਈ ਦਿੰਦੇ ਹਨ।
2. ..... ਦੇ ਅਨੁਸਾਰ ਸਾਫਟਵੇਅਰ ਬਦਲਾਵ ਨੂੰ ਅਸਾਨੀ ਨਾਲ ਧਾਰਨ ਕਰ ਲੈਂਦਾ ਹੈ।
3. ..... ਵਿਚ ਪ੍ਰੋਗਰਾਮ ਨੂੰ ਛੋਟੇ-ਛੋਟੇ ਯੂਨਿਟ ਵਿਚ ਵੰਡਿਆ ਜਾਂਦਾ ਹੈ।
4. OOP ਦਾ ਅਰਥ ਹੈ .....।
5. OOP ਪ੍ਰੋਗਰਾਮ ਨੂੰ, ਕਈ Entities ਜਿਨ੍ਹਾਂ ਨੂੰ ..... ਕਿਹਾ ਜਾਂਦਾ ਹੈ, ਵਿਚ ਵੰਡ ਦਿੰਦਾ ਹੈ।

2. ਪ੍ਰਸ਼ਨਾਂ ਦੇ ਉੱਤਰ ਦਿਉ——

1. Correctness ਉੱਤੇ ਸੰਖੇਪ ਨੋਟ ਲਿਖੋ।
2. ਮਾਡੂਲੈਰਿਟੀ ਕੀ ਹੈ ?
3. ਆਬਜੈੱਕਟ ਓਰੀਐਂਟਿਡ ਪ੍ਰੋਗਰਾਮਿੰਗ ਅਤੇ ਪ੍ਰੋਸੀਜੀਰਲ (Procedural) ਪ੍ਰੋਗਰਾਮਿੰਗ ਵਿਚ ਕੀ ਫਰਕ ਹੈ ?
4. OOP ਦੇ ਬੇਸਿਕ ਕਾਨਸੈਪਟ ਕੀ ਹਨ ?

ਪਾਠ 2

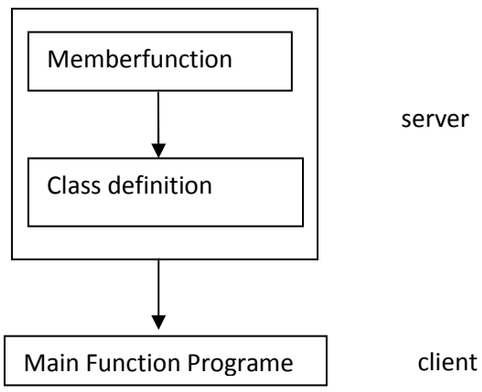
C++ ਭਾਸ਼ਾ ਵਿਸ਼ੇਸ਼ਤਾਵਾਂ
(C++ Language Features)

C++ ਭਾਸ਼ਾ ਇਕ ਆਬਜੈੱਕਟ ਓਰੀਐਂਟਿਡ (OOP) ਭਾਸ਼ਾ ਹੈ। ਇਹ ਭਾਸ਼ਾ ਸਭ ਤੋਂ ਪਹਿਲਾਂ, ਨਿਊਜਰਸੀ ਵਿੱਚ ਸਥਿਤ ਏ.ਟੀ. ਐਂਡ. ਟੀ. ਬੈਲ ਲਬੋਰਟਰੀ (AT & T.Bell Labortaries) ਦੇ Bjarne Stroustrup ਦੁਆਰਾ 198● ਵਿਚ ਬਣਾਈ ਗਈ। ਸਟਰਾਊਸਟਰਪ, ਸਿਮੂਲਾ 67 (Simula 67) ਅਤੇ C ਭਾਸ਼ਾ ਨੂੰ ਮਿਲਾਕੇ ਇਕ ਸ਼ਕਤੀਸ਼ਾਲੀ ਭਾਸ਼ਾ ਦਾ ਨਿਰਮਾਣ ਕਰਨਾ ਚਾਹੁੰਦਾ ਸੀ ਜੋ ਕਿ ਆਬਜੈੱਕਟ ਓਰੀਐਂਟਿਡ ਭਾਸ਼ਾ ਦੀਆਂ ਵਿਸ਼ੇਸ਼ਤਾਵਾਂ ਨੂੰ ਲਵੇ ਅਤੇ ਨਾਲ ਹੀ C ਭਾਸ਼ਾ ਦੀ power ਅਤੇ elegance ਨੂੰ ਵੀ ਆਪਣੇ ਆਪ ਵਿਚ ਸਮਾ ਕੇ ਰੱਖੇ ਇਸ ਤਰ੍ਹਾਂ C++ ਭਾਸ਼ਾ ਦਾ ਨਿਰਮਾਣ ਹੋਇਆ।

ਅਸਲ C ਭਾਸ਼ਾ ਵਿਚ ਕਲਾਸ (class) ਇਕ ਸਭ ਤੋਂ ਵੱਡਾ ਜੋੜ (addition) ਸੀ। ਇਸ ਕਰਕੇ ਪਹਿਲਾਂ ਇਸ ਭਾਸ਼ਾ ਨੂੰ 'C ਵਿਦ ਕਲਾਸ ' (C with classes) ਨਾਮ ਦਿੱਤਾ ਗਿਆ। 1983 ਵਿਚ ਇਸ ਦਾ ਨਾਮ ਬਦਲ ਕੇ C++ ਕਰ ਦਿੱਤਾ ਗਿਆ। C++ ਨਾਮ ਰੱਖਣ ਦਾ ਵਿਚਾਰ (++) ਇਨਕਰੀਮੈਂਟ ਓਪਰੇਟਰ ਤੋਂ ਆਇਆ, ਜੋ ਇਹ ਦੱਸਦਾ ਹੈ ਕਿ C++, C ਦਾ ਹੀ ਇਕ ਇਨਕਰੀਮੈਂਟ ਹੈ।

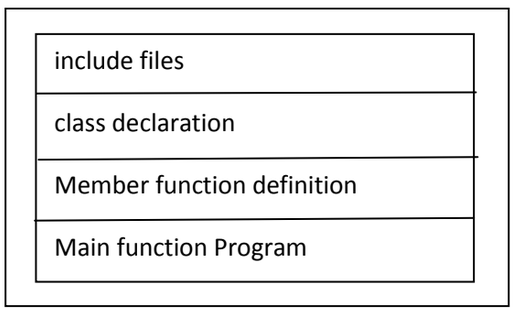
2.1 C++ ਦਾ ਸਟਰਕਚਰ (Structure of C++)

C++ ਪ੍ਰੋਗਰਾਮ ਦੇ ਚਾਰ ਭਾਗ ਹੁੰਦੇ ਹਨ ਜੋ ਕਿ ਹੇਠਾਂ ਲਿਖੇ ਅਨੁਸਾਰ ਹਨ——



THE CLIENT-SERVER MODEL

The Client-Server Model
ਚਿੱਤਰ 2.1



Structure of a C++ Program
ਚਿੱਤਰ 2.1.1

2.2 ਇਨਪੁਟ ਓਪਰੇਟਰ (Input operator)

```
# include <iostream.h>
int main( )
{
    float number 1, number 2,
    sum, average ;
```

```
cout << “Enter two numbers :”; // prompt
cin >> number 1 ; // Reads numbers
cin >> number 2 ; // from keyboad
sum = number 1 + number 2 ;
average = sum/2 ;
cout << “ sum = ” << sum << “\n” ;
cout << “Average =“ << average << “\n” ;
return 0 ;
}
```

```
The output of the program is
Enter two numbers : 6.5  7.5
Sum = 14
Average = 7
```

2.2.1 cin >> number 1 : ਇਕ ਇਨਪੁਟ ਸਟੇਟਮੈਂਟ ਹੈ ਜਿਸ ਵਿਚ ਯੂਜਰ ਜਦ ਤੱਕ ਨੰਬਰ ਟਾਈਪ ਕਰਦਾ ਹੈ ਤੱਦ ਤਕ ਪ੍ਰੋਗਰਾਮ ਨੂੰ ਇੰਤਜ਼ਾਰ ਕਰਨ ਲਈ ਕਿਹਾ ਜਾਂਦਾ ਹੈ। ਇਸ ਸਟੇਟਮੈਂਟ ਵਿਚ number 1 ਇਕ ਵੇਰੀਏਬਲ ਹੈ ਅਤੇ cin, C++ ਵਿਚ ਪਹਿਲਾਂ ਤੋਂ ਪਰਿਭਾਸ਼ਿਤ ਕੀਤਾ ਆਬਜੈਕਟ ਹੈ ਜੋ ਕਿ ਸਟੈਂਡਰਡ ਇਨਪੁਟ ਸਟਰੀਮ (stream) ਦੇ ਨਾਲ ਸੰਬੰਧਿਤ ਹੁੰਦਾ ਹੈ।

ਇੱਥੇ ਇਹ ਸਟਰੀਮ (Stream) ਕੀਬੋਰਡ ਨੂੰ ਦਰਸਾਉਂਦੀ ਹੈ। ਓਪਰੇਟਰ >> ਨੂੰ ਐਕਸਟਰੈਕਸ਼ਨ (extraction) ਜਾਂ get from ਓਪਰੇਟਰ ਕਿਹਾ ਜਾਂਦਾ ਹੈ।

2.3 ਆਊਟਪੁਟ ਓਪਰੇਟਰ (Output operator)

```
# include <iostream.h>
int main( )
{
    cout << “C++ is better than C” ;                    // C++ statement
    return 0;
}          // End of example.
```

ਇਸ ਪ੍ਰੋਗਰਾਮ ਵਿਚ ਸਿਰਫ ਇਕ ਆਊਟਪੁਟ ਸਟੇਟਮੈਂਟ ਹੈ।

```
cout << “C++ is better than C.” ;
```

ਇਹ ਸਟੇਟਮੈਂਟ ਕੋਟੇਸ਼ਨ ਮਾਰਕਸ (“ ”) ਵਿਚ ਲਿਖੀ ਸਟਰਿੰਗ ਨੂੰ ਸਕਰੀਨ ਉੱਤੇ ਦਿਖਾਉਂਦਾ ਹੈ। ਇਸ ਸਟੇਟਮੈਂਟ ਵਿਚ ਅਸੀਂ C++ ਦੀ ਦੋ ਨਵੀਆਂ ਵਿਸ਼ੇਸ਼ਤਾਵਾਂ ਦੇਖਦੇ ਹਾਂ।

cout ਅਤੇ <<

cout, C++ ਵਿਚ ਪਹਿਲਾਂ ਤੋਂ ਪਰਿਭਾਸ਼ਿਤ ਇਕ ਆਬਜੈੱਕਟ ਹੈ। << ਨੂੰ ਇਨਸਰਸ਼ਨ ਜਾਂ Put to (operator) ਓਪਰੇਟਰ ਵੀ ਕਿਹਾ ਜਾਂਦਾ ਹੈ।

2.4 ਵੇਰੀਏਬਲ ਡੈਕਲੇਰੇਸ਼ਨ (Variable Declaration)

C++ ਵਿਚ ਸਾਰੇ ਵੇਰੀਏਬਲਜ਼ ਨੂੰ ਘੋਸ਼ਿਤ ਕਰਨਾ ਜ਼ਰੂਰੀ ਹੈ। (ਲਾਗੂ ਕਰਨ ਵਾਲੀ (executable) ਸਟੇਟਮੈਂਟਸ ਤੋਂ ਪਹਿਲਾਂ) C ਵਿਚ ਸਾਰੇ ਵੇਰੀਏਬਲ ਸ਼ੁਰੂਆਤ ਵਿਚ ਹੀ ਘੋਸ਼ਿਤ ਕੀਤੇ ਜਾਂਦੇ ਹਨ, ਪਰ C++ ਵੇਰੀਏਬਲਜ਼ ਨੂੰ ਕਿਤੇ ਵੀ ਘੋਸ਼ਿਤ ਕਰਨ (declare) ਦੀ ਆਗਿਆ ਦਿੰਦਾ ਹੈ ਜਿਸ ਦਾ ਅਰਥ ਹੈ ਕਿ ਵੇਰੀਏਬਲ ਨੂੰ ਸਭ ਤੋਂ ਪਹਿਲਾਂ ਇਸਤੇਮਾਲ ਵਾਲੀ ਥਾਂ ਤੇ ਘੋਸ਼ਿਤ ਕੀਤਾ ਜਾ ਸਕਦਾ ਹੈ।

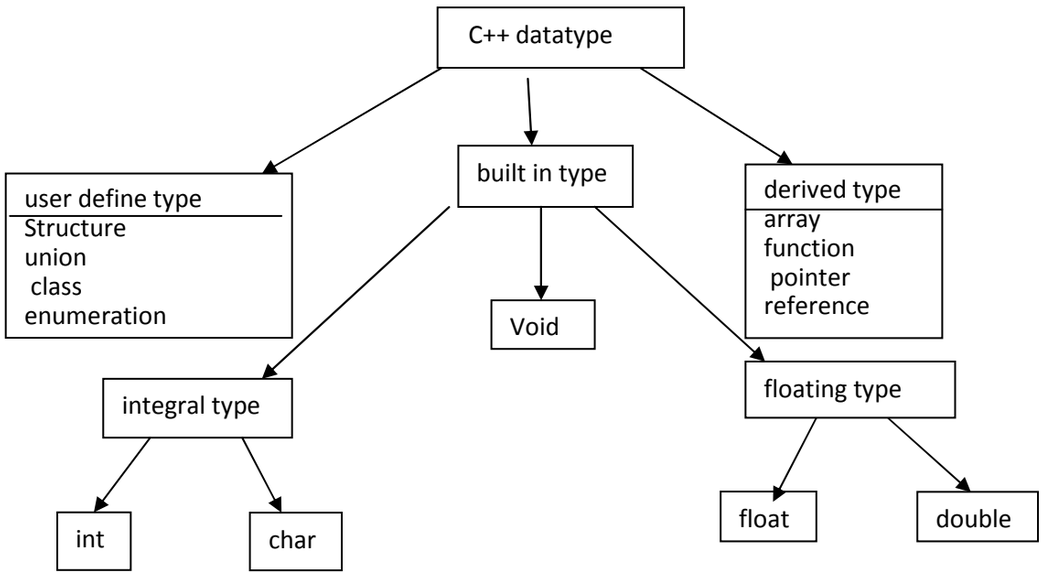
2.4 ਉਦਾਹਰਣ ਤੇ ਵਿਚਾਰ ਕਰੋ

```
int main( )
{
    float x ;
    float sum = 0 ;                 // declaration
    for (int i = 1 ; i < 5 ; i++)
    {                               // declaration
    cin >> x ;
    sum = sum + x ;
    }
    float average ;
```

```
    average = sum / i ;       //declaration
    cout << average ;
    return 0 ;
}
```

2.5 C++ ਵਿਚ ਡਾਟਾ ਟਾਈਪਸ

C++ ਦੀਆਂ ਬੇਸਿਕ ਡਾਟਾ ਟਾਈਪਸ ਹੇਠਾਂ ਲਿਖੇ ਅਨੁਸਾਰ ਹਨ ਜੋ ਕਿ ਚਿੱਤਰ 2.5 ਵਿਚ ਦਿਖਾਈਆਂ ਹਨ।



C ਅਤੇ C++ ਕੰਪਾਈਲਰ ਸਾਰੀਆਂ ਬਿਲਟ ਇਨ (built-in) data types ਨੂੰ support ਕਰਦਾ ਹੈ।

2.6 C++ ਦੇ ਓਪਰੇਟਰ

C ਦੇ ਸਾਰੇ ਓਪਰੇਟਰ C++ ਵਿਚ ਉਪਲਬਧ ਹਨ। ਇਸ ਤੋਂ ਇਲਾਵਾ C++ ਵਿਚ ਕੁਝ ਨਵੇਂ ਓਪਰੇਟਰ ਵੀ ਆਏ ਹਨ। ਦੋ ਓਪਰੇਟਰ ਬਾਰੇ ਅਸੀਂ ਪਹਿਲਾਂ ਹੀ ਪੜ੍ਹ ਚੁੱਕੇ ਹਾਂ। Insertion ਓਪਰੇਟਰ >> ਅਤੇ extraction ਓਪਰੇਟਰ <<

ਕੁਝ ਹੋਰ ਓਪਰੇਟਰ ਇਸ ਤਰ੍ਹਾਂ ਹਨ :

::	Scope resolution ਓਪਰੇਟਰ
::*	Pointer-to-member declarator
→.*	Pointer-to-member operator
* .	Pointer-to-member operator
delete	Memory release operator
endl	Line Feed operator
new	Memory allocation operator
set w	Field width operator

2.7 ਕਾਂਸਟੈਂਟ ਕੁਆਲੀਫਾਇਰ (The const Qualifier)

ਕੀ-ਵਰਡ ਕਾਂਸਟੈਂਟ (const.), ਜੇਕਰ ਹੈ, ਤਾਂ ਵੇਰੀਏਬਲ ਦੇ ਡਾਟਾ ਟਾਈਪ ਤੋਂ ਪਹਿਲਾਂ ਲਿਖਿਆ ਜਾਂਦਾ ਹੈ। ਇਸ ਤੋਂ ਇਹ ਪਤਾ ਲੱਗਦਾ ਹੈ ਕਿ ਵੇਰੀਏਬਲ ਦੀ ਕੀਮਤ ਪੂਰੇ ਪ੍ਰੋਗਰਾਮ ਵਿਚ ਨਹੀਂ ਬਦਲਦੀ। ਜੇਕਰ ਉਸ ਵੇਰੀਏਬਲ ਦੀ ਕੀਮਤ ਨੂੰ ,ਜੋ ਕਿ ਕੁਆਲੀਫਾਇਰ (Qualifier) ਦੇ ਨਾਲ ਘੋਸ਼ਿਤ ਹੁੰਦਾ ਹੈ ਛੇੜਿਆ ਜਾਵੇ ਤਾਂ ਕੰਪਾਇਲਰ ਤੋਂ ਐਰਰ ਮੈਸੇਜ ਆਉਂਦਾ ਹੈ।

ਕਾਂਸਟੈਂਟ #defined constant ਨੂੰ ਬਦਲਣ ਲਈ ਵਰਤੇ ਜਾਂਦੇ ਹਨ।

const qualifier ਇਹ ਨਿਸ਼ਚਿਤ ਕਰਦਾ ਹੈ, ਤੁਹਾਡਾ ਪ੍ਰੋਗਰਾਮ ਵੇਰੀਏਬਲ ਜੋ ਕਿ constant ਹੈ ਨੂੰ ਆਲਟਰ (alter) ਨਹੀਂ ਕਰਦਾ। ਇਹ, ਇਹ ਵੀ ਯਾਦ ਦਿਵਾਉਂਦਾ ਹੈ ਕਿ ਜੇਕਰ ਕੋਈ ਪ੍ਰੋਗਰਾਮ ਪੜ੍ਹ ਰਿਹਾ ਹੋਵੇ ਕਿ ਤਾਂ ਇਹ ਲਿਸਟ ਕੀਤਾ ਜਾਂਦਾ ਹੈ ਕਿ ਵੇਰੀਏਬਲ ਨਹੀਂ ਬਦਲ ਸਕਦਾ।

ਇਸ ਕੁਆਲੀਫਾਇਰ ਦੇ ਵੇਰੀਏਬਲ ਨੂੰ ਅਪਰਕੇਸ ਵਿਚ ਨਾਮ ਦਿੱਤਾ ਜਾਂਦਾ ਹੈ ਅਤੇ ਇਹ ਯਾਦ ਦਿਵਾਉਂਦਾ ਹੈ ਕਿ ਇਹ ਕਾਂਸਟੈਂਟ ਹਨ।

ਹੇਠ ਲਿਖਿਆ ਪ੍ਰੋਗਰਾਮ const ਦੀ ਵਰਤੋਂ ਦਰਸਾਉਂਦਾ ਹੈ,

```
# include <iostream.h>
void main( )
{
    float r, a ;
    const float Pi = 3.14 ;
    cin >> r ;
```

```
    a = Pi * r * r ;
    cout << end 1 << ‘‘Area of Circle = ’’ << a ;
}
```

const, # define ਦੇ ਮੁਕਾਬਲੇ ਵਧੀਆ ਤਰੀਕਾ ਹੈ, ਪਰੰਤੂ ਇਸ ਦੇ ਆਪਰੇਸ਼ਨ (Operation) ਨੂੰ ਫੰਕਸ਼ਨ ਦੇ ਅੰਦਰ ਜਾਂ ਬਾਹਰ ਕੰਟਰੋਲ ਕੀਤਾ ਜਾਂਦਾ ਹੈ।

ਜੇਕਰ const ਨੂੰ ਫੰਕਸ਼ਨ ਦੇ ਅੰਦਰ ਰੱਖਿਆ ਜਾਂਦਾ ਹੈ ਤਾਂ ਇਸ ਦਾ ਅਸਰ ਲੋਕਲਾਈਜ਼ਡ (Localised) ਹੋਵੇਗਾ। ਜੇਕਰ ਬਾਹਰ ਰੱਖਿਆ ਜਾਂਦਾ ਹੈ ਤਾਂ ਗਲੋਬਲ (Global) ਹੋਵੇਗਾ।

2.8 ਇਨਲਾਈਨ ਫੰਕਸ਼ਨਜ਼ (Inline Functions)

ਫੰਕਸ਼ਨ ਦੀ ਵਰਤੋਂ ਕਰਨ ਦੀ ਸਭ ਤੋਂ ਵੱਡੀ ਵਿਸ਼ੇਸ਼ਤਾ ਇਹ ਹੈ ਕਿ ਇਹ ਮੈਮਰੀ ਸਪੇਸ ਬਚਾਉਣ ਵਿਚ ਮਦਦ ਕਰਦੀ ਹੈ।

ਫੰਕਸ਼ਨ ਨੂੰ ਕਾਲ ਕਰਨ ਨਾਲ ਇਕੋ ਹੀ ਕੋਡ ਲਾਗੂ ਹੁੰਦਾ ਹੈ ਅਤੇ ਫੰਕਸ਼ਨ ਬਾਡੀ ਨੂੰ ਮੈਮਰੀ ਵਿਚ ਡੁਪਲੀਕੇਟ ਨਹੀਂ ਬਣਾਇਆ ਜਾਂਦਾ।

ਜਦੋਂ ਇਕ ਛੋਟੇ ਫੰਕਸ਼ਨ ਨੂੰ ਬਹੁਤ ਵਾਰ ਪ੍ਰੋਗਰਾਮ ਵਿਚ ਬੁਲਾਇਆ ਜਾਂਦਾ ਹੈ ਉਸ ਹਾਲਤ ਵਿਚ ਫੰਕਸ਼ਨ ਨੂੰ Call ਕਰਨ ਸਮੇਂ ਕੁਝ ਸਮੱਸਿਆਵਾਂ (overheads) ਆਉਂਦੀਆਂ ਹਨ।

ਵੈਲਿਊ ਪਾਸ ਕਰਨ ਲਈ, ਪਾਸਿੰਗ ਕੰਟਰੋਲ (Passing Control), ਵੈਲਿਓ ਰਿਟਰਨ ਕਰਨ ਲਈ ਅਤੇ ਕੰਟਰੋਲ ਰਿਟਰਨ ਕਰਨ ਦੇ ਲਈ ਸਮਾਂ ਦੇਣਾ ਪੈਂਦਾ ਹੈ।

ਇਹਨਾਂ ਹਲਾਤਾਂ ਵਿਚ ਲਾਗੂ ਸਮਾਂ ਬਚਾਉਣ ਲਈ C++ ਕੰਪਾਈਲਰ ਨੂੰ ਫੰਕਸ਼ਨ ਬਾਡੀ ਵਿਚ ਕੋਡ ਲਗਾਉਣ ਲਈ ਹਦਾਇਤਾਂ ਦਿੱਤੀਆਂ ਜਾਂਦੀਆਂ ਹਨ ਜੋ ਕਿ ਸਿੱਧੇ ਤੌਰ ਤੇ ਪ੍ਰੋਗਰਾਮ ਕਾਲ ਵਿਚ ਕੋਡ ਦੇ ਅੰਦਰ ਹੁੰਦੀ ਹੈ।

ਮਤਲਬ ਹਰ ਜਗ੍ਹਾ ਜਿੱਥੇ ਕਿਤੇ ਵੀ ਫੰਕਸ਼ਨ ਸੋਰਸ ਫਾਈਲ ਵਿਚ ਕਾਲ ਹੋਵੇਗਾ ਤਾਂ ਫੰਕਸ਼ਨ ਨੂੰ ਜੰਪ ਕਰਵਾਉਣ ਦੀ ਬਜਾਇ ਅਸਲ ਕੋਡ ਫੰਕਸ਼ਨ ਤੋਂ ਇਨਸਰਟ ਕੀਤਾ ਜਾਵੇਗਾ ਬਜਾਇ ਫੰਕਸ਼ਨ ਨੂੰ ਜੰਪ ਕਰਵਾਉਣ ਦੇ। ਇਹਨਾਂ ਫੰਕਸ਼ਨਜ਼ ਨੂੰ **inline functions** ਕਿਹਾ ਜਾਂਦਾ ਹੈ।

ਹੇਠ ਲਿਖਿਆ ਪ੍ਰੋਗਰਾਮ 2.8 Inline function ਦੇ ਕੰਮ ਬਾਰੇ ਦੱਸਦਾ ਹੈ।

```
# include < iostream.h >
inline void reporters (char * str)
{
    cout << end l << str ;
    exit (1) ;
}
void main ( )
{
                                    // code to open source file
if (file open is failed)
reporter (‘‘unable to open source file’’) ;
                                  // code to open target file
if (file opening failed)
reporters / unable to open target file’’);
                                     //code to copy contents of source file into target file
}
```

ਇਨਲਾਈਨ ਫੰਕਸ਼ਨ # define macros ਦੇ ਬਰਾਬਰ ਹੁੰਦੇ ਹਨ ਜਦਕਿ ਇਹ ਵਧੀਆ type checking (ਟਾਈਪ ਚੈਕਿੰਗ) ਦਿੰਦੇ ਹਨ ਅਤੇ ਨਾ ਹੀ ਇਨ੍ਹਾਂ ਦਾ ਕੋਈ ਗਲਤ ਪ੍ਰਭਾਵ ਹੁੰਦਾ ਹੈ। ਜਦੋਂ ਇਹ ਮੈਕਰੋਜ਼ ਨਾਲ ਸੰਬੰਧਿਤ ਹੁੰਦੇ ਹਨ।

2.9 ਉਦਾਹਰਣ ਵਜੋਂ ਹੇਠ ਲਿਖਿਆ ਪ੍ਰੋਗਰਾਮ

```
# includ < iostream.h >
# define SQUARE (X) x*x
inline float square (float y)
{
    return y * y ;
}
void main( )
{
    float a = 0.5, b = 0.5, c, d ;
    c = SQUARE (++ a) ;
    d = Square (++ b) ;
}
```

ਪ੍ਰੀਪ੍ਰੋਸੈਸਿੰਗ ਸਮੇਂ macro SQUARE (expand) ਹੋ ਜਾਂਦਾ ਹੈ।
C = ++x * ++x ; ਤੁਸੀਂ ਨੋਟ ਕਰੋਗੇ ਕਿ ਜਦੋਂ macro ਨੂੰ expand ਕੀਤਾ ਜਾਂਦਾ ਹੈ ਤਾਂ ਕਈ ਅਣਚਾਹੇ ਪ੍ਰਭਾਵ ਪੈਦਾ ਹੋ ਜਾਂਦੇ ਹਨ।

2.10 ਫੰਕਸ਼ਨ (Functions)

ਇਕ ਫੰਕਸ਼ਨ ਪ੍ਰੋਗਰਾਮ ਵਿਚ ਦਿੱਤੀ ਹੋਈ ਹਦਾਇਤਾਂ (Statement) ਦਾ ਇਕ ਸਮੂਹ ਹੈ ਜੋ ਕਿ ਇਕ ਖ਼ਾਸ ਕੰਮ ਨੂੰ ਕਰਨ ਲਈ ਬਣਾਇਆ ਜਾਂਦਾ ਹੈ। ਹਰ ਇਕ C++ ਪ੍ਰੋਗਰਾਮ ਵਿਚ ਘੱਟ ਤੋਂ ਘੱਟ ਇਕ ਫੰਕਸ਼ਨ ਹੁੰਦਾ ਹੈ ਤੇ ਉਹ ਹੈ main( )
ਅਸੀਂ ਆਪਣੇ ਪ੍ਰੋਗਰਾਮ ਦੇ ਕੋਡ ਨੂੰ ਅਲੱਗ–ਅਲੱਗ ਫੰਕਸ਼ਨਜ਼ ਵਿਚ ਵੰਡ ਸਕਦੇ ਹਾਂ। ਇਕ ਫੰਕਸ਼ਨ ਉਸ ਸਮੇਂ ਹੀ (execute) ਲਾਗੂ ਹੁੰਦਾ ਹੈ ਜਦੋਂ ਪ੍ਰੋਗਰਾਮ ਵਿਚ ਕਿਸੇ ਥਾਂ ਤੇ ਉਸ ਨੂੰ ਬੁਲਾਇਆ ਜਾਂਦਾ ਹੈ। ਫੰਕਸ਼ਨ ਨੂੰ ਬਣਾਉਣ ਦਾ ਤਰੀਕਾ (Format) ਹੇਠਾਂ ਲਿਖੇ ਅਨੁਸਾਰ ਹੈ :

```
type name (Parameter 1, Parameter 2, ....)
{
    statements ;
    ..... ;
}
```

ਇੱਥੇ 1. type ਦੱਸਦਾ ਹੈ ਕਿ ਫੰਕਸ਼ਨ ਕਿਸ ਤਰ੍ਹਾਂ ਦਾ ਡਾਟਾ ਵਾਪਿਸ (return) ਕਰੇਗਾ।
2. name, function ਦਾ ਨਾਂ ਹੈ।
3. Parameters ਉਹ variables ਹਨ ਜੋ ਫੰਕਸ਼ਨ ਵਿਚੋਂ ਪਾਸ ਕਰਵਾਉਣੇ ਹਨ।
4. Statement ਉਹ ਹਦਾਇਤਾਂ ਦਾ ਸਮੂਹ (group) ਹਨ ਜਿਸ ਲਈ function ਬਣਾਇਆ ਗਿਆ ਹੈ।

2.10.1 ਫੰਕਸ਼ਨ ਪ੍ਰੋਟੋਟਾਈਪਸ (Function Prototypes)

ਇਕ ਫੰਕਸ਼ਨ ਪ੍ਰੋਟੋਟਾਈਪ ਇਕ ਘੋਸ਼ਣਾ ਹੈ ਜਿਸ ਵਿਚ ਪਾਸ ਕੀਤੇ ਜਾਣ ਵਾਲੇ ਐਰਗੀਮੈਂਟਸ ਅਤੇ ਫੰਕਸ਼ਨ ਦੁਆਰਾ ਵਾਪਸ ਕੀਤੀ ਵੈਲਿਊ ਦੀ ਟਾਈਪ, ਦੋਵਾਂ ਨੂ ਪਰਭਾਸ਼ਿਤ ਕੀਤਾ ਜਾਂਦਾ ਹੈ। ਜੇਕਰ ਅਸੀਂ ਕਿਸੇ ਫੰਕਸ਼ਨ foot ( ) ਨੂੰ ਬੁਲਾਉਣਾ (call) ਹੈ ਜੋ ਕਿ float ਪ੍ਰਾਪਤ ਕਰਦਾ ਹੈ ਅਤੇ int ਨੂੰ ਆਰਗੂਮੈਂਟ ਦੇ ਤੌਰ ਤੇ ਲੈਂਦਾ ਹੈ ਅਤੇ double value ਵਾਪਿਸ ਕਰਦਾ ਹੈ ਤਾਂ ਇਸ ਦਾ ਪ੍ਰੋਟੋਟਾਈਪ ਇਸ ਤਰ੍ਹਾਂ ਹੈ—

double foot (float, int) ;

ਆਓ ਹੁਣ ਦੇਖਦੇ ਹਾਂ C++ ਵਿਚ ਫੰਕਸ਼ਨ ਕਿਸ ਤਰ੍ਹਾਂ ਪਰਿਭਾਸ਼ਿਤ ਕੀਤੇ ਜਾਂਦੇ ਹਨ, ਕੁਝ C++ ਕੰਪਾਈਲਰ ਅਜੇ ਵੀ K & R ਸਟਾਈਲ ਸਵੀਕਾਰ ਕਰਦੇ ਹਨ।
ਦੋਵੇਂ ਫਾਰਮੈਟ ਹੇਠਾਂ ਲਿਖੇ ਹਨ :

```
                        // K & R style
double foot (a, b)
int a ; float b ;
{
    // some code
}
    // Prototype - Like style
    double foot (int a, float b)
  {
    // Some code
}
```

ਨੋਟ : ਕੁਝ C++ ਕੰਪਾਈਲਰ ਦੋਵੇਂ ਸਟਾਈਲ ਸਵੀਕਾਰ ਕਰਦੇ ਹਨ।

2.10.2 ਫੰਕਸ਼ਨ ਓਵਰਲੋਡਿੰਗ (Function Overloading)

C++ ਵਿਚ ਫੰਕਸ਼ਨਜ਼ ਦੀ ਸਮਰੱਥਾ ਵਿਚ ਇਕ ਹੋਰ ਮਹੱਤਵਪੂਰਨ ਚੀਜ਼ ਜੋੜੀ ਗਈ ਹੈ ਜਿਸ ਨੂੰ ਫੰਕਸ਼ਨ ਓਵਰਲੋਡਿੰਗ ਕਿਹਾ ਜਾਂਦਾ ਹੈ। ਇਸ ਸੁਵਿਧਾ ਨਾਲ ਤੁਸੀਂ ਬਹੁਤ ਸਾਰੇ ਫੰਕਸ਼ਨ ਇੱਕੋ ਨਾਮ ਨਾਲ ਦੇਖਦੇ ਹੋ ਪਰ ਸੀ ਵਿੱਚ ਪ੍ਰੋਗਰਾਮ ਵਿਚ ਫੰਕਸ਼ਨ ਦਾ ਵੱਖਰਾ ਨਾਮ ਹੁੰਦਾ ਹੈ। ਉਦਾਹਰਣ ਦੇ ਤੌਰ ਤੇ C ਵਿਚ ਕਈ ਪ੍ਰਕਾਰ ਦੇ ਫੰਕਸ਼ਨ ਹੁੰਦੇ ਹਨ ਜੋ ਕਿ ਐਬਸੈਲਿਊਟ ਵੈਲਯੂ ਵਾਪਿਸ ਕਰਦੇ ਹਨ ਜਦਕਿ ਵੱਖਰਾ ਨਾਮ ਚਾਹੀਦਾ ਹੁੰਦਾ ਹੈ ਤਾਂ ਹਰ ਨਿਉਮੈਰਿਕ ਡਾਟਾ ਟਾਈਪ ਲਈ ਇਕ ਅਲੱਗ ਫੰਕਸ਼ਨ ਹੁੰਦਾ ਹੈ। ਇਸ ਕਰਕੇ ਤਿੰਨ ਅਲੱਗ ਫੰਕਸ਼ਨ ਹੁੰਦੇ ਹਨ ਜੋ ਐਬਸੈਲਿਊਟ ਵੈਲਯੂ ਆਰਗੂਮੈਂਟ ਦੀ ਵਾਪਿਸ ਕਰਦੇ ਹਨ।

```
int abs (int i) ;
long labs (long l) ;
double fabs (double d) ;
```

ਉੱਪਰ ਸਾਰੇ ਫੰਕਸ਼ਨ ਇਕ ਹੀ ਕੰਮ ਕਰ ਰਹੇ ਹਨ ਅਤੇ ਤਿੰਨ ਅਲੱਗ ਅਲੱਗ ਫੰਕਸ਼ਨ ਲੈਣੇ ਜ਼ਰੂਰੀ ਨਹੀਂ ਹਨ। C++ ਇਸ ਸਮੱਸਿਆ ਨੂੰ ਠੀਕ ਕਰਦੀ ਹੈ ਅਤੇ ਇਕੋ ਨਾਮ ਦੇ ਤਿੰਨ ਅਲੱਗ ਫੰਕਸ਼ਨ ਨੂੰ ਪ੍ਰੋਗਰਾਮ ਦੀ ਆਗਿਆ ਦਿੰਦੀ ਹੈ। ਇਸ ਪ੍ਰੋਸੈਸ ਨੂੰ ਫੰਕਸ਼ਨ ਓਵਰਲੋਡਿੰਗ ਕਹਿੰਦੇ ਹਨ।

ਹੇਠਾਂ ਲਿਖੇ ਪ੍ਰੋਗਰਾਮ 2.10.2 ਦੁਆਰਾ ਦੱਸਿਆ ਜਾ ਸਕਦਾ ਹੈ।

```
# include < iostream.h >
void main( )
{
    int i = – 25, j ;
    long 1 = – 100000 L, m ;
    double d = – 12.34, e ;
   int abs (int) ;
    long abs (long) ;
    double abs (double) ;
    i = abs (i) ;
    m = abs (l) ;
    e = abs (d) ;
}
cout << end 1 << j << end 1 << m << end 1 <<e ;
}
    int abs (int ii)
{
    return (ii > 0 ? ii : ii * –1) ;

long abs (logn ll)
{
    return (ll > 0 ? ll : ll * –1) ;
}
    double abs (double dd)
{
    return (dd > 0 ? dd : dd * –1) ;
}
```

C++ ਕੰਪਾਇਲਰ ਨੂੰ ਕਿਵੇਂ ਪਤਾ ਹੁੰਦਾ ਹੈ ਕਿ ਕਿਹੜਾ abs ( ) s ਨੂੰ ਬੁਲਾਇਆ ਜਾਵੇਗਾ। ਇਹ ਆਰਗੂਮੈਂਟ ਦੀ ਟਾਈਪ ਜੋ ਕਿ ਫੰਕਸ਼ਨ ਕਾਲ ਕਰਨ ਵੇਲੇ ਪਾਸ ਕੀਤੀ ਜਾਂਦੀ ਹੈ।

ਉਦਾਹਰਣ : ਜੇਕਰ int pass ਕੀਤਾ ਜਾਵੇਗਾ ਤਾਂ ਇੰਟੀਜਰ ਵਾਚਨ ਨੂੰ abs (gets) ਬੁਲਾਇਆ ਜਾਂਦਾ ਹੈ। ਜੇਕਰ double pass ਕੀਤਾ ਜਾਵੇਗਾ ਤਾਂ double version of abs (gets) ਕਾਲ ਕੀਤਾ ਜਾਵੇਗਾ।

2.11 ਫੰਕਸ਼ਨ ਦੇ ਵਿਚ ਡਿਫਾਲਟ ਆਰਗੂਮੈਂਟ

C ਵਿਚ ਜੇਕਰ ਫੰਕਸ਼ਨ ਦੋ ਆਰਗੂਮੈਂਟ ਪ੍ਰਾਪਤ ਕਰਨ ਲਈ ਪਰਿਭਾਸ਼ਿਤ ਕੀਤਾ ਜਾਵੇ, ਜਦੋਂ ਅਸੀਂ ਫੰਕਸ਼ਨ ਨੂੰ ਬੁਲਾਉਂਦੇ ਹਾਂ (Call) ਸਾਨੂੰ ਦੋ ਕੀਮਤਾਂ ਇਸ ਫੰਕਸ਼ਨ ਵਿਚ ਪਾਸ (Pass) ਕਰਨੀਆਂ ਪੈਂਦੀਆਂ ਹਨ ਜੇਕਰ ਅਸੀਂ ਇਕ ਕੀਮਤ (Value) ਪਾਸ ਕਰਦੇ ਹਾਂ ਤਾਂ ਗਾਰਬੇਜ ਕੀਮਤਾਂ (Garbage Value) ਨੂੰ ਅਖੀਰਲਾ ਆਰਗੂਮੈਂਟ ਮੰਨਿਆ ਜਾਂਦਾ ਹੈ। ਜਦੋਂ ਫੰਕਸ਼ਨ ਨੂੰ ਬੁਲਾਇਆ ਜਾਂਦਾ ਹੈ ਤਾਂ C++ ਦੇ ਫੰਕਸ਼ਨ ਵਿੱਚ ਆਰਗੁਮੈਂਟ ਲਈ ਉਹਨਾਂ ਡਿਫਾਲਟ ਕੀਮਤਾਂ (values) ਨੂੰ ਪਰਿਭਾਸ਼ਿਤ ਕਰਨ ਦੀ ਸਮਰੱਥਾ ਹੁੰਦੀ ਹੈ ਜਿਜ਼ੀਆਂ ਪਾਸ ਨਹੀ ਕੀਤੀਆਂ ਜਾਂਦੀਆਂ ਜਦੋ ਫੰਕਸ਼ਨ ਨੂੰ ਬੁਲਾਇਆ ਜਾਦਾਂ ਹੈ

ਇਸ ਉਦਾਹਰਣ ਤੋਂ ਸਪੱਸ਼ਟ ਹੋ ਜਾਵੇਗਾ :

```
# include < iostream.h >
# include < conio.h >
void box (int sr = 1, int sc = 1, int er = 24, int ec = 80) ;
void main( )
{
    clrscr ( );
    box (10, 20, 22, 70) ;
    box (10, 20, 15) ;
    box (5, 10) ;
    box ( ) ;
}
void box (int sr, int sc, int er, int ec)
{
    int r, c ;
    goto xy (sc, sr)
```

```
cout << (char) 218 ; // outputs a graphic character whose ASCII value is 128.
goto xy (ec, sr) ;
cout << (char) 191 ;
goto xy (sc, er) ;
cout << (char) 192 ;
goto xy (ec, er) ;
cout << (char) 217 ;
for (r = sr + 1 ; r < er ; r ++)
{
    goto xy (sc, r) ;
    cout << (char) 179 ;
    goto xy (ec, r) ;
    cout << (char) 179 ;
}
    for (c = sc + 1 ; c < ec ; c++)
 {
    gotoxy (c, sr) ;
    cout << (char) 196 ;
    gotoxy (c, er) ;
    cout << (char) 196 ;
 }
}
```

ਜਦੋਂ ਅਸੀਂ function box ( ) ਨੂੰ 4 ਆਰਗੂਮੈਂਟ ਨਾਲ ਬੁਲਾਉਂਦੇ ਹਾਂ ਤਾਂ ਬਾਕਸ ਆਰਗੂਮੈਂਟ ਪਾਸ ਕਰਨ ਦੇ ਨਾਲ ਪ੍ਰਾਪਤ ਹੁੰਦਾ ਹੈ। ਪਰ ਜਦੋਂ ਅਸੀਂ ਇਸ ਨੂੰ 3 ਆਰਗੂਮੈਂਟ ਨਾਲ ਬੁਲਾਉਂਦੇ ਹਾਂ ਤਾਂ ਡਿਫਾਲਟ ਕੀਮਤ Prototype of box ( ) ਵਰਤਿਆ ਜਾਂਦਾ ਹੈ।ਅਖੀਰਲੇ ਆਰਗੂਮੈਂਟ ਲਈ ਉਸੇ ਤਰ੍ਹਾਂ ਜਦੋਂ ਅਸੀਂ ਦੋ ਆਰਗੂਮੈਂਟ ਨਾਲ ਬੁਲਾਉਂਦੇ ਹਾਂ ਤਾਂ ਅਖੀਰਲੇ 2 ਆਰਗੂਮੈਂਟਾਂ ਲਈ ਡਿਫਾਲਟ ਕੀਮਤਾਂ ਵਰਤੀਆਂ ਜਾਂਦੀਆਂ ਹਨ ਅਤੇ ਅਖ਼ੀਰ ਵਿਚ ਅਸੀਂ ਇਸ ਨੂੰ ਬਿਨਾਂ ਆਰਗੂਮੈਂਟ ਤੋਂ ਬੁਲਾਉਂਦੇ ਹਾਂ ਤੇ ਇਕ box (ਬਾਕਸ) ਡਰਾਅ ਹੋ ਜਾਂਦਾ ਹੈ ਜੋ ਕਿ ਜਿਸ ਦੀਆਂ ਸਾਰੀਆਂ ਡਿਫਾਲਟ ਕੀਮਤਾਂ ਹਨ।

2.12 Extern ‘‘C’’ Declaration

C ਪ੍ਰੋਗਰਾਮਿੰਗ ਭਾਸ਼ਾ ਵਿਚ, External ਵੇਰੀਏਬਲ ਉਹ variable ਹੁੰਦਾ ਹੈ ਜੋ ਕਿ ਫੰਕਸ਼ਨ ਬਲਾਕ ਤੋਂ ਬਾਹਰ Declare ਕੀਤਾ ਜਾਂਦਾ ਹੈ। ਫੰਕਸ਼ਨ ਬਲਾਕ ਦੇ ਅੰਦਰ Declare ਕੀਤਾ ਹੋਇਆ ਵੇਰੀਏਬਲ ਲੋਕਲ (local) variable ਕਹਿਲਾਉਂਦਾ ਹੈ।
ਇਕ ਐਕਸਟਰਨਲ (external) ਵੇਰੀਏਬਲ ਨੂੰ ਪ੍ਰੋਗਰਾਮ ਦੇ ਸਾਰੇ ਫੰਕਸ਼ਨਜ਼ ਦੁਆਰਾ ਐਕਸੈਸ (Access) ਕੀਤਾ ਜਾ ਸਕਦਾ ਹੈ।ਇਹ ਇਕ ਗਲੋਬਲ (Global) ਵੇਰੀਏਬਲ (Variable) ਹੁੰਦਾ ਹੈ।ਐਕਸਟਰਨਲ ਵੇਰੀਏਬਲ ਨੂੰ ਜਦੋਂ ਪ੍ਰੋਗਰਾਮ ਸ਼ੁਰੂ ਹੁੰਦਾ ਹੈ ਉਸ ਸਮੇਂ ਹੀ ਮੈਮਰੀ ਐਲੋਕੇਟ (allocate) ਕੀਤੀ ਜਾਂਦੀ ਹੈ ਤੇ ਜਦੋਂ ਪ੍ਰੋਗਰਾਮ ਖ਼ਤਮ ਹੁੰਦਾ ਹੈ ਤਾਂ ਮੈਮਰੀ ਰੀਲੀਜ਼ (Release) ਹੋ ਜਾਂਦੀ ਹੈ।ਇਸ ਦਾ ਲਾਈਫ ਟਾਈਮ ਪ੍ਰੋਗਰਾਮ ਦੇ ਲਾਈਫ ਟਾਈਮ ਦੇ ਬਰਾਬਰ ਹੁੰਦਾ ਹੈ।

ਫਾਰਮੈਟ ਫਾਈਲ 1 :

```
int Global Variable ;   // implicit definition
void Some Function (void) ;
    // function prototype (declaration)
int main( ) {
Global Variable = 1 ;
Some Function( );
return 0 ;
}
```

ਫਾਈਲ 2 :

```
extern int Global Variable ;  // Explicit declaration
void Some Function (void) {   // Function header (definition)
++ Global Variable ;
}
```

2.13 ਰੇਫਰੈਂਸ vs ਪੁਆਇੰਟਰ (Reference vs Pointer)

ਹੇਠਾਂ ਲਿਖੇ ਕੁਝ ਪੁਆਇੰਟਰਸ ਰਾਹੀਂ ਉਪਰੋਕਤ ਵਿਸ਼ੇ ਨੂੰ ਸਮਝਾਇਆ ਜਾ ਸਕਦਾ ਹੈ :

1. ਇਕ ਪੁਆਇੰਟਰ ਨੂੰ ਅਸੀਂ ਜਦੋਂ ਮਰਜੀ ਰੀ–ਅਸਾਈਨਡ (re-assigned) ਕਰ ਸਕਦੇ ਹਾਂ ਪਰ ਰੇਫਰੈਨਸ (reference) ਇਕ ਵਾਰ ਇਨਸ਼ਿਲਾਈਜਡ (initialized) ਹੋਣ ਤੋਂ ਬਾਅਦ ਰੀ–ਅਸਾਈਨਡ ਨਹੀਂ ਹੁੰਦਾ।
2. ਇਕ ਪੁਆਇੰਟਰ NULL ਦੀ ਤਰਫ਼ ਇਸ਼ਾਰਾ (Point) ਕਰ ਸਕਦਾ ਹੈ ਪਰ ਰੇਫਰੈਂਸ (Reference) ਕਦੇ ਵੀ NULL ਦੀ ਤਰਫ ਪੁਆਇੰਟ ਨਹੀਂ ਕਰਦਾ।
3. ਅਸੀਂ ਪੁਆਇੰਟਰ ਦੀ ਤਰ੍ਹਾਂ ਰੇਫਰੈਂਸ (reference) ਦਾ ਐਡਰੈਸ (Address) ਪ੍ਰਾਪਤ ਨਹੀਂ ਕਰ ਸਕਦੇ।
4. ਇਥੇ ਕੋਈ (Reference Arithmatic) ਨਹੀਂ ਹੁੰਦਾ ਪਰ ਅਸੀਂ ਉਸ ਆਬਜੈਕਟ (Object) ਦਾ ਐਡਰੈਸ ਲੈ ਸਕਦੇ ਹਾਂ ਜਿਸ ਨੂੰ reference ਪੁਆਇੰਟ ਕਰ ਰਿਹਾ ਹੈ ਤੇ ਉਸ ਤੇ 'Pointer Arithmetic' ਕਰ ਸਕਦੇ ਹਾਂ।

2.14 Memory Allocation using New operator and Deallocation using Delete operator

ਰਨ ਟਾਈਮ ਤੇ ਡਾਇਨਾਮਿਕ ਤਰੀਕੇ ਨਾਲ ਮੈਮਰੀ ਐਲੋਕੇਟ ਕਰਨ ਵਾਸਤੇ 'C' malloc( ) ਅਤੇ calloc( ) ਫੰਕਸ਼ਨ ਦੀ ਵਰਤੋਂ ਕਰਦਾ ਹੈ। ਨਾਲ ਹੀ ਇਹ free( ) ਫੰਕਸ਼ਨ ਦੀ ਵਰਤੋਂ ਡਾਇਨਾਮਿਕ ਤਰੀਕੇ ਨਾਲ ਐਲੋਕੇਟ ਕੀਤੀ ਹੋਈ ਮੈਮਰੀ ਨੂੰ free ਕਰਨ ਲਈ ਕਰਦਾ ਹੈ। ਅਸੀਂ ਡਾਇਨਾਮਿਕ ਮੈਮਰੀ ਤਕਨੀਕ ਤਾਂ ਵਰਤਦੇ ਹਾਂ ਜਦੋਂ ਇਹ ਪਤਾ ਨਹੀਂ ਹੁੰਦਾ ਕਿ ਕਿੰਨੇ ਮੈਮਰੀ ਸਪੇਸ ਦੀ ਜ਼ਰੂਰਤ ਹੈ। ਹਾਲਾਂਕਿ 'C++' ਵੀ ਇਨ੍ਹਾਂ ਫੰਕਸ਼ਨਸ ਨੂੰ ਸਮਰਥਨ (Support) ਕਰਦਾ ਹੈ ਪਰ ਇਹ ਮੈਮਰੀ ਐਲੋਕੇਸ਼ਨ ਦਾ ਕੰਮ ਹੋਰ ਬੇਹਤਰ ਤਰੀਕੇ ਦੇ ਨਾਲ ਕਰਨ ਲਈ ਦੋ ਨਵੇਂ ਯੂਨਰੀ (Unary) ਉਪਰੇਟਰ new ਅਤੇ delete ਨੂੰ ਵੀ ਪਰਿਭਾਸ਼ਿਤ ਕਰਦਾ ਹੈ। new ਅਤੇ delete ਉਪਰੇਟਰ ਮੈਮਰੀ ਐਲੋਕੇਸ਼ਨ ਅਤੇ ਫਰੀ (free) ਕਰਨ ਦਾ ਕੰਮ ਵਧੀਆ ਅਤੇ ਸੌਖੇ ਢੰਗ ਨਾਲ ਕਰਦੇ ਹਨ।

ਇਕ ਆਬਜੈਕਟ (Object) new ਉਪਰੇਟਰ ਦੀ ਵਰਤੋਂ ਕਰਕ ਬਣਾਇਆ ਜਾ ਸਕਦਾ ਹੈ ਤੇ ਜਦੋਂ ਲੋੜ ਹੋਵੇ ਤਾਂ delete operator (ਉਪਰੇਟਰ) ਦੀ ਵਰਤੋਂ ਕਰਕੇ ਮਿਟਾਇਆ (Destroyed) ਜਾ ਸਕਦਾ ਹੈ (ਜਦੋਂ ਲੋੜ ਹੋਵੇ ਤਾਂ)। new operator (ਉਪਰੇਟਰ) ਦੇ ਨਾਲ ਕਰੀਏਟ ਕੀਤਾ ਹੋਇਆ ਡਾਟਾ ਆਬਜੈਕਟ ਉਸ ਸਮੇਂ ਤੱਕ ਹੋਂਦ ਵਿਚ ਰਹਿੰਦਾ ਹੈ ਜਦੋਂ ਤੱਕ ਕਿ delete ਉਪਰੇਟਰ ਦੁਆਰਾ ਉਸ ਨੂੰ ਬਾਹਰੀ ਤੌਰ ਤੇ ਨਸ਼ਟ (Destroy) ਨਹੀਂ ਕਰ ਦਿੱਤਾ ਜਾਂਦਾ। ਇਸ ਤਰ੍ਹਾਂ ਇਕ ਆਬਜੈਕਟ ਦੇ ਜੀਵਨ ਸਮੇਂ (Life time) ਦਾ ਕੰਟਰੋਲ ਸਿੱਧੇ ਤੌਰ ਤੇ ਸਾਡੇ ਹੱਥ ਵਿਚ ਰਹਿੰਦਾ ਹੈ ਤੇ ਇਸ ਦਾ ਪ੍ਰੋਗਰਾਮ ਦੇ ਬਲਾਕ ਸਟਰਕਚਰ ਨਾਲ ਕੋਈ ਸੰਬੰਧ ਨਹੀਂ ਹੁੰਦਾ।

new ਉਪਰੇਟਰ ਦੀ ਵਰਤੋਂ ਹੇਠ ਲਿਖੇ ਤਰੀਕੇ ਨਾਲ ਕੀਤੀ ਜਾ ਸਕਦੀ ਹੈ :

Pointer – Variable = new data-type ;

new ਉਪਰੇਟਰ ਡਾਟਾ ਆਬਜੈਕਟ ਲਈ ਲੋੜ ਅਨੁਸਾਰ ਮੈਮਰੀ ਐਲੋਕੇਟ ਕਰ ਦਿੰਦਾ ਹੈ ਤੇ ਆਬਜੈਕਟ ਦਾ ਐਡਰੈਸ (Address) ਵਾਪਿਸ ਕਰਦਾ ਹੈ। ਡਾਟਾ ਟਾਈਪ ਕੋਈ ਵੀ ਵੇਲਿਡ ਡਾਟਾ–ਟਾਈਪ ਹੋ ਸਕਦੀ ਹੈ। Pointer-Variable ਪੁਆਇੰਟਰ ਵੇਰੀਏਬਲ ਐਲੋਕੇਟ ਕੀਤੇ ਹੋਏ ਮੈਮਰੀ ਸਪੇਸ ਦਾ ਐਡਰੇਸ ਰੱਖਦਾ ਹੈ।

ਉਦਾਹਰਣ 2.14 :

p = new float ;
q = new int ;

ਇੱਥੇ *p,* float ਟਾਈਪ ਦਾ ਪੁਆਇੰਟਰ ਹੈ ਤੇ *q,* int ਟਾਈਪ ਦਾ ਪੁਆਇੰਟਰ ਹੈ।

ਅਸੀਂ New ਉਪਰੇਟਰ ਦੀ ਵਰਤੋਂ ਕਰਕੇ ਮੈਮਰੀ ਨੂੰ ਇਨਿਸ਼ਲਾਈਜ਼ਡ (intialized) ਵੀ ਕਰ ਸਕਦੇ ਹਾਂ। ਜਿਵੇਂ ਕਿ ਹੇਠਾਂ ਦਿਖਾਇਆ ਗਿਆ ਹੈ :

Pointer – Variable = new data-type (Value)

ਜਿਵੇਂ ਕਿ—

int * *p*= new int (25)
float * *q* = new float (7.5) ;

new ਉਪਰੇਟਰ ਨੂੰ ਅਸੀਂ ਕਿਸੇ ਵੀ ਡਾਟਾ–ਟਾਈਪ ਲਈ ਮੈਮਰੀ ਸਪੇਸ ਕਰੀਏਟ ਕਰਨ ਲਈ ਵੀ ਵਰਤ ਸਕਦੇ ਹਾਂ (user-defined ਡਾਟਾ ਟਾਈਪਸ ਲਈ ਵੀ।) ਜਿਵੇਂ ਕਿ arrays, structures ਅਤੇ classes ਇਕ ਡਾਇਮੇਨਸ਼ਨਲ (dimensional) ਐਰੇ ਲਈ ਮੈਮਰੀ ਸਪੇਸ ਬਣਾਉਣ (create) ਲਈ ਤਰੀਕਾ ਹੇਠ ਲਿਖੇ ਅਨੁਸਾਰ ਹੈ :

int * *p* = new int [10]

ਮਲਟੀ ਡਾਈਮੇਨਸ਼ਨਲ ਐਰੇ ਲਈ :

```
array_ptr = new int [3] [5] [4] ;     //legal
array_ptr = new int [m] [5] [4] ;     //legal
array_ptr = new int [3] [5] [ ] ;     //illegal
array_ptr = new int [ ] [5] [4] ;     //illegal
```

ਜਦੋਂ ਡਾਟਾ ਆਬਜੈਕਟ ਦੀ ਲੋੜ ਨਹੀਂ ਰਹਿੰਦੀ ਤਾਂ ਮੈਮਰੀ ਸਪੇਸ ਨੂੰ ਫਰੀ (free) ਕਰਵਾਉਣ ਲਈ delete ਉਪਰੇਟਰ ਦੀ ਵਰਤੋਂ ਕੀਤੀ ਜਾਂਦੀ ਹੈ।

ਇਸ ਦਾ ਤਰੀਕਾ ਹੇਠਾਂ ਲਿਖੇ ਅਨੁਸਾਰ ਹੈ—

delete pointer-variable

ਇੱਥੇ ਪੁਆਇੰਟਰ ਵੇਰੀਏਬਲ ਉਹ ਪੁਆਇੰਟਰ ਹੈ ਜੋ new ਉਪਰੇਟਰ ਦੀ ਵਰਤੋਂ ਕਰਕੇ ਬਣਾਏ ਗਏ ਡਾਟਾ ਆਬਜੈਕਟ ਦੀ ਤਰਫ ਪੁਆਇੰਟ ਕਰਦਾ ਹੈ।
ਜਿਵੇਂ ਕਿ

delete *p* ;
delete *q* ;

ਜੇਕਰ ਅਸੀਂ ਡਾਇਨਾਮਿਕ ਤਰੀਕੇ ਨਾਲ ਐਲੋਕੇਟ ਹੋਏ ਐਰੇ ਦੀ ਮੈਮਰੀ ਨੂੰ ਫਰੀ (free) ਕਰਵਾਉਣਾ ਚਾਹੁੰਦੇ ਹਾਂ ਤਾਂ ਹੇਠਾਂ ਲਿਖਿਆ ਤਰੀਕਾ ਅਪਣਾਉਣਾ ਪਵੇਗਾ—

delete [size] pointer_variable ;

2.15 ਆਈ.ਉ. ਸਟਰੀਮਸ (I.O. Streams)

ਅਸੀਂ ਆਪਣੇ ਪ੍ਰੋਗਰਾਮ ਵਿਚ ਹੇਠਾਂ ਲਿਖਿਆ # include directive ਵਰਤਦੇ ਹਾਂ :

include < iostream >

ਇਹ directive, preprocessor ਇਕ iostream ਫਾਈਲ ਦੇ contents ਦਾਖਲ ਕਰਨ ਲਈ ਕਹਿੰਦਾ ਹੈ। ਇਸ ਵਿਚ ''cout ਤੇ ਉਪਰੇਟਰ <<'' ਅਤੇ ''cin ਤੇ ਉਪਰੇਟਰ >>'' ਦੀ ਡਿਕਲੇਅਰੇਸ਼ਨ (declaration) ਸ਼ਾਮਲ ਹੁੰਦੀਆਂ ਹਨ।
ਹੇਡਰ ਫਾਈਲ (header file) iostream ਉਨ੍ਹਾਂ ਸਾਰੇ ਪ੍ਰੋਗਰਾਮ ਦੀ ਸ਼ੁਰੂਆਤ ਵਿਚ ਲਿਖਣੀ ਚਾਹੀਦੀ ਹੈ ਜੋ Input/Output ਸਟੇਟਮੈਂਟ ਦੀ ਵਰਤੋਂ ਕਰ ਰਹੇ ਹੁੰਦੇ ਹਨ।

2.16 Comparison of cout & cin with printf & scanf

ਅਸੀਂ ਜਾਣਦੇ ਹਾਂ ਕਿ C++, C ਭਾਸ਼ਾ ਦਾ ਸੁਪਰ ਸੈਟ ਹੈ, ਜੋ ਕੁਝ ਕੰਮ ਅਸੀਂ C ਵਿਚ ਕਰਦੇ ਹਾਂ ਅਤੇ ਜਿਨ੍ਹਾਂ ਚੀਜ਼ਾਂ ਦੀ ਵਰਤੋਂ ਅਸੀਂ C ਵਿਚ ਕਰਦੇ ਹਾਂ, ਅਸੀਂ C++ ਵਿਚ ਵੀ ਉਸ ਨੂੰ ਵਰਤ ਸਕਦੇ ਹਾਂ।

printf ਅਤੇ scanf

printf( ) C ਵਿਚ ਵਰਤੀਆਂ ਜਾਣ ਵਾਲਾ ਇਕ ਆਊਟਪੁਟ function ਹੈ। scanf( ) ਇਕ ਇਨਪੁਟ ਫੰਕਸ਼ਨ ਹੈ। ਇਹ ਦੋਨੋਂ ਫੰਕਸ਼ਨ header file < stdio.h > ਵਿਚ ਪਰਿਭਾਸ਼ਿਤ ਹੁੰਦੇ ਹਨ। ਇਨ੍ਹਾਂ ਦੋਨੋਂ ਫੰਕਸ਼ਨਸ ਦੀ ਵਰਤੋਂ ਅਸੀਂ C++ ਵਿਚ ਵੀ ਕਰ ਸਕਦੇ ਹਾਂ।
C++ ਵਿਚ ਦੋ ਹੋਰ ਫੰਕਸ਼ਨ ਇਸ ਕੰਮ ਲਈ ਵਰਤੇ ਜਾਂਦੇ ਹਨ cout ਅਤੇ cin.
ਇਹ ਦੋਨੋਂ ਫੰਕਸ਼ਨ >> (Input) ਅਤੇ << (Output) operator ਦੇ ਨਾਲ ਕੰਮ ਕਰਦੇ ਹਨ।

ਨੋਟ : ਜੇਕਰ ਸਪੀਡ ਦੇ ਬਾਰੇ ਗੱਲ ਕਰੀਏ ਤਾਂ printf ( ) ਤੇ scanf( ) ਦੀ ਸਪੀਡ cout ਅਤੇ cin ਨਾਲੋਂ ਜ਼ਿਆਦਾ ਹੈ। ਪਰ Formatting ਕਰਦੇ ਸਮੇਂ printf ਅਤੇ scanf ਵਿਚ ਕੰਮ ਗੁੰਝਲਦਾਰ ਹੋ ਜਾਂਦਾ ਹੈ।
cout ਅਤੇ cin ਵਿਚ ਇਸ ਤਰ੍ਹਾਂ ਦੀ ਕੋਈ ਮੁਸ਼ਕਿਲ ਫਾਰਮੈਟਿੰਗ (Formatting) ਸਮੇਂ ਤੇ ਨਹੀਂ ਆਉਂਦੀ।

ਯਾਦ ਰੱਖਣ ਯੋਗ ਗੱਲਾਂ (Points to Remember) :

1. C++ ਇਕ ਆਬਜੈਕਟ ਓਰੀਐਂਟਿਡ ਭਾਸ਼ਾ (OOP) ਹੈ।
2. C++ ਵਿਚ cin ਦੇ ਨਾਲ >> ਉਪਰੇਟਰ (Input operator) ਦੀ ਵਰਤੋਂ ਕੀਤੀ ਜਾਂਦੀ ਹੈ।
3. C++ ਵਿਚ cout ਦੇ ਨਾਲ << ਉਪਰੇਟਰ (Output operator) ਦੀ ਵਰਤੋਂ ਕੀਤੀ ਜਾਂਦੀ ਹੈ।
4. C++ ਵਿਚ ਵੇਰੀਏਬਲ ਨੂੰ ਘੋਸ਼ਿਤ (Declare) ਕਰਨਾ ਜ਼ਰੂਰੀ ਹੈ।
5. C ਦੇ ਸਾਰੇ ਉਪਰੇਟਰ C++ ਵਿਚ ਉਪਲਬਧ ਹਨ।
6. ਇਕ ਫੰਕਸ਼ਨ ਪ੍ਰੋਗਰਾਮ ਵਿਚ ਦਿੱਤੀ ਹੋਈ ਹਦਾਇਤਾਂ ਦਾ ਇਕ ਸਮੂਹ ਹੈ ਜੋ ਕਿ ਇਕ ਖ਼ਾਸ ਕੰਮ ਨੂੰ ਕਰਨ ਲਈ ਬਣਾਇਆ ਜਾਂਦਾ ਹੈ।
7. ਇਨਸਾਈਨ ਫੰਕਸ਼ਨ # define macros ਦੇ ਬਰਾਬਰ ਹੁੰਦੇ ਹਨ।
8. ਫੰਕਸ਼ਨ ਬਲਾਕ ਦੇ ਅੰਦਰ ਘੋਸ਼ਿਤ ਕੀਤੇ ਹੋਏ ਵੇਰੀਏਬਲ ਨੂੰ ਲੋਕਲ ਵੇਰੀਏਬਲ ਕਿਹਾ ਜਾਂਦਾ ਹੈ।
9. ਫੰਕਸ਼ਨ ਬਲਾਕ ਦੇ ਬਾਹਰ ਘੋਸ਼ਿਤ ਕੀਤੇ ਹੋਏ ਵੇਰੀਏਬਲ ਨੂੰ ਗਲੋਬਲ ਵੇਰੀਏਬਲ ਕਿਹਾ ਜਾਂਦਾ ਹੈ।
10. new ਅਤੇ delete ਉਪਰੇਟਰ ਮੈਮਰੀ ਐਲੋਕੇਸ਼ਨ ਅਤੇ ਡੀ–ਐਲੋਕੇਸ਼ਨ ਦਾ ਕੰਮ ਕਰਦੇ ਹਨ।

ਅਭਿਆਸ (Exercise)

1. ਖਾਲੀ ਥਾਵਾਂ ਭਰੋ—

1. ..... ਉਪਰੇਟਰ ਨੂੰ ਆਊਟਪੁਟ (Output) ਉਪਰੇਟਰ ਕਿਹਾ ਜਾਂਦਾ ਹੈ।
2. ਇਨਲਾਈਨ ਫੰਕਸ਼ਨ ..... ਦੇ ਬਰਾਬਰ ਹੁੰਦੇ ਹਨ।
3. C++ ਦੇ ਫੰਕਸ਼ਨਜ਼ ਵਿਚ ਆਰਗੂਮੈਂਟ ਲਈ ..... ਪਰਿਭਾਸ਼ਿਤ ਕਰਨ ਦੀ ਸਮਰੱਥਾ ਹੁੰਦੀ ਹੈ।
4. new ਉਪਰੇਟਰ ਦੀ ਵਰਤੋਂ ..... ਲਈ ਕੀਤੀ ਜਾਂਦੀ ਹੈ।
5. C ਅਤੇ C++ ਕੰਪਾਈਲਰ ਸਾਰੀ ..... ਡਾਟਾ ਟਾਈਪ ਨੂੰ ਸਮਰਥਨ ਦਿੰਦਾ ਹੈ।

2. ਪ੍ਰਸ਼ਨਾਂ ਦੇ ਉੱਤਰ ਦਿਉ—

1. C++ ਭਾਸ਼ਾ ਦੇ ਇਤਿਹਾਸ ਬਾਰੇ ਜਾਣਕਾਰੀ ਦਿਉ।
2. C++ ਵਿਚ ਵੇਰੀਏਬਲ ਡੈਕਲੇਰੇਸ਼ਨ ਕਿਵੇਂ ਕੀਤੀ ਜਾਂਦੀ ਹੈ ?
3. C++ ਵਿਚ ਵਰਤੇ ਜਾਣ ਵਾਲੇ ਉਪਰੇਟਰਾਂ ਬਾਰੇ ਦੱਸੋ।
4. ਇਨਲਾਈਨ ਫੰਕਸ਼ਨ ਤੇ ਇਕ ਨੋਟ ਲਿਖੋ।
5. ਫੰਕਸ਼ਨ ਓਵਰਲੋਡਿੰਗ ਕੀ ਹੈ ?
6. ਰੈਫਰੈਂਸ (Reference) ਅਤੇ ਪੁਆਇੰਟਰ ਵਿਚ ਅੰਤਰ ਲਿਖੋ।
7. new ਉਪਰੇਟਰ ਦੀ ਵਰਤੋਂ ਕਰਕੇ ਮੈਮਰੀ ਐਲੋਕੇਸ਼ਨ ਦੀ ਪ੍ਰਕ੍ਰਿਆ ਦਾ ਵਰਣਨ ਕਰੋ।
8. cout ਅਤੇ cin ਵਿਚ ਅੰਤਰ ਲਿਖੋ।

Lab Activity

Program 1:

```
// my first program in C++

#include <iostream.h>
using namespace std;

int main()
{
cout<< "Hello World!";
return 0;
}
```

Program2:

```
// my second program in C++

#include <iostream.h>

using namespace std;

int main()
{
cout<<"Hello World! ";
cout<<"I'm a C++ program";
return 0;
}
```

Program 3:

```
/* my second program in C++

with more comments */

#include <iostream.h>
using namespace std;

int main()
{
cout<<"Hello World! ";     // prints Hello World!
cout<<"I'm a C++ program"; // prints I'm a C++ program
return 0;
}
```

Program 4:

```
// operating with variables

#include <iostream.h>
using namespace std;

int main()
{
// declaring variables:
```

```
int a, b;
int result;

// process:
  a = 5;
  b = 2;
  a = a + 1;
  result = a - b;

// print out the result:
cout<< result;

// terminate the program:
return 0;
}
```

Program 5:

```
// initialization of variables

#include <iostream.h>
using namespace std;

int main()
{
int a=5;               // initial value = 5
int b(2);              // initial value = 2
int result;            // initial value undetermined

  a = a + 3;
result = a - b;
cout<< result;

return 0;
}
```

Program 6:

```
// defined constants: calculate circumference

#include <iostream.h>
using namespace std;

#define PI 3.14159
#define NEWLINE '\n'

int main()
{
double r=5.0;               // radius
double circle;

  circle = 2 * PI * r;
cout<< circle;
cout<< NEWLINE;

return 0;
}
```

Program 7:

```
// assignment operator

#include <iostream.h>
using namespace std;

int main()
{
int a, b;         // a:?,  b:?
  a = 10;           // a:10, b:?
  b = 4;            // a:10, b:4
  a = b;            // a:4,  b:4
  b = 7;            // a:4,  b:7

cout<<"a:";
cout<< a;
```

```
cout<<" b:";
cout<< b;

return 0;
}
```

Program 8:

```
// compound assignment operators

#include <iostream.h>
using namespace std;

int main()
{
int a, b=3;
  a = b;
a+=2;              // equivalent to a=a+2
cout<< a;
return 0;
}
```

Program 9:

```
// conditional operator

#include <iostream.h>
using namespace std;

int main()
{
inta,b,c;

  a=2;
  b=7;
  c = (a>b) ?a : b;

cout<< c;

return 0;
}
```

Program 10:

```
/ i/o example

#include <iostream.h>
using namespace std;

int main()
{
int i;
cout<<"Please enter an integer value: ";
cin>> i;
cout<<"The value you entered is "<< i;
cout<<" and its double is "<< i*2 <<".\n";
return 0;
}
```

Program 11:

```
// function example
#include <iostream.h>
using namespace std;
int addition (int a, int b)
{
int r;
  r=a+b;
return (r);
}

int main()
{
int z;
  z = addition (5,3);
cout<<"The result is "<< z;
return 0;
}
```

Program 12

// function example

```
#include <iostream.h>
using namespace std;

int subtraction (int a, int b)
{
int r;
  r=a-b;
return (r);
}

int main()
{
int x=5, y=3, z;
  z = subtraction (7,2);
cout<<"The first result is "<< z <<'\n';
cout<<"The second result is "<< subtraction (7,2) <<'\n';
cout<<"The third result is "<< subtraction (x,y) <<'\n';
  z= 4 + subtraction (x,y);
cout<<"The fourth result is "<< z <<'\n';
return 0;
}
```

Program 13

```
// void function example
#include <iostream.h>
using namespace std;

void printmessage()
{
cout<<"I'm a function!";
}

int main()
{
printmessage();
return 0;
}
```

Program 14

```
// passing parameters by reference
#include <iostream.h>
using namespace std;

void duplicate(int& a, int& b, int& c)
{
  a*=2;
  b*=2;
  c*=2;
}


int main()
{
int x=1, y=3, z=7;
  duplicate (x, y, z);
cout<< "x=" << x << ", y=" << y << ", z=" << z;
return 0;
}
```

Program 15:

```
// more than one returning value
#include <iostream.h>
using namespace std;

voidprevnext(int x, int&prev, int& next)
{
```

```
prev = x-1;
next = x+1;
}

int main()
{
int x=100, y, z;
prevnext (x, y, z);
cout<<"Previous="<< y <<", Next="<< z;
return 0;
}
```

Program 16

```
// default values in functions
#include <iostream.h>
using namespace std;

int divide(int a, int b=2)
{
int r;
  r=a/b;
return (r);
}

int main()
{
cout<< divide (12);
cout<<endl;
cout<< divide (20,4);
return 0;
}
```

Program 17

```
// overloaded function
#include <iostream.h>
using namespace std;

int operate(int a, int b)
{
return (a*b);
}

float operate(float a, float b)
{
return (a/b);
}

int main()
{
int x=5,y=2;
float n=5.0,m=2.0;
cout<< operate (x,y);
cout<<"\n";
cout<< operate (n,m);
cout<<"\n";
return 0;
}
```

Program 18

```
// factorial calculator
#include <iostream.h>
using namespace std;

long factorial(long a)
{
if (a > 1)
return (a * factorial (a-1));
else
return(1);
}

int main()
{
```

```
long number;
cout<<"Please type a number: ";
cin>> number;
cout<< number <<"! = "<< factorial (number);
return 0;
}
```

Program 19

```
// declaring functions prototypes
#include <iostream.h>
using namespace std;

void odd(int a);
void even(int a);

int main()
{
int i;
do {
cout<<"Type a number (0 to exit): ";
cin>> i;
odd (i);
  } while (i!=0);
return 0;
}

void odd(int a)
{
if ((a%2)!=0) cout<<"Number is odd.\n";
else even(a);
}

void even(int a)
{
if ((a%2)==0) cout<<"Number is even.\n";
else odd(a);
}
```

Program 20

```
/ rememb-o-matic using new and delete operator
#include <iostream.h>
#include <new>
using namespace std;

int main()
{
int i,n;
int * p;
cout<<"How many numbers would you like to type? ";
cin>> i;
  p= new (nothrow) int[i];
if (p == 0)
cout<<"Error: memory could not be allocated";
else
  {
for (n=0; n<i; n++)
    {
cout<<"Enter number: ";
cin>> p[n];
    }
cout<<"You have entered: ";
for (n=0; n<i; n++)
cout<< p[n] <<", ";
delete[] p;
  }
return 0;
}
```

Exercise

1. Point out the errors, if any, in the following programs
   a.
```
void main()
{
int  a=30;
f();
}
void f()
{
int b=20;
}
```
   b.
```
#include <iostream.h>
void main()
{
void fun1(void);
void fun2(void);
fun1();
}
void fun1 (void)
{
fun2();
cout<<endl<<"hi…Hello";
}
void fun2(void)
{
cout<<endl<<"to you";
}
```

2. Show the output of the following Programs
   a.
```
#include <iostream.h>
void main()
{
floatr,a;
const float PI=3.14;
cin>>r;
a=PI*r*r;
cout<<endl<<"area of circle="<<a;
}
```
   b.
```
#include <iostream.h>
inline float mul(float x, float y)
{
        return(x*y);
}
inline double div(double p, double q)
{
        return (p/q);
}
int main()
{
float a=12.345;
```

```
float b=9.82;
cout<<mul(a,b) <<“\n”;
cout<<div(a,b) <<“\n”;
return 0;
}
```

ਪਾਠ 3

C++ ਕਲਾਸ ਕਾਨਸੈਪਟ

3.1 C++ Class Concept

ਕਲਾਸ ਇਕ ਤਰੀਕਾ ਹੈ ਜਿਸ ਦੁਆਰਾ ਅਸੀਂ ਡਾਟਾ ਅਤੇ ਉਸ ਨਾਲ ਸੰਬੰਧਿਤ ਫੰਕਸ਼ਨਜ਼ ਨੂੰ ਇਕੱਠਾ ਜੋੜਦੇ ਹਾਂ ਜੇ ਜ਼ਰੂਰੀ ਹੋਵੇ ਤਾਂ ਇਹ ਡਾਟਾ ਅਤੇ ਫੰਕਸ਼ਨਜ਼ ਨੂੰ ਬਾਹਰੀ ਵਰਤੋਂ ਤੋਂ ਛੁਪਾਉਣ ਦੀ ਆਗਿਆ ਦਿੰਦਾ ਹੈ। ਜਦੋਂ ਕਲਾਸ ਨੂੰ ਪਰਿਭਾਸ਼ਿਤ ਕੀਤਾ ਜਾਂਦਾ ਹੈ ਤਾਂ ਅਸੀਂ ਇਕ ਨਵੀਂ ਐਬਸਟਰੈਕਟ ਡਾਟਾ ਟਾਈਪ ਕਰੀਏਟ ਕਰਦੇ ਹਾਂ ਜੋ ਕਿ ਹੋਰ ਬਿਲਟ-ਇਨ ਡਾਟਾ ਟਾਈਪ ਦੀ ਤਰ੍ਹਾਂ ਹੁੰਦੀ ਹੈ।

ਕਲਾਸ ਨੂੰ ਡਿਕਲੇਅਰ ਕਰਨ ਦਾ ਤਰੀਕਾ——

```
class class name
{
    private:
variable declarations ;
    function declarations ;
public ;
variable declarations ;
    function declarations;
}
```

ਕਲਾਸ (class) ਦੀ ਬਾਡੀ (body) ਨੂੰ ਬਰੈਕਟਾਂ ਵਿਚ ਲਿਖਿਆ ਜਾਂਦਾ ਹੈ ਅਤੇ ਉਸ ਨੂੰ ਸੈਮੀ ਕਾਲਮ; ਨਾਲ ਖ਼ਤਮ ਕੀਤਾ ਜਾਂਦਾ ਹੈ। ਕਲਾਸ ਬਾਡੀ ਵਿਚ ਵੇਰੀਏਬਲ ਅਤੇ ਫੰਕਸ਼ਨਜ਼ ਨੂੰ ਡਿਕਲੇਅਰ ਕੀਤਾ ਜਾਂਦਾ ਹੈ। ਇਹਨਾਂ ਵੇਰੀਏਬਲ ਅਤੇ ਫੰਕਸ਼ਨਜ਼ ਨੂੰ ਕਲਾਸ ਮੈਂਬਰਜ਼ (class members) ਕਿਹਾ ਜਾਂਦਾ ਹੈ।

ਇਨ੍ਹਾਂ ਨੂੰ 2 ਭਾਗਾਂ ਵਿਚ ਵੰਡਿਆ ਗਿਆ ਹੈ—— ਪ੍ਰਾਈਵੇਟ (private) ਅਤੇ ਪਬਲਿਕ (public) ਜਿਸ ਤੋਂ ਦਰਸਾਇਆ ਜਾਂਦਾ ਹੈ ਕਿ ਕਿਹੜੇ ਮੈਂਬਰ ਪ੍ਰਾਈਵੇਟ (private) ਦੇ ਅਤੇ ਕਿਹੜੇ ਮੈਂਬਰ ਪਬਲਿਕ (public) ਹਨ। ਪ੍ਰਾਈਵੇਟ ਅਤੇ ਪਬਲਿਕ ਕੀ ਵਰਡਜ਼ ਨੂੰ ਅਸੈਸ ਸਪੈਸੀਫਾਇਰ ਕਿਹਾ ਜਾਂਦਾ ਹੈ।

ਜਿਹੜੇ ਕਲਾਸ ਮੈਂਬਰਜ ਨੂੰ ਪ੍ਰਾਈਵੇਟ ਡਿਕਲੇਅਰ ਕੀਤਾ ਜਾਂਦਾ ਹੈ, ਉਹ ਕਲਾਸ ਦੇ ਨਾਲ ਹੀ ਅਸੈਸ ਕੀਤੇ ਜਾਂਦੇ ਹਨ। ਜਦਕਿ ਪਬਲਿਕ ਮੈਂਬਰਜ਼ ਕਲਾਸ ਦੇ ਬਾਹਰ ਤੋਂ ਵੀ ਅਸੈਸ ਕੀਤੇ ਜਾ ਸਕਦੇ ਹਨ।

ਪ੍ਰਾਈਵੇਟ ਕੀ ਵਰਡ ਦੀ ਵਰਤੋਂ ਚੋਣਵੀਂ ਹੁੰਦੀ ਹੈ। ਕਲਾਸ ਦੇ ਮੈਂਬਰਜ਼ ਜੋ ਕਿ ਡਿਫਾਲਟ ਹੀ ਪ੍ਰਾਈਵੇਟ ਹੁੰਦੇ ਹਨ।

3.1.1 ਡਾਟਾ ਮੈਂਬਰਜ ਅਤੇ ਮੈਂਬਰ ਫੰਕਸ਼ਨਜ਼—— ਜਿਹੜੇ ਵੇਰੀਏਬਲ ਕਲਾਸ ਦੇ ਅੰਦਰ ਡਿਕਲੇਅਰ ਕੀਤੇ ਜਾਂਦੇ ਹਨ ਉਹਨਾਂ ਨੂੰ ਡਾਟਾ ਮੈਂਬਰਜ਼ ਕਿਹਾ ਜਾਂਦਾ ਹੈ ਅਤੇ ਫੰਕਸ਼ਨਜ਼ ਨੂੰ ਮੈਂਬਰ ਫੰਕਸ਼ਨਜ਼ ਕਿਹਾ ਜਾਂਦਾ ਹੈ। ਸਿਰਫ ਮੈਂਬਰਜ਼ ਫੰਕਸ਼ਨਜ਼ ਹੀ ਪ੍ਰਾਇਵੇਟ ਡਾਟਾ ਮੈਂਬਰਜ਼ ਅਤੇ ਪ੍ਰਾਈਵੇਟ ਫੰਕਸ਼ਨਜ਼ ਅਸੈਸ ਕਰ ਸਕਦੇ ਹਨ।

3.1.2 ਆਬਜੈਕਟ ਕਰੀਏਟ ਕਰਨਾ—— ਇਕ ਵਾਰ ਜਦੋਂ ਕਲਾਸ ਡਿਕਲੇਅਰ ਕਰ ਦਿੱਤੀ ਜਾਂਦੀ ਹੈ ਤਾਂ ਅਸੀਂ class name ਦੀ ਵਰਤੋਂ ਕਰਕੇ ਉਸ ਟਾਈਪ ਦੇ ਵੇਰੀਏਬਲਜ਼ ਕਰੀਏਟ ਕਰ ਸਕਦੇ ਹਾਂ।

ਉਦਾਹਰਣ :

item *x* ; // memory for *x* is created

C++ ਵਿਚ ਕਲਾਸ ਵੇਰੀਏਬਲਜ਼ ਨੂੰ ਆਬਜੈਕਟ ਕਿਹਾ ਜਾਂਦਾ ਹੈ। ਇਸ ਕਰਕੇ *x* ਨੂੰ ਇਕ ਟਾਈਪ (item) ਦਾ ਆਬਜੈਕਟ ਕਿਹਾ ਜਾਂਦਾ ਹੈ। ਅਸੀਂ ਇਕ ਸਟੇਟਮੈਂਟ ਵਿਚ ਇਕ ਤੋਂ ਜ਼ਿਆਦਾ ਆਬਜੈਕਟ ਡਿਕਲੇਅਰ ਕਰ ਸਕਦੇ ਹਾਂ।

ਉਦਾਹਰਣ : item *x, y, z* ;

ਜਦੋਂ ਕਲਾਸ ਨੂੰ ਪਰਿਭਾਸ਼ਿਤ ਕੀਤਾ ਜਾਂਦਾ ਹੈ ਤਾਂ ਉਸ ਦੀ } ਬਰੈਕਟ ਬੰਦ ਕਰਨ ਤੋਂ ਬਾਅਦ ਵੀ ਆਬਜੈਕਟ ਦਾ ਨਾਮ ਲਿਖ ਕੇ ਵੀ ਆਬਜੈਕਟ ਨੂੰ ਕਰੀਏਟ ਕੀਤਾ ਜਾ ਸਕਦਾ ਹੈ।

```
class item
{
    .....
    .....
    .....
}
x, y, z ;
```

ਕਲਾਸ ਮੈਂਬਰਜ਼ ਨੂੰ ਅਸੈਸ ਕਰਨਾ—— ਮੈਂਬਰਜ਼ ਫੰਕਸ਼ਨ ਨੂੰ ਕਾਲ (Call) ਕਰਨ ਦਾ ਫਾਰਮੈਟ ਹੇਠ ਲਿਖੇ ਅਨੁਸਾਰ ਹੈ——

Object_name : function function-name (actual-argument) ;

3.1.3 ਮੈਂਬਰਜ਼ ਫੰਕਸ਼ਨ ਨੂੰ ਪਰਿਭਾਸ਼ਿਤ ਕਰਨਾ

ਮੈਂਬਰ ਫੰਕਸ਼ਨ ਨੂੰ ਦੋ ਤਰੀਕਿਆਂ ਨਾਲ ਪਰਿਭਾਸ਼ਿਤ ਕੀਤਾ ਜਾਂਦਾ ਹੈ :

1. ਕਲਾਸ ਪਰਿਭਾਸ਼ਾ ਦੇ ਬਾਹਰ
2. ਕਲਾਸ ਪਰਿਭਾਸ਼ਾ ਦੇ ਅੰਦਰ

3.1.3.1 ਕਲਾਸ ਡੈਫੀਨੇਸ਼ਨ ਦੇ ਬਾਹਰ:

ਤਰੀਕਾ——

```
return type class-name :: function-name(argument declaration)
{
    function body
  }
```

ਲੇਬਲ (Lable) classname :: ਇਹ ਕੰਪਾਇਲਰ ਨੂੰ ਦੱਸਦਾ ਹੈ ਕਿ ਦਿੱਤਾ ਗਿਆ ਫੰਕਸ਼ਨ ਕਲਾਸ ਦੇ ਨਾਲ ਸੰਬੰਧਿਤ ਹੈ ਜਿਸ ਦਾ ਅਰਥ ਹੈ

ਕਿ ਇਹ ਫੰਕਸ਼ਨ ਦੀ ਸੰਭਾਵਨਾ (Scope) ਨੂੰ ਪਰਿਭਾਸ਼ਿਤ ਕਰਦਾ ਹੈ। ਇਸ ਚਿੰਨ੍ਹ :: ਨੂੰ (Scope Resolution operator) ਸਕੋਪ ਰੈਸੋਲਿਊਸ਼ਨ ਓਪਰੇਟਰ ਆਖਦੇ ਹਨ।

ਮੰਨ ਲਉ ਅਸੀਂ ਮੈਂਬਰ ਫੰਕਸ਼ਨ getdata( ) ਅਤੇ putdata( ) ਲੈਂਦੇ ਹਾਂ ਤਾਂ ਇਹਨਾਂ ਨੂੰ ਹੇਠ ਲਿਖੇ ਤਰੀਕੇ ਨਾਲ Code ਕੀਤਾ ਜਾਂਦਾ ਹੈ।

```
void item :: getdata (int a, float b)
{
    number = a ;
    cost = b ;
}
    void item :: putdata (void)
{
    cout << ‘‘number :’’
    cout << ‘‘cost :’’ << cost << ‘‘\n’’ ;
```

3.1.3.2 ਕਲਾਸ ਡੈਫੀਨੇਸ਼ਨ ਦੇ ਅੰਦਰ (Inside the class Definition)

ਜਦੋਂ ਫੰਕਸ਼ਨ ਨੂੰ ਕਲਾਸ ਦੇ ਅੰਦਰ ਪਰਿਭਾਸ਼ਿਤ ਕੀਤਾ ਜਾਂਦਾ ਹੈ ਤਾਂ ਇਹ ਇਨ ਲਾਈਨ (Inline) ਫੰਕਸ਼ਨ ਦੀ ਤਰ੍ਹਾਂ ਵਿਹਾਰ ਕਰਦਾ ਹੈ। ਕਲਾਸ ਦੇ ਅੰਦਰ ਪਰਿਭਾਸ਼ਿਤ ਕਰਨ ਦਾ ਤਰੀਕਾ ਹੇਠ ਲਿਖੇ ਅਨੁਸਾਰ ਹੈ——

ਮਿਸਾਲ

```
class item
{
    int number ;
    float cost ;
    public :
void getdata (int a, float b) ; //declaration
// inline function
void putdata (void) ; //definition inside the class
  {
    cout << number << ‘‘/n’’ ;
    cout << cost << ‘‘/n’’ ;
  }
};
```

ਜਦੋਂ ਫੰਕਸ਼ਨ ਨੂੰ ਕਲਾਸ ਦੇ ਅੰਦਰ ਪਰਿਭਾਸ਼ਿਤ ਕੀਤਾ ਜਾਂਦਾ ਹੈ ਤਾਂ ਇਸ ਨੂੰ ਇਨਲਾਇਨ ਫੰਕਸ਼ਨ ਕਿਹਾ ਜਾਂਦਾ ਹੈ।

3.2 ਸਟੈਟਿਕ ਡਾਟਾ ਮੈਂਬਰਜ਼

ਇਕ ਕਲਾਸ ਦੇ ਡਾਟਾ ਮੈਂਬਰਜ਼ ਸਟੈਟਿਕ (static) ਵੀ ਹੋ ਸਕਦੇ ਹਨ। ਇਕ ਸਟੈਟਿਕ ਮੈਂਬਰ ਦੀਆਂ ਕੁਝ ਖ਼ਾਸ ਵਿਸ਼ੇਸ਼ਤਾਵਾਂ ਹੁੰਦੀਆਂ ਹਨ ਜੋ ਕਿ ਹੇਠ ਲਿਖੇ ਅਨੁਸਾਰ ਹਨ :

* ਜਦੋਂ ਕਲਾਸ ਦਾ ਪਹਿਲਾ ਆਬਜੈਕਟ ਬਣਾਇਆ ਜਾਂਦਾ ਹੈ ਤਾਂ ਸਟੈਟਿਕ ਮੈਂਬਰ 0 ਤੋਂ ਇਨੀਸ਼ਲਾਈਜ਼ (initialize) ਹੋ ਜਾਂਦਾ ਹੈ। ਹੋਰ ਕਿਸੇ ਵੀ ਤਰ੍ਹਾਂ ਦੀ ਇਨੀਸ਼ਲਾਈਜ਼ ਦੀ ਆਗਿਆ ਨਹੀਂ ਹੈ।
* ਸਟੈਟਿਕ ਮੈਂਬਰ ਦੀ ਪੂਰੀ ਕਲਾਸ (class) ਦੇ ਲਈ ਇਕ ਹੀ ਕਾਪੀ ਬਣਾਈ ਜਾਂਦੀ ਹੈ ਅਤੇ ਇਹ ਕਲਾਸ ਦੇ ਸਾਰੇ ਆਬਜੈਕਟਸ ਦੁਆਰਾ ਸਾਂਝੀ ਕੀਤੀ ਜਾਂਦੀ ਹੈ।
* ਇਹ ਪੂਰਾ ਸਮਾਂ ਪ੍ਰੋਗਰਾਮ ਦੇ ਵਿਚ ਕੰਮ ਕਰਦਾ ਹੈ ਪਰ ਦਿੱਖਦਾ ਸਿਰਫ ਕਲਾਸ ਵਿਚ ਹੈ।

ਸਟੈਟਿਕ ਵੇਰੀਏਬਲਜ਼ ਉਹਨਾਂ ਕੀਮਤਾਂ ਦੀ ਸਾਂਭ ਸੰਭਾਲ ਲਈ ਵਰਤੀਆਂ ਜਾਂਦੀਆਂ ਹਨ ਜੋ ਕਿ ਸਾਰੀਆਂ ਕਲਾਸਾਂ ਨਾਲ ਸਾਂਝੀਆਂ ਹੁੰਦੀਆਂ ਹਨ।

3.3 ਸਟੈਟਿਕ ਮੈਂਬਰ ਫੰਕਸ਼ਨਜ਼ (Static Member Functions)

ਸਟੈਟਿਕ ਵੇਰੀਏਬਲ ਦੀ ਤਰ੍ਹਾਂ ਹੀ ਸਟੈਟਿਕ ਫੰਕਸ਼ਨ ਵੀ ਹੁੰਦੇ ਹਨ। ਇਕ ਮੈਂਬਰ ਫੰਕਸ਼ਨ ਜਿਸ ਨੂੰ ਅਸੀਂ ਸਟੈਟਿਕ ਬਣਾਉਂਦੇ ਹਾਂ ਦੀਆਂ ਹੇਠ ਲਿਖੀਆਂ ਵਿਸ਼ੇਸ਼ਤਾਵਾਂ ਹੁੰਦੀਆਂ ਹਨ :

* ਸਟੈਟਿਕ ਫੰਕਸ਼ਨ ਸਿਰਫ ਉਹਨਾਂ ਸਟੈਟਿਕ ਮੈਂਬਰਜ਼ ਨੂੰ ਅਸੈਸ ਕਰ ਸਕਦਾ ਹੈ ਜੋ ਇੱਕੋ ਜਿਹੀ ਕਲਾਸ (class) ਵਿਚ ਡਿਕਲੇਅਰ ਕੀਤੇ ਜਾਂਦੇ ਹਨ।
* ਇਕ ਸਟੈਟਿਕ ਮੈਂਬਰ ਫੰਕਸ਼ਨ ਨੂੰ ਇਕ ਕਲਾਸ ਦੀ ਵਰਤੋਂ ਕਰਕੇ ਵੀ ਬੁਲਾਇਆ (call) ਜਾ ਸਕਦਾ ਹੈ। ਜਿਵੇਂ ਹੇਠਾਂ ਦਿੱਤਾ ਗਿਆ ਹੈ :

class_name :: function_name ;

ਹੇਠ ਲਿਖਿਆ ਪ੍ਰੋਗਰਾਮ ਇਹਨਾਂ ਵਿਸ਼ੇਸ਼ਤਾਵਾਂ ਨੂੰ ਲਾਗੂ ਕਰਦਾ ਹੈ :

ਸਟੈਟਿਕ ਮੈਂਬਰ ਫੰਕਸ਼ਨਜ਼ (ਪ੍ਰੋਗਰਾਮ)

```
# include < iostream >
using namespace std ;
class test
{
    int code ;
    static int count ;  //static member variable
    public :
    void setcode (void)
{
    code = ++ count ;
}
void showcode (void)
{
    cout << ‘‘object number :’’ << code << ‘‘/4’’ ;
}
    static void showcount (void) // Static member function
  {
    cout << ‘‘count :’’ << count << ‘‘\n’’ ;
  }
};


int test :: count ;
int main ( )
{
test t1, t2 ;
t1 . setcode ( ) ;
t2 . setcode ( ) ;
test :: showcount ( ) ; // accessing static function
test t3 ;
t3 . setcode ( ) ;
test :: show count ( ) ;
t1 . show code ( ) ;
t2 . show code ( ) ;
t3 . show code ( ) ;
return 0 ;
}
```

3.4 ਫਰੈਂਡਲੀ ਫੰਕਸ਼ਨਜ਼ (friendly Functions)

ਅਸੀਂ ਸ਼ੁਰੂ ਤੋਂ ਹੀ ਇਸ ਅਭਿਆਸ ਵਿਚ ਦੱਸਦੇ ਆ ਰਹੇ ਹਾਂ ਕਿ ਪ੍ਰਾਈਵੇਟ ਮੈਂਬਰ (private Members) ਕਲਾਸ ਦੇ ਬਾਹਰ ਤੋਂ ਅਸੈਸ ਨਹੀਂ ਕੀਤੇ ਜਾ ਸਕਦੇ ਹਨ।

ਇਕ ਨਾਨ ਮੈਂਬਰ ਫੰਕਸ਼ਨ ਕਲਾਸ ਦੇ ਪ੍ਰਾਈਵੇਟ ਡਾਟਾ ਤੱਕ ਪਹੁੰਚ ਨਹੀਂ ਸਕਦਾ। ਉਸ ਹਾਲਤ ਵਿਚ ਅਸੀਂ ਦੋ ਕਲਾਸਾਂ ਨੂੰ ਇਕ ਫੰਕਸ਼ਨ ਸਾਂਝਾ ਕਰਨ ਦੀ ਆਗਿਆ ਦਿੰਦੇ ਹਾਂ।

ਉਸ ਹਾਲਤ ਵਿਚ C++ ਕਾਮਨ (Common) ਫੰਕਸ਼ਨ ਨੂੰ ਦੋਵਾਂ ਕਲਾਸ ਨਾਲ ਮਿੱਤਰਤਾ (friendly) ਵਾਲਾ ਬਣਾਇਆ ਜਾਂਦਾ ਹੈ ਅਤੇ ਫੰਕਸ਼ਨ ਨੂੰ ਕਲਾਸਾਂ ਦੇ ਪ੍ਰਾਈਵੇਟ ਡਾਟਾ ਨੂੰ ਅਸੈਸ ਕਰਨ ਦੀ ਆਗਿਆ ਦਿੱਤੀ ਜਾਂਦੀ ਹੈ। ਉਸ ਫੰਕਸ਼ਨ ਨੂੰ ਕਿਸੇ ਵੀ ਕਲਾਸ ਦਾ ਮੈਂਬਰ ਹੋਣ ਦੀ ਜ਼ਰੂਰਤ ਨਹੀਂ ਹੁੰਦੀ। ਜੇਕਰ ਬਾਹਰੀ ਫੰਕਸ਼ਨ ਨੂੰ ਇਕ ਕਲਾਸ ਨੂੰ friendly ਬਣਾਉਣਾ ਹੈ ਤਾਂ ਸਾਨੂੰ ਸਿੱਧੇ ਤੌਰ ਤੇ ਫੰਕਸ਼ਨ ਨੂੰ ਇਕ friend ਦੀ ਤਰ੍ਹਾਂ ਡਿਕਲੇਅਰ ਕੀਤਾ ਜਾਂਦਾ ਹੈ।

ਉਦਾਹਰਣ :

```
class ABC
{
        .....
        .....
        public
        .....
        .....
        friend void xyz (void) ; //declaration
};
```

ਫੰਕਸ਼ਨ ਡਿਕਲੇਅਰੇਸ਼ਨ ਨੂੰ ਕੀ ਵਰਡ friend ਨਾਲ ਡਿਕਲੇਅਰ ਕੀਤਾ ਜਾਂਦਾ ਹੈ। ਇਕ ਸਾਧਾਰਣ C++ ਫੰਕਸ਼ਨ ਦੀ ਤਰ੍ਹਾਂ ਕਿਤੇ ਵੀ ਡਿਕਲੇਅਰ ਕੀਤਾ ਜਾ ਸਕਦਾ ਹੈ।

ਹੇਠ ਲਿਖਿਆ ਪ੍ਰੋਗਰਾਮ friend Function ਦੀ ਵਰਤੋਂ ਦੱਸਦਾ ਹੈ।

FRIEND FUNCTION

```
# include < iostream >
using namespace std ;
class Sample
{
    int a ;
    int b ;
public :
    void setvalue ( ) {a = 25 ; b = 40 ;}
    friend float mean (sample S) ;
};
    float mean (sample S)
{
    return float (S.a + S.b) / 2.0 ;
}
    int main ( )
  {
    sample x ;    // object X
    X . setvalue ( ) ;
    cout << ‘‘Mean Value = ‘‘ << mean (x) << ‘‘/n’’ ;
    return 0 ;
}
```

3.5 ਕਾਂਸਟੈਂਟ ਮੈਂਬਰ ਫੰਕਸ਼ਨ (const. Member Functions)

ਜੇ ਕੋਈ ਮੈਂਬਰ ਫੰਕਸ਼ਨ ਕਿਸੇ ਵੀ ਕਲਾਸ ਦੇ ਡਾਟਾ ਨੂੰ ਬਦਲਣਾ (alter) ਨਹੀਂ ਚਾਹੁੰਦਾ ਤਾਂ ਇਸ ਨੂੰ const. Member Function ਬਣਾ ਦਿੰਦੇ ਹਨ। ਇਸ ਨੂੰ ਹੇਠ ਲਿਖੇ ਤਰੀਕੇ ਨਾਲ ਡਿਕਲੇਅਰ ਕਰ ਸਕਦੇ ਹਾਂ :

```
void mul (int, int) const ;
double get_balance ( ) const ;
```

3.5.1 ਕਾਂਸਟੈਂਟ ਕਲਾਸ ਆਬਜੈੱਕਟਸ

ਕਿਸੇ ਵੀ ਹੋਰ ਬਿਲਟ-ਇਨ-ਡਾਟਾ ਟਾਈਪ (Built-in-data type) ਦੀ ਤਰ੍ਹਾਂ ਅਸੀਂ ਕਲਾਸ ਦੇ ਆਬਜੈੱਕਟਸ ਨੂੰ ਵੀ ਕਾਂਸਟੈਂਟ ਬਣਾ ਸਕਦੇ ਹਾਂ।

```
const int nvalue = 5 ;
const int nvalue 2(7) ;
```

ਕਲਾਸਾਂ ਦੇ ਕੇਸ ਵਿਚ, ਇੰਸਲਾਈਜੇਸ਼ਨ (initialization) constructors ਦੇ ਜਰੀਏ ਕੀਤੀ ਜਾ ਸਕਦੀ ਹੈ।

```
const Date CDate ;
const Date CDate2 (10, 16, 20, 20) ;
```

ਜਦੋਂ ਕਲਾਸ ਆਬਜੈੱਕਟ ਨੂੰ Construction ਦੇ ਜਰੀਏ ਇਨੀਸ਼ਲਾਈਜ਼ਡ (initialized) ਕਰ ਦਿੱਤਾ ਜਾਂਦਾ ਹੈ ਤਾਂ ਇਸ ਦੇ ਮੈਂਬਰ ਵੇਰੀਏਬਲ ਨੂੰ ਬਦਲਣ (modify) ਦੀ ਕੋਈ ਵੀ ਕੋਸ਼ਿਸ਼ ਕਾਮਯਾਬ ਨਹੀਂ ਹੁੰਦੀ।

ਮਿਸਾਲ :

```
class Something
{
public :
    int m_nValue;
    Something ( ) { m_nValue = 0 ; )

    void Resetvalue ( )  { m_n Value = 0 ; }
    void Setvalue (int Value)  { m_n Value = nValue; }

    int Getvalue ( )  {return m_n Value; }
} ;
int main ( )
{
    const Something cSomething; // calls default constructor

    cSomething.m_nValue = 5 ; //violates const
    cSomething.ResetValue( ) ; //violates const
    cSomething.SetValue (5) ; //vilolates const
    return 0 ;
}
```

3.6 ਪੁਆਇੰਟਰ ਮੈਂਬਰਜ਼ (Pointers to Members)

ਇਕ ਕਲਾਸ ਦੇ ਮੈਂਬਰ ਦਾ ਪਤਾ (address) ਲੈ ਕੇ ਪੁਆਇੰਟਰ ਨੂੰ ਦਿੱਤਾ (assign) ਜਾ ਸਕਦਾ ਹੈ ਅਤੇ ਉਹ address, ਪੂਰੀ ਤਰ੍ਹਾਂ ਕੁਆਲੀਫਾਈਡ (fully qualified) ਓਪਰੇਟਰ, ਤੋਂ ਪ੍ਰਾਪਤ ਕੀਤਾ ਜਾ ਸਕਦਾ ਹੈ।

ਇਕ ਕਲਾਸ ਮੈਂਬਰ ਪੁਆਇੰਟਰ ਓਪਰੇਟਰ :: ਦੀ ਵਰਤੋਂ ਕਰਕੇ ਡਿਕਲੇਅਰ ਕੀਤਾ ਜਾ ਸਕਦਾ ਹੈ।

```
class A
{
    private :
    int on ;
    public
    void show ( ) ;
} ;
```

ਅਸੀਂ ਮੈਂਬਰ *m* ਦੇ ਪੁਆਇੰਟਰ ਨੂੰ ਹੇਠਾਂ ਦਿਖਾਏ ਤਰੀਕੇ ਨਾਲ ਪਰਿਭਾਸ਼ਿਤ ਕਰ ਸਕਦੇ ਹਾਂ :

```
int A ::* iP = & A :: m ;
```

ਉੱਪਰ ਲਿਖੀ ਲਾਈਨ ਵਿਚ (A :: *) ਦਾ ਅਰਥ ਹੈ——

(Pointer-to-member of A class) ਅਤੇ (&A :: m) ਦਾ ਅਰਥ ਹੈ——(address of the m member of A class)

3.7 ਕੰਸਟਰਕਟਰਜ਼ (Constructors)

ਕੰਸਟਰਕਟਰਜ਼ ਇਕ ਖ਼ਾਸ ਮੈਂਬਰ ਫੰਕਸ਼ਨ ਹੈ ਜਿਸ ਦਾ ਕੰਮ ਕਲਾਸ (Class) ਦੇ ਆਬਜੈਕਟ (Object) ਨੂੰ ਇਨੀਸ਼ਲਾਈਜ਼ ਕਰਨਾ ਹੈ। ਇਹ ਪਾਸ ਹੈ ਕਿਉਂਕਿ ਇਸ ਦਾ ਨਾਮ ਕਲਾਸ ਨਾਮ (class name) ਦੀ ਤਰ੍ਹਾਂ ਹੈ। ਕੰਸਟਰਕਟਰਜ਼ ਨੂੰ ਉਦੋਂ ਬੁਲਾਇਆ ਜਾਂਦਾ ਹੈ ਜਦੋਂ ਉਸ ਨਾਲ ਸੰਬੰਧਤ ਆਬਜੈੱਕਟ ਸ (Objects) ਨੂੰ ਕਰੀਏਟ ਕੀਤਾ ਜਾਂਦਾ ਹੈ। ਇਹਨਾਂ ਨੂੰ ਕੰਸਟਰਕਟਰਜ਼ ਕਿਹਾ ਜਾਂਦਾ ਹੈ ਕਿਉਂਕਿ ਇਹ ਕਲਾਸ ਦੇ ਡਾਟਾ ਮੈਂਬਰ ਦੀਆਂ ਕੀਮਤਾਂ ਕੰਸਟਰਕਟ ਕਰਦੀਆਂ ਹਨ।

ਕੰਸਟਰਕਟਰ ਨੂੰ ਹੇਠਾਂ ਦਿਖਾਏ ਤਰੀਕੇ ਨਾਲ ਡਿਕਲੇਅਰ ਅਤੇ ਡਿਫਾਈਨ (Declare & Define) ਕੀਤਾ ਜਾਂਦਾ ਹੈ।

```
// class with constructor
class integer
{
    int m, n ;
    public ;
    integer (void) ;   //constructor declare
    .....
    .....
},
integer :: integer (void)  //constructor defined
  {
    m = 0 ; n = 0 ;
}
```

ਕੰਸਟਰਕਟਰਜ਼ ਫੰਕਸ਼ਨ ਦੀਆਂ ਵਿਸ਼ੇਸ਼ਤਾਵਾਂ ਹੇਠ ਲਿਖੀਆਂ ਹਨ :

* ਉਹ ਹਮੇਸ਼ਾ ਪਬਲਿਕ ਸੈਕਸ਼ਨ ਵਿਚ ਡਿਕਲੇਅਰ ਕੀਤੇ ਜਾਣੇ ਚਾਹੀਦੇ ਹਨ।
* ਉਹ ਆਪਣੇ ਆਪ ਹੀ Call ਹੋ ਜਾਂਦੇ ਹਨ ਜਦੋਂ ਆਬਜੈਕਟ create ਕੀਤੇ ਜਾਂਦੇ ਹਨ।
* ਇਹ ਡਿਰਾਈਵਡ ਕਲਾਸ ਤੋਂ ਗ੍ਰਹਿਣ ਨਹੀਂ ਹੁੰਦੇ ਅਤੇ ਬੇਸ ਕਲਾਸ ਕੰਸਟਰਕਟਰਜ਼ (Base Class Constructor) ਨੂੰ ਕਾਲ (Call) ਕਰ ਸਕਦੇ ਹਨ।
* ਕੰਸਟਰਕਟਰਜ਼ virtual ਨਹੀਂ ਹੋ ਸਕਦੇ।
* ਅਸੀਂ ਇਹਨਾਂ ਦੇ ਐਡਰੈਸ ਨੂੰ ਸੰਬੋਧਨ ਨਹੀਂ ਕਰ ਸਕਦੇ।
* ਇਕ ਆਬਜੈਕਟ ਜੋ ਕਿ ਕੰਸਟਰਕਟਰ ਨਾਲ ਹੈ, ਉਸ ਨੂੰ ਯੂਨੀਅਨ (Union) ਦੇ ਮੈਂਬਰ ਦੀ ਤਰ੍ਹਾਂ ਨਹੀਂ ਵਰਤੇ ਜਾ ਸਕਦੇ।

ਮਿਸਾਲ—

class with constructor

```
# include < iostream >
using namespace std ;
class integer
  {
    int m, n ;
   public :
    integer (int, int) ;                    //constructor declared
    void display (void)
    {
    cout << ‘‘m = ’’ << m << ‘‘\n’’ ;
    cout << ‘‘n = ’’ << n << ‘‘\n’’ ;
  }
}
    integer :: integer (int x, int y)
    {
    m = x ; n = y ;                         //constructor defined
    }
    int main( )
    {
    integer int1 (0, 100) ;                 //constructo called implicit g.
    integer int2 = integer (25, 75) ;   //constructor called explicity
    cout << ‘‘\n OBJECT 1’’ << ‘‘\n’’ ;
    int 1. display ( ) ;
    cout << ‘‘\’’ OBJECT 2’’ << ‘‘\n’’ ;
    int2 display ( ) ;
    return 0 ;
}
```

3.8 ਓਵਰਲੋਡਿਡ ਕੰਸਟਰਕਟਰ :

ਜਦੋਂ ਇਕ ਤੋਂ ਜ਼ਿਆਦਾ ਕੰਸਟਰਕਟਰ ਫੰਕਸ਼ਨ, ਇਕ ਕਲਾਸ ਵਿਚ ਪਰਿਭਾਸ਼ਿਤ ਕੀਤੇ ਜਾਂਦੇ ਹਨ ਤਾਂ ਉਸ ਨੂੰ ਓਵਰਲੋਡਿਡ ਕੰਸਟਰਕਟਰ ਕਹਿੰਦੇ ਹਨ।

```
# include < iostream >
using namespace std ;


class complex
{
    float x, y ;
public :
    complex( ) & }
    complex (float a) {x = y = a; }
    complex (float real, float imag)
    {x = real ; y = imag ; }
friend complex sum (complex, complex) ;
    friend void show (complex) ;
};
complex sum (complex C1, complex C2)
{
    complex C3 ;
    C3.x = C1.x + C2.x ;
    C3.y = C1.y + C2.y ;
    return (C3) ;
}
void show (complex C)
{
   cout << C.x << ''+ j'' << C.y << ''\n'' ;
}
int main ( )
{
    complex A (2.7, 3.5) ;
    complex B (1, 6) ;
    complex C ;
    C = Sum (A, B) ;
    cout << ''A = ''; Show (A) ;
    cout << ''B = '' ; Show (B) ;
    cout << ''C = '' ; Show (C) ;
    return 0;
}
```

3.9 ਕਾਪੀ ਕੰਸਟਰਕਟਰ (Copy Constructor)

ਇਕ ਕਾਪੀ ਕੰਸਟਰਕਟਰ ਇਕ ਆਬਜੈਕਟ ਨੂੰ ਦੂਸਰੇ ਆਬਜੈਕਟ ਤੋਂ ਇਨੀਸ਼ਿਲਾਈਜ਼ (initialize) ਅਤੇ ਡਿਕਲੇਅਰ ਕਰਨ ਲਈ ਵਰਤਿਆ ਜਾਂਦਾ ਹੈ।

ਮਿਸਾਲ : integer I2 (I1) ;

ਇਹ I2 ਆਬਜੈਕਟ (Object) ਨੂੰ ਪਰਿਭਾਸ਼ਿਤ ਕਰਦਾ ਹੈ ਅਤੇ ਇਕੋ ਹੀ ਸਮੇਂ ਵਿਚ I2 ਦੀਆਂ ਕੀਮਤਾਂ I1 ਦੀਆਂ ਬਣ ਜਾਂਦੀਆਂ ਹਨ।

ਕਾਪੀ ਕੰਸਟਰਕਟਰ ਦੇ ਦੁਆਰਾ Initialize ਕਰਨ ਦੀ ਪ੍ਰਕਿਰਿਆ ਨੂੰ COPY ਇਨਿਸ਼ਲਾਈਜੇਸ਼ਨ ਕਹਿੰਦੇ ਹਨ।

3.10 Desctructor

ਜਿਵੇਂ ਕਿ ਨਾਮ ਤੋਂ ਹੀ ਪਤਾ ਚੱਲਦਾ ਹੈ ਕਿ ਜੋ ਆਬਜੈਕਟ constructor ਦੇ ਦੁਆਰਾ ਬਣਾਏ ਜਾਂਦੇ ਹਨ, ਇਹ ਉਹਨਾਂ ਨੂੰ ਨਸ਼ਟ ਕਰਨ ਦੇ ਕੰਮ ਆਉਂਦਾ ਹੈ।

ਕੰਸਟਰਕਟਰ ਦੀ ਤਰ੍ਹਾਂ ਡਿਸਟਰਕਟਰ ਵੀ ਇਕ ਮੈਂਬਰ ਫੰਕਸ਼ਨ ਹੁੰਦਾ ਹੈ ਜਿਸ ਦਾ ਨਾਮ ਉਹੀ ਹੈ ਜੋ ਕਲਾਸ ਦਾ ਹੈ, ਪਰ ਇਸ ਦੇ ਨਾ ਅੱਗੇ (~) Tilde ਦਾ ਚਿੰਨ੍ਹ ਲੱਗਿਆ ਹੁੰਦਾ ਹੈ।

ਇਕ Destructor ਨਾ ਹੀ ਕੋਈ ਆਰਗੂ ਲੈਂਦਾ ਹੈ ਨਾ ਹੀ ਕੋਈ ਕੀਮਤ ਰਿਟਰਨ (Return) ਕਰਦਾ ਹੈ।

ਜਦੋਂ ਅਸੀਂ ਪ੍ਰੋਗਰਾਮ ਤੋਂ ਬਾਹਰ ਆਉਂਦੇ ਹਨ ਤਾਂ ਕੰਪਾਇਲਰ ਇਸ ਨੂੰ ਆਪਣੇ ਆਪ ਹੀ ਬੁਲਾ ਲੈਂਦਾ ਹੈ ਤਾਂਕਿ ਇਹ ਫੰਕਸ਼ਨ ਆ ਕੇ ਉਸ ਮੈਮਰੀ ਸਪੇਸ ਨੂੰ ਸਾਫ ਕਰੇ ਜੋ ਵਰਤੋਂ ਵਿਚ ਨਹੀਂ ਆ ਰਿਹਾ।

3.11 C++ this Pointer

C++ ਵਿਚ ਹਰ ਇਕ ਆਬਜੈਕਟ ਆਪਣੇ ਪਤੇ (address) ਨੂੰ ਅਸੈਸ ਕਰਨ ਲਈ ਇਕ ਖ਼ਾਸ ਪੁਆਇੰਟਰ (this pointer) ਦੀ ਵਰਤੋਂ ਕਰਦੇ ਹਨ। this pointer ਸਾਰੇ member functions ਲਈ ਇਕ Implicit Parameter ਹੈ।
friend functions ਦਾ this pointer ਨਹੀਂ ਹੁੰਦਾ ਕਿਉਂਕਿ friend class ਦੇ ਮੈਂਬਰ ਨਹੀਂ ਹੁੰਦੇ।

```
# include <iostream.h>
using namespace std;
class Box
{
     public :                    // Constructor definition

          Box (double 1 = 2.0, double b = 2.0, double h = 2.0)
          {
               cout <<‘‘Constructor called.‘‘ << end1;
               length = l ;
               breadth = b ;
               height = h ;
          }
          double Volume ( )
          {
               return length * breadth * height ;
          }
          int compare (Box box)
          {
               return this->Volume ( ) > box.Volume ( ) ;
          }
     private :
          double length;              // Length of a box
          double breadth;             // Breadth of a box
          double height;              // Height of a box
 };
int main (void)
{
     Box Box1 (3.3, 1.2, 1.5) ;      // Declare box1
     Box Box2 (8.5, 6.0, 2.0) ;      // Declare box2
     if (Box1.compare (Box2))
     {
          cout << ‘‘Box2 is smaller than Box1’’ <<end1;
     }
     else
     {
          cout << ‘‘Box2 is equal to or larger than Box1’’ <<end1;
     }
     return 0;
  }
```

3.12 ਐਂਪਟੀ ਕਲਾਸਾਂ (Empty classes)

ਅਸੀਂ ਐਂਪਟੀ ਕਲਾਸਾਂ ਘੋਸ਼ਿਤ ਕਰ ਸਕਦੇ ਹਾਂ ਪਰ ਇਨ੍ਹਾਂ ਕਲਾਸਾਂ ਦੇ ਆਬਜੈੱਕਟ ਦਾ ਸਾਈਜ਼ ਨਾਨ-ਜ਼ੀਰੋ (Non-zero) ਹੁੰਦਾ ਹੈ।
ਮਿਸਾਲ :

Example :

```
// empty_classes.cpp
// compile with : /EHsC
# include <iostream.h>
class NoMembers
{
};
using namespace std ;
int main ( )
{
```

```
    NoMembers n;  // Object of type Nomembers.
    cout << ‘‘The size of an object of empty class is :’’
         << size of n << endl;
}
```

Output

The size of an object of empty class is : 1.

ਇਨ੍ਹਾਂ ਆਬਜੈੱਕਟ ਲਈ ਨਾਨ-ਜੀਰੋ ਸਾਈਜ਼ ਦੀ ਮੈਮਰੀ ਵੰਡੀ ਜਾਂਦੀ ਹੈ। ਇਸ ਕਰਕੇ ਇਨ੍ਹਾਂ ਆਬਜੈੱਕਟ ਦਾ ਮੈਮਰੀ ਵਿਚ ਅਲਗ ਪਤਾ ਹੁੰਦਾ ਹੈ।

3.13 Assignment vs Initialization

ਅਸਾਈਨਮੈਂਟ ਅਤੇ ਇਨੀਸ਼ਿਲਾਈਜੇਸ਼ਨ ਦੋਨੋਂ ਹੀ ਅਲਗ-ਅਲਗ ਓਪਰੇਸ਼ਨ ਹਨ ਅਤੇ ਇਨ੍ਹਾਂ ਦੀ ਵੱਖਰੀ ਯੂਜੇਸ (uses) ਹਨ। ਅਸਾਈਨਮੈਂਟ ਤੇ ਇਨੀਸ਼ਿਲਾਈਜੇਸ਼ਨ ਇਸ ਕਰਕੇ ਵੀ ਵੱਖਰੇ ਹਨ ਕਿਉਂਕਿ ਇਹ ਵੱਖਰੇ ਕੰਨਟੈਕਸਟ (context) ਵਿਚ ਵਰਤੇ ਜਾਂਦੇ ਹਨ ਤੇ ਵੱਖਰੇ-ਵੱਖਰੇ ਕੰਮ ਕਰਦੇ ਹਨ। ਪਰ ਇਨ੍ਹਾਂ ਦੇ ਵਿਚ ਇਹ ਅੰਤਰ ਬਿਲਟ-ਇਨ (Built-in) ਡਾਟਾ ਟਾਈਪ ਜਿਵੇਂ ਕਿ int on double ਵਿਚ ਲਾਗੂ ਨਹੀਂ ਹੁੰਦਾ। ਕਿਉਂਕਿ ਇਹ ਦੋਨੋਂ ਉਪਰੇਸ਼ਨ ਇਸ ਕੇਸ ਵਿਚ ਸਿਰਫ਼ ਕੁਝ ਬਿਟਸ (bits) ਨੂੰ ਹੀ ਕਾਪੀ ਕਰਦੇ ਹਨ।

```
int a = 12;          //initialization, copy 0 × 000c to a
a = 12 ;             //assignment, copy 0 × 000c to a
```

ਯੂਜਰ ਡਿਫਾਈਨਡ (user-defined) ਡਾਟਾ ਟਾਈਪ ਦੇ ਕੇਸ ਵਿਚ ਇਹ ਅਲੱਗ ਤਰੀਕੇ ਨਾਲ ਹੁੰਦਾ ਹੈ ਜੋ ਹੇਠਾਂ ਦਿੱਤੀ ਮਿਸਾਲ ਨਾਲ ਸਮਝਾਇਆ ਗਿਆ ਹੈ—

```
class string {
    public :
    string (const char * init) ;  //intentionally not explicit !
    ~ string ( ) ;
    string (const string & that),
    string & operator = (const. string & that) ;
    string & operator = (const. char * Str) ;
    void Swap (string & that) ;
    friend const string // concate ~ at r
    operator + (const string & const string &) ;
    friend bool operator < (const string &, const string &) ;
    // .....
    private
    string (const char *, const char *) ;  // Computational
    char * & ;
 };
```

string ਆਬਜੈੱਕਟ ਨੂੰ character string ਦੇ ਨਾਲ initialise ਅੱਗੇ ਦਿੱਤੇ ਤਰੀਕੇ ਨਾਲ ਕੀਤਾ ਜਾਂਦਾ ਹੈ।

```
string :: string (const char * int) &
if (!int) init = ‘‘  ’’;
S_ = new char [strlen (init) + 1];
strcpy (S_1 init) ;
}
```

Destructor ਤਾਂ ਉਹੀ ਕੰਮ ਕਰੇਗਾ ਜੋ ਉਸ ਨੂੰ ਆਉਂਦਾ ਹੈ

```
string :: ~ string ( ) & delete [ ]  S_, }
```

ਕਨਸਟਰਕਸ਼ਨ ਦੇ ਮੁਕਾਬਲੇ ਅਸਾਈਨਮੈਂਟ ਜ਼ਿਆਦਾ ਮੁਸ਼ਕਿਲ ਕੰਮ ਹੈ।

```
string & string :: operator = (const char * str)}
if (! str) str = ‘‘ ’’ ;
char * imp = strcpy [new char [strlen (str) + 1], str) ;
delete[ ] S_,
S_ = temp;
return * this ;
}
```

3.14 class vs Object

ਆਬਜੈਕਟ (Object) : ਅਸਲੀ ਸੰਸਾਰ ਦੇ ਆਬਜੈਕਟ ਦੀ ਦੋ ਮੇਨ (main) ਵਿਸ਼ੇਸ਼ਤਾਵਾਂ ਹੁੰਦੀਆਂ ਹਨ—— ਸਟੇਟ ਅਤੇ ਬਿਹੇਵੀਅਰ (State & Behaviour)। ਜਿਵੇਂ ਕਿ ਮਨੁੱਖ ਦੀ ਸਟੇਟ (ਨਾਮ, ਉਮਰ) ਤੇ ਬਿਹੇਵੀਅਰ (ਦੌੜਣਾ, ਸੌਣਾ)। ਸਾਫਟਵੇਅਰ ਆਬਜੈਕਟ ਵੀ ਅਸਲੀ ਸੰਸਾਰ ਦੇ ਆਬਜੈਕਟ ਦੀ ਤਰ੍ਹਾਂ ਹੀ ਹੁੰਦੇ ਹਨ ਜਿਸ ਦਾ ਆਪਣਾ ਫੀਲਡ ਤੇ ਆਪਣੇ ਫੰਕਸ਼ਨ ਹੁੰਦੇ ਹਨ।

ਕਲਾਸ (class) : ਕਲਾਸ ਇਕ ਤਰ੍ਹਾਂ ਦਾ ਟੈਂਪਲੇਟ ਜਾਂ Blue Print ਹੈ ਜੋ ਆਬਜੈਕਟ ਬਣਾਉਣ ਦੇ ਕੰਮ ਆਉਂਦਾ ਹੈ। ਇਕ ਕਲਾਸ ਵਿਚ ਫੀਲਡ, ਸਟੈਟਿਕ ਫੀਲਡ, ਮੈਥਡ, ਸਟੈਟਿਕ ਮੈਥਡ ਤੇ ਕੰਸਟਰਕਟਰ ਹੁੰਦੇ ਹਨ। ਫੀਲਡ ਕਲਾਸ ਦੀ ਸਟੇਟ ਨੂੰ ਰੱਖਦੀ (hold) ਹੈ ਤੇ ਮੈਤਡ class ਦੇ ਬਿਹੇਵੀਅਰ (Behaviour) ਨੂੰ।

ਸੰਖੇਪ ਵਿਚ ਕਿਹਾ ਜਾ ਸਕਦਾ ਹੈ ਕਿ ਇਕ ਆਬਜੈਕਟ ਆਪਸ ਵਿਚ ਸੰਬੰਧਿਤ ਸਟੇਟ ਅਤੇ ਬਿਹੇਵੀਅਰ ਦਾ ਇਕ ਸਾਫਟਵੇਅਰ ਬੰਡਲ ਹੈ ਤੇ ਕਲਾਸ ਉਹ Blueprint ਹੈ ਜੋ ਆਬਜੈਕਟ ਬਣਾਉਣ ਲਈ ਵਰਤਿਆ ਜਾਂਦਾ ਹੈ।

3.15 ਐਰੇ ਆਫ਼ ਆਬਜੈਕਟਸ (Array of Objects)

ਵੈਰੀਏਬਲਸ ਦਾ ਉਹ ਐਰੇ ਜਿਸ ਦੀ ਟਾਈਪ "ਕਲਾਸ" ਹੋਵੇ ਨੂੰ ਐਰੇ ਆਫ ਆਬਜੈੱਕਟ ਸ ਕਿਹਾ ਜਾਂਦਾ ਹੈ। ਐਰੇ ਆਫ਼ ਆਬਜੈੱਕਟ ਸ ਨੂੰ ਰੈਫਰ ਕਰਨ ਲਈ ਜਿਸ 'Identifier' ਦੀ ਵਰਤੋਂ ਕੀਤੀ ਜਾਂਦੀ ਹੈ, ਉਹ ਇਕ ਯੂਜਰ ਦੁਆਰਾ ਪਰਿਭਾਸ਼ਿਤ ਡਾਟਾ ਟਾਈਪ ਹੁੰਦੀ ਹੈ।

Example :

```
# include <iostream.h>
const int MAX = 100 ;
class Details
 {
   private :
        int salary ;
        float roll;
   public :
        void getname ( )
            {
                cout << ''\n Enter the Salary :'' ;
                cin >> salary ;
                cout << ''\n Enter the roll :'' ;
                cin >> roll ;
            }
        void putname ( )
            {
                cout << ''Employees'' << salary <<
                ''and roll is'' << roll << '\n' ;
            }
        };

   void main ( )
        {
            Details det [MAX];
            int n = 0;
            char ans ;
            do {
                cout << ''Enter the Employee Number ::'' << n + 1;
                det [n++].getname;
                cout << ''Enter another (y/n) ? :'' ;
                cin >> ans ;
            } while (ans ! = 'n') ;
        for (int j = 0 ; j < n ; j++)
            {
                cout << ''\nEmployee Number is ::'' << j + 1;
                det [j].putname ( ) ;
            }
        }
```

3.16 ਓਵਰਲੋਡਿੰਗ ਆਫ ਨਿਉ ਐਂਡ ਡਿਲੀਟ ਉਪਰੇਟਰ (Overloading of New and Delete operator)

ਨਿਊ ਅਤੇ ਡਿਲੀਟ (new and delete) ਓਪਰੇਟਰ ਨੂੰ ਅਸੀਂ ਕਿਸੇ ਵੀ ਹੋਰ C++ ਓਪਰੇਟਰ ਦੀ ਤਰ੍ਹਾਂ ਓਵਰਲੋਡ ਕਰ ਸਕਦੇ ਹਾਂ। ਪਰ ਇਸ ਕੰਮ ਦੌਰਾਨ ਸਾਨੂੰ ਪੈਰਾਮੀਟਰ ਲੈਣ (accepting of parameter), ਕੀਮਤ ਰਿਟਰਨ ਕਰਨ (value to return) ਅਤੇ ਥਾਂ ਘੋਸ਼ਿਤ ਕਰਨ (place to declare) ਸਬੰਧੀ ਸਾਵਧਾਨੀ ਵਰਤਣੀ ਪੈਂਦੀ ਹੈ।

ਅਸੀਂ ਜਾਣਦੇ ਹਾਂ ਕਿ C++ ਵਿਚ new operator ਮੈਮਰੀ ਵੰਡਣ (Memory allocation) ਲਈ ਵਰਤਿਆ ਜਾਂਦਾ ਹੈ। C ਵਿਚ ਇਸ ਦੀ ਥਾਂ ਤੇ malloc ( ) ਫੰਕਸ਼ਨ ਦੀ ਵਰਤੋਂ ਕੀਤੀ ਜਾਂਦੀ ਹੈ।

ਨਿਊ ਅਤੇ ਡਿਲੀਟ ਉਪਰੇਟਰ ਨੂੰ ਓਵਰਲੋਡ ਕਰਨ ਦੀ ਲੋੜ ਕਿਉਂ ਪੈਂਦੀ ਹੈ ?

1. ਮੈਮਰੀ ਦੀ ਵੰਡ ਦੇ ਉੱਤੇ ਕੰਟਰੋਲ ਕਰਨ ਲਈ।
2. ਮੈਮਰੀ ਦੀ ਵੰਡ ਅਤੇ ਖਾਲੀ ਕਰਨ (Allocation and deallocation) ਦਾ ਪਤਾ (track) ਰੱਖਣ ਲਈ।
3. ਮੈਮਰੀ ਦੀ ਵੰਡ ਦੇ ਸਮੇਂ ਕੋਈ ਹੋਰ ਉਪਰੇਸ਼ਨ ਕਰਨ ਲਈ।

ਹੇਠਾਂ ਇਕ ਮਿਸਾਲ ਦਿੱਤੀ ਗਈ ਹੈ ਜੋ New ਅਤੇ Delete ਉਪਰੇਟਰ ਦੀ ਓਵਰਲੋਡਿੰਗ ਨੂੰ ਬਹੁਤ ਹੀ ਸੌਖੇ ਤਰੀਕੇ ਨਾਲ ਦੱਸਦੀ ਹੈ।

```
void * operator new (size_t new)
{
    return malloc (nuon);
}
void operator delete (void * ptr)
{
    free (ptr) ;
}
```

Example

The following example overloads the + operator to add two complex numbers and returns the result.

```
// operator_overloading.cpp
// compile with : /EHsc
# include <iostream.h>
using namespace std;
class complex
{
public :
    complex (double r, double i) : re(r), im(i) { }
    complex operator + (complex & other) ;
    void Display { } {cout << re << ‘‘,’’ << im << endl;}
private :
    double re, im;
};
// operator overloaded using a member function
complex complex :: operator + (complex & other)
{
return complex (re + other.re, im + other.im);
}
int main ( )
{
    complex a = complex (1.2, 3.4) ;
    complex b = complex (5.6, 7.8) ;
    complex c = complex (0.0, 0.0) ;
    c = a + b ;
    c. Display ( );
```

A simple Message Header

```
// sample of operator Overloading
# include <string>
class PlMessageHeader
{
    std :: string m_ThreadSender;
    std :: string m_ThreadReceiver;
    //return true if the messages are equal, false otherwise
    inline bool operator == (const PlMessageHeader & b) const
    {
        return ((b.m_Threadsender == m_ThreadSender) &&
```

```
            (b.m_ThreadReceiver == m_ThreadReceiver));
    }
    //return true if the message is for name
    inline bool is For (const std :: string & name) const
    {
        return (m_ThreadReceiver == name) ;
    }
    //return true if the message is for name
    inline bool is For (const char * name) const
    {
        return (m_ThreadReceiver == name); //since name type is std :: string, it becomes unsafe if name ==
        NULL
    }
};
```

3.17 ਟਾਈਪ ਕਨਵਰਜਨ (Type Conversion)

ਵਿਗਿਆਨਿਕ ਗਣਨਾਵਾਂ ਵਿਚ ਇਕੋ ਪ੍ਰਕਾਰ ਦੇ ਡਾਟੇ ਨੂੰ ਅਲਗ-ਅਲਗ ਰੂਪ ਵਿਚ ਦਿਖਾਉਣਾ ਇਕ ਆਮ ਗੱਲ ਹੈ। ਇਸ ਵਿਚ ਡਾਟੇ ਦੀ ਇਕ ਕਿਸਮ ਤੋਂ ਦੂਜੀ ਕਿਸਮ ਵਿਚ ਤਬਦੀਲੀ ਸ਼ਾਮਿਲ ਹੁੰਦੀ ਹੈ। ਜਿਵੇਂ ਕਿ ਰੇਡਿਅਨ (radian) ਤੋਂ ਡਿਗਰੀ (Degree) ਵਿਚ ਤਬਦੀਲੀ, ਪੋਲਰ (Polar) ਤੋਂ ਰੈਕਟਐਂਗੁਲਰ (Rectangular) ਵਿਚ ਤਬਦੀਲੀ ਆਦਿ। C++ ਵਿਚ ਤਬਦੀਲੀ ਦੀ ਇਹ ਪ੍ਰਕਿਰਿਆ ਅਸਾਈਨਮੈਂਟ (assignment) ਉਪਰੇਟਰ ਨੂੰ ਓਵਰਲੋਡ ਕਰ ਕੇ ਕੀਤੀ ਜਾ ਸਕਦੀ ਹੈ। ਜਿਵੇਂ ਕਿ

var1 = var2

ਇੱਥੇ var1 ਅਤੇ var2 ਦੋਨੋਂ ਹੀ ਇੰਟੀਜਰ (integer) ਟਾਈਪ ਦੇ ਹਨ। ਇਸੇ ਤਰ੍ਹਾਂ ਇਕੋ ਕਲਾਸ ਵਿਚ ਬਣਾਏ ਗਏ ਉਪਰੇਟਰ ਦੀ ਕੀਮਤ (value) ਵੀ ਇਕ ਦੂਜੇ ਨੂੰ ਅਸਾਈਨ (assign) ਕੀਤੀ ਜਾ ਸਕਦੀ ਹੈ। ਜਿਵੇਂ ਕਿ C3 = C1 + C2 // C1, C2 ਅਤੇ C3 ਇਕ ਕਲਾਸ ਦੇ ਆਬਜੈਕਟ ਹਨ।

ਇਕੋ ਪ੍ਰਕਾਰ ਦੇ ਡਾਟਾ ਆਈਟਮਜ਼ ਦੀ ਤਬਦੀਲੀ ਕੰਪਾਇਲਰ ਦੁਆਰਾ ਕੀਤੀ ਜਾਂਦੀ ਹੈ ਤੇ ਯੂਜਰ ਵੱਲੋਂ ਕੋਈ ਯਤਨ ਨਹੀਂ ਕੀਤਾ ਜਾਂਦਾ। ਜੇਕਰ ਡਾਟਾ ਆਈਟਮਸ ਅਲਗ-ਅਲਗ ਹਨ ਤਾਂ ਡਾਟੇ ਦੀ ਤਬਦੀਲੀ ਲਈ ਫੰਕਸ਼ਨ ਯੂਜਰ ਨੂੰ ਬਣਾਉਣਾ ਪੈਂਦਾ ਹੈ।

3.17.1 ਬੇਸਿਕ ਡਾਟਾ ਟਾਈਪ ਵਿਚ ਕਨਵਰਸ਼ਨ (Conversion between Basic Data Types)

ਮਿਸਾਲ : Weight = Age

ਇਥੇ weight ਦੀ ਡਾਟਾ ਟਾਈਪ ਫਲੋਟ (float) ਹੈ ਤੇ age ਦੀ ਡਾਟਾ ਟਾਈਪ ਇੰਟੀਜਰ (integer) ਹੈ।

ਇਥੇ ਕੰਪਾਇਲਰ ਇਕ ਸਪੈਸ਼ਲ ਫੰਕਸ਼ਨ ਨੂੰ ਕਾਲ ਕਰਦਾ ਹੈ ਜੋ age ਦੀ ਕੀਮਤ ਨੂੰ float ਵਿਚ ਬਦਲ ਦਿੰਦੀ ਹੈ।

ਕੰਪਾਇਲਰ ਕੋਲ ਇਹੋ ਜਿਹੇ ਬਹੁਤ ਸਾਰੇ ਬਿਲਟ-ਇਨ-ਫੰਕਸ਼ਨ (Built-in-Function) ਹੈ ਜੋ ਬੇਸਿਕ ਡਾਟਾ ਟਾਈਪਸ ਜਿਵੇਂ ਕਿ char ਤੋਂ int, float ਤੋਂ double ਆਦਿ ਵਿਚ ਬਦਲ ਦਿੰਦੇ ਹਨ।

ਕੰਪਾਇਲਰ ਦੀ ਉਹ ਵਿਸ਼ੇਸ਼ਤਾ ਜਿਸ ਵਿਚ ਡਾਟਾ ਦੀ ਤਬਦੀਲੀ ਯੂਜਰ ਦੀ ਮਦਦ ਤੋਂ ਬਿਨਾਂ ਕੀਤੀ ਜਾਂਦੀ ਹੈ, ਨੂੰ ਇੰਪਲੀਸਿਟ ਟਾਈਪ ਕਨਵਰਸ਼ਨ (Implicit Type Conversion) ਕਿਹਾ ਜਾਂਦਾ ਹੈ।

ਕੰਪਾਇਲਰ ਨੂੰ ਬਾਹਰ ਤੋਂ ਟਾਈਪ ਕਾਸਟ ਓਪਰੇਟਰ (Type Cast operator) ਦੀ ਵਰਤੋਂ ਕਰਕੇ, ਟਾਈਪ ਕਨਵਰਸ਼ਨ ਕਰਨ ਲਈ ਹਦਾਇਤਾਂ ਦਿੱਤੀਆਂ ਜਾਂਦੀਆਂ ਹਨ। ਜਿਵੇਂ ਕਿ int ਤੋਂ float ਵਿਚ ਬਦਲਣ ਲਈ ਹਦਾਇਤ ਹੈ—

weight = (float) age;

ਇਸੇ ਸ਼ਬਦ float ਜੋ ਕਿ ਬਰੈਕਟਾਂ ਵਿਚ ਬੰਦ ਹੈ ਨੂੰ ਟਾਈਪ ਕਾਸਟ ਉਪਰੇਟਰ ਕਿਹਾ ਜਾਂਦਾ ਹੈ।

float ਤੋਂ int ਦੀ ਐਕਸਪਲੀਸਿਟ ਕਨਵਰਸ਼ਨ (Explicit Conversion) ਉਹ ਹੀ ਫੰਕਸ਼ਨ ਅਪਣਾਉਣਗੇ ਜੋ ਕਿ Implicit Conversion ਇੰਪਲੀਸਿਟ ਕਨਵਰਸ਼ਨ ਦੁਆਰਾ ਅਪਣਾਏ ਜਾਂਦੇ ਹਨ।

3.17.2 ਆਬਬਜੈਕਟ ਅਤੇ ਬੇਸਿਕ ਡਾਟਾ ਟਾਈਪਸ ਦੇ ਵਿਚ ਕਨਵਰਸ਼ਨ (Conversion between Object and Basic Data Types)

ਕੰਪਾਇਲਰ ਸਿਰਫ ਉਹਨਾਂ ਬਿਲਟ-ਇਨ-ਡਾਟਾ ਟਾਈਪਸ (Built-in-Data Types) ਦੀ ਕਨਵਰਸ਼ਨ ਕਰ ਸਕਦਾ ਹੈ ਜੋ ਪ੍ਰੋਗਰਾਮਿੰਗ ਭਾਸ਼ਾ ਦੁਆਰਾ ਸਪੋਰਟ (Support) ਕੀਤੀ ਜਾਂਦੀ ਹੈ। ਯੂਜਰ ਪਰਿਭਾਸ਼ਿਤ ਡਾਟਾ ਟਾਈਪਸ ਦੀ ਪਰੀਮੀਟਿਵ (Primitive) ਡਾਟਾ ਟਾਈਪ ਵਿਚ ਤਬਦੀਲੀ ਲਈ ਅਸੀਂ ਕੰਪਾਇਲਰ ਦੇ ਭਰੋਸੇ ਨਹੀਂ ਰਹਿ ਸਕਦੇ ਕਿਉਂਕਿ ਕੰਪਾਇਲਰ ਨੂੰ ਯੂਜਰ ਪਰਿਭਾਸ਼ਿਤ ਡਾਟਾ ਟਾਈਪਸ ਦਾ ਲੋਜਿਕਲ ਅਰਥ ਨਹੀਂ ਪਤਾ ਹੁੰਦਾ। ਇਸ ਕਰਕੇ ਇਕ ਅਰਥਪੂਰਣ ਤਬਦੀਲੀ ਕਰਨ ਲਈ ਯੂਜਰ ਨੂੰ ਲੋੜ ਅਨੁਸਾਰ ਕਨਵਰਸ਼ਨ ਫੰਕਸ਼ਨ ਬਣਾਉਣੇ ਪੈਂਦੇ ਹਨ ।ਯੂਜਰ ਪਰਿਭਾਸ਼ਿਤ ਡਾਟਾ ਟਾਈਪਸ ਤੇ ਬੇਸਿਕ ਡਾਟਾ ਟਾਈਪਸ ਵਿਚ ਤਬਦੀਲੀ ਦੀ ਪ੍ਰਕਿਰਿਆ ਨੂੰ ਪ੍ਰੋਗਰਾਮ meter.cpp ਵਿਚ ਵਿਸਥਾਰ ਨਾਲ ਸਮਝਾਇਆ ਗਿਆ ਹੈ।

ਕਿਵੇਂ ਅਤੇ ਕਿੱਥੇ ਇਹ ਕਨਵਰਸ਼ਨ ਫੰਕਸ਼ਨ ਮੌਜੂਦ ਹੁੰਦੇ ਹਨ ?(Where and How the Conversion function should exist ?)

ਬੇਸਿਕ ਟਾਈਪ ਤੋਂ ਯੂਜਰ ਪਰਿਭਾਸ਼ਿਤ ਟਾਈਪ ਵਿਚ ਤਬਦੀਲੀ ਲਈ, ਕਨਵਰਸ਼ਨ ਫੰਕਸ਼ਨ, ਕਨਸਟਰਕਟਰ (Constructor) ਦੇ ਰੂਪ ਵਿਚ, ਯੂਜਰ ਪਰਿਭਾਸ਼ਿਤ ਕਲਾਸ ਦੇ ਆਬਬਜੈਕਟ ਦੇ ਵਿਚ ਪਰਿਭਾਸ਼ਿਤ ਹੋਣਾ ਚਾਹੀਦਾ ਹੈ।

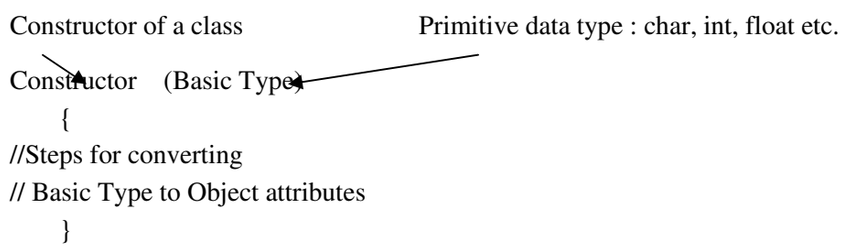


Fig. Conversion function : basic to user-defined

ਇਸੇ ਤਰ੍ਹਾਂ ਯੂਜਰ ਪਰਿਭਾਸ਼ਿਤ ਟਾਈਪ ਤੋਂ ਬੇਸਿਕ ਡਾਟਾ ਟਾਈਪ ਵਿਚ ਤਬਦੀਲੀ ਲਈ, ਕਨਵਰਸ਼ਨ ਫੰਕਸ਼ਨ, ਓਪਰੇਟਰ ਫੰਕਸ਼ਨ ਦੇ ਰੂਪ ਵਿਚ, ਯੂਜਰ ਪਰਿਭਾਸ਼ਿਤ ਕਲਾਸ ਦ ਆਬਬਜੈਕਟ ਦੇ ਵਿਚ ਪਰਿਭਾਸ਼ਿਤ ਹੋਣਾ ਚਾਹੀਦਾ ਹੈ।ਉਪਰੇਟਰ ਫੰਕਸ਼ਨ ਨੂੰ ਓਵਰਲੋਡਿਡ ਡਾਟਾ–ਟਾਈਪ ਜਿਸ ਵਿਚ ਕੋਈ ਆਰਗੂਮੈਂਟ ਨਾ ਹੋਵੇ, ਦੇ ਰੂਪ ਵਿਚ ਵੀ ਪਰਿਭਾਸ਼ਿਤ ਕੀਤਾ ਜਾ ਸਕਦਾ ਹੈ।

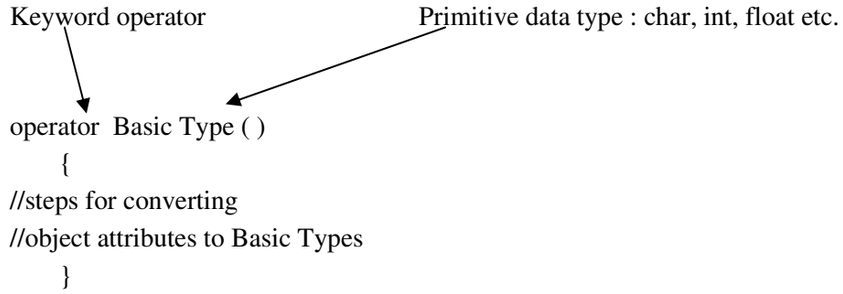


Fig. Conversion Function : User defined to Basic

ਮਿਸਾਲ : meter.cpp

```
//Conversion from Meter to Centimeter and Vice-versa
# include < iostream.h >
//Meter class for MKS measurement system
class Meter
{
    private :
    float length       //kength in meter
    public :
    Meter ( )       //constructor O, no arguments
    {
        length = 0.0 ;
    }

//Conversion for Basic data-item to user-defined Type
//Initlength is in centimeter Unit
Meter (float Initlength) //Constructorl, one argument
{
    Length = Initlength/100.0 ;   //Centimeter to meter
}
//Conversion from user-defined type to Basic data-item
//i.e. from meter to centimeter
    operator float ( )
    cout << * length (in meter = '' << length ;
  }
};
void main ( )
{
    //Basic to user-defined conversion demonstration section
    Meter meter1;   //user constructor 0
    float length1;
    cout << ''Enter length (in cms):'' ;
    cin >> length1 ;
    meter1 = length1; //converts basic to user-defined uses construction
    meter1 . show length ( ) ;
```

```
//user defined to Basic conversion demonstration section
Meter meter 2 ; //uses constructor 0
float length2;
    meter2. (Tet length( ) ;
Length2 = meter2 ;   //converts user = defined to basic uses operator float)
cout << * Length (in cms) = '' << Length 2'' ;
}
```

Output :

Enter length (in cms.) : 1500
Length (in meter) = 1.5
Enter length (in meters) : 1.669
Length (in cms.) = 166.900009

ਯਾਦ ਰੱਖਣ ਯੋਗ ਗੱਲਾਂ (Points to Remember) :

1. ਕਲਾਸ ਇਕ ਤਰੀਕਾ ਹੈ ਜਿਸ ਨਾਲ ਅਸੀਂ ਡਾਟਾ ਅਤੇ ਉਸ ਦੇ ਨਾਲ ਸੰਬੰਧਿਤ ਫੰਕਸ਼ਨਸ ਨੂੰ ਇਕੱਠਾ ਜੋੜਦੇ ਹਾਂ।
2. ਇਕ ਕਲਾਸ ਦੇ ਡਾਟਾ ਮੈਂਬਰ ਅਤੇ ਮੈਂਬਰ ਫੰਕਸ਼ਨ ਸਟੈਟਿਕ ਵੀ ਹੋ ਸਕਦੇ ਹਨ।
3. ਕੰਨਸਟਰਕਟਰ ਇਕ ਖ਼ਾਸ ਮੈਂਬਰ ਫੰਕਸ਼ਨ ਹੈ, ਜਿਸ ਦਾ ਕੰਮ ਕਲਾਸ ਦੇ ਆਬਜੈਕਟ ਨੂੰ ਇਨੀਸ਼ਲਾਈਜ਼ ਕਰਨਾ ਹੈ।
4. ਇਕ ਕਾਪੀ ਕੰਨਸਟਕਟਰ ਇਕ ਆਬਜੈਕਟ ਨੂੰ ਦੂਸਰੇ ਆਬਜੈਕਟ ਤੋਂ ਇਨੀਸ਼ਲਾਈਜ਼ ਕਰਨ ਲਈ ਵਰਤਿਆ ਜਾਂਦਾ ਹੈ।
5. ਵੈਰੀਏਬਲਸ ਦਾ ਉਹ ਐਰੇ ਜਿਸ ਦੀ ਟਾਈਪ ''ਕਲਾਸ'' ਹੋਵੇ ਨੂੰ ਐਰੇ ਆਫ ਆਬਜੈਕਟਸ ਕਿਹਾ ਜਾਂਦਾ ਹੈ।
6. ਡਾਟੇ ਦੀ ਇਕ ਕਿਸਮ ਤੋਂ ਦੂਸਰੀ ਕਿਸਮ ਵਿਚ ਤਬਦੀਲੀ ਨੂੰ ਟਾਈਪ ਕਨਵਰਸ਼ਨ ਕਿਹਾ ਜਾਂਦਾ ਹੈ।

ਅਭਿਆਸ
(Exercise)

1. ਖਾਲੀ ਥਾਵਾਂ ਭਰੋ—

1. ..... ਹਮੇਸ਼ਾ ਪਬਲਿਕ ਸੈਕਸ਼ਨ ਵਿਚ ਡਿਕਲੇਅਰ ਕੀਤੇ ਜਾਂਦੇ ਹਨ।
2. ਆਬਜੈਕਟਸ ਦੀ ਵਿਸ਼ੇਸ਼ਤਾਵਾਂ ਹਨ ..... ਅਤੇ .....
3. ਕਾਪੀ ਕੰਸਟਰਕਟਰ ਦੁਆਰਾ ਇਨੀਸ਼ਲਾਈਜ਼ ਕਰਨ ਦੀ ਪ੍ਰਕਿਰਿਆ ਨੂੰ ..... ਕਿਹਾ ਜਾਂਦਾ ਹੈ।
4. ਮੈਂਬਰ ਫੰਕਸ਼ਨ ਨੂੰ ..... ਤਰੀਕਿਆਂ ਨਾਲ ਪਰਿਭਾਸ਼ਿਤ ਕੀਤਾ ਜਾ ਸਕਦਾ ਹੈ।
5. ਜਦੋਂ ਇਕ ਫੰਕਸ਼ਨ ਨੂੰ ਕਲਾਸ ਦੇ ਅੰਦਰ ਪਰਿਭਾਸ਼ਿਤ ਕੀਤਾ ਜਾਂਦਾ ਹੈ ਤਾਂ ਇਸ ਨੂੰ ..... ਕਿਹਾ ਜਾਂਦਾ ਹੈ।

2. ਪ੍ਰਸ਼ਨਾਂ ਦੇ ਉੱਤਰ ਦਿਉ—

1. ਸਕੋਪ ਰੈਸੋਲਿਉਸ਼ਨ ਉਪਰੇਟਰ (::) Scope Resolution ਉਪਰੇਟਰ ਤੇ ਸੰਖੇਪ ਨੋਟ ਲਿਖੋ।
2. ਸਟੈਟਿਕ ਡਾਟਾ ਮੈਂਬਰ ਅਤੇ ਮੈਂਬਰ ਫੰਕਸ਼ਨਸ ਬਾਰੇ ਜਾਣਕਾਰੀ ਦਿਉ।
3. ਕਲਾਸ ਕਾਨਸੈਪਟ ਬਾਰੇ ਵਿਸਥਾਰ ਨਾਲ ਜਾਣਕਾਰੀ ਦਿਉ।
4. ਕੰਨਸਟਰਕਟਰ ਕੀ ਹੈ ?ਉਦਾਹਰਣ ਸਹਿਤ ਦੱਸੋ। ਇਹ ਡਿਸਟਰਕਟਰ ਨਾਲੋਂ ਕਿਵੇਂ ਵੱਖਰਾ ਹੈ ?
5. ਆਬਜੈਕਟਸ ਅਤੇ ਬੇਸਿਕ ਡਾਟਾ ਟਾਈਪ ਵਿਚ ਕਨਵਰਸ਼ਨ ਦੀ ਪ੍ਰਕਿਰਿਆ ਦੱਸੋ।
6. ਆਬਜੈਕਟ ਅਤੇ ਕਲਾਸ ਵਿਚ ਅੰਤਰ ਦੱਸੋ।
7. ਐਰੇ ਆਫ ਆਬਜੈਕਟਸ ਕਿਸ ਨੂੰ ਕਿਹਾ ਜਾਂਦਾ ਹੈ ?ਉਦਾਹਰਣ ਨਾਲ ਦੱਸੋ।
8. ਨਿਊ (New) ਅਤੇ ਡਿਲੀਟ (Delete) ਓਪਰੇਟਰ ਨੂੰ ਓਵਰਲੋਡ ਕਰਨ ਦੀ ਲੋੜ ਕਿਉਂ ਪੈਂਦੀ ਹੈ ?
9. ਫਰੈਂਡ ਫੰਕਸ਼ਨ ਕੀ ਹੁੰਦੇ ਹਨ ?
10. C++ ਵਿਚ 'this' pointer (ਪੁਆਇੰਟਰ) ਦੀ ਵਰਤੋਂ ਕਿਉਂ ਕੀਤੀ ਜਾਂਦੀ ਹੈ ?

Lab Activity

Classes

Program 1:

```
// classes example
#include <iostream.h>
using namespace std;

class CRectangle {
    int x, y;
  public:
    void set_values (int,int);
    int area () {return (x*y);}
};

void CRectangle::set_values (int a, int b) {
  x = a;
  y = b;
}

int main () {
  CRectangle rect;
  rect.set_values (3,4);
  cout << "area: " << rect.area();
  return 0;
}
```

Program 2:

```
// example: one class, two objects
#include <iostream.h>
using namespace std;

class CRectangle {
    int x, y;
  public:
    void set_values (int,int);
    int area () {return (x*y);}
};

void CRectangle::set_values (int a, int b) {
  x = a;
  y = b;
}

int main () {
  CRectangle rect, rectb;
  rect.set_values (3,4);
  rectb.set_values (5,6);
  cout << "rect area: " << rect.area() << endl;
  cout << "rectb area: " << rectb.area() << endl;
  return 0;
}
```

Program 3:

```
// example: class constructor
#include <iostream.h>
using namespace std;

class CRectangle {
    int width, height;
  public:
    CRectangle (int,int);
    int area () {return (width*height);}
};

CRectangle::CRectangle (int a, int b) {
  width = a;
  height = b;
}

int main () {
  CRectangle rect (3,4);
  CRectangle rectb (5,6);
  cout << "rect area: " << rect.area() << endl;
  cout << "rectb area: " << rectb.area() << endl;
  return 0;
}
```

Program 4:

```
// example on constructors and destructors
#include <iostream.h>
using namespace std;

class CRectangle {
    int *width, *height;
  public:
    CRectangle (int,int);
    ~CRectangle ();
    int area () {return (*width * *height);}
};

CRectangle::CRectangle (int a, int b) {
  width = new int;
  height = new int;
  *width = a;
  *height = b;
}

CRectangle::~CRectangle () {
  delete width;
  delete height;
}

int main () {
  CRectangle rect (3,4), rectb (5,6);
  cout << "rect area: " << rect.area() << endl;
  cout << "rectb area: " << rectb.area() << endl;
  return 0;
}
```

Program 5:

```
// overloading class constructors
#include <iostream.h>
using namespace std;

class CRectangle {
    int width, height;
  public:
    CRectangle ();
    CRectangle (int,int);
    int area (void) {return (width*height);}
};

CRectangle::CRectangle () {
  width = 5;
  height = 5;
}

CRectangle::CRectangle (int a, int b) {
  width = a;
  height = b;
}

int main () {
  CRectangle rect (3,4);
  CRectangle rectb;
  cout << "rect area: " << rect.area() << endl;
  cout << "rectb area: " << rectb.area() << endl;
  return 0;
}
```

Program 6:

```
// pointer to classes example
#include <iostream.h>
using namespace std;

class CRectangle {
    int width, height;
  public:
    void set_values (int, int);
```

```
    int area (void) {return (width * height);}
};


void CRectangle::set_values (int a, int b) {
  width = a;
  height = b;
}

int main () {
  CRectangle a, *b, *c;
  CRectangle * d = new CRectangle[2];
  b= new CRectangle;
  c= &a;
  a.set_values (1,2);
  b->set_values (3,4);
  d->set_values (5,6);
  d[1].set_values (7,8);
  cout << "a area: " << a.area() << endl;
  cout << "*b area: " << b->area() << endl;
  cout << "*c area: " << c->area() << endl;
  cout << "d[0] area: " << d[0].area() << endl;
  cout << "d[1] area: " << d[1].area() << endl;
  delete[] d;
  delete b;
  return 0;
}
```

Classes -II

Program 7:

```
// vectors: overloading operators example
#include <iostream.h>
using namespace std;

class CVector {
  public:
    int x,y;
    CVector () {};
    CVector (int,int);
    CVector operator + (CVector);
};

CVector::CVector (int a, int b) {
  x = a;
  y = b;
}

CVector CVector::operator+ (CVector param) {
  CVector temp;
  temp.x = x + param.x;
  temp.y = y + param.y;
  return (temp);
}

int main () {
  CVector a (3,1);
  CVector b (1,2);
  CVector c;
  c = a + b;
  cout << c.x << "," << c.y;
  return 0;
}
```

Program 8:

```
// this
#include <iostream.h>
using namespace std;

class CDummy {
  public:
    int isitme (CDummy& param);
```

```
};

int CDummy::isitme (CDummy& param)
{
  if (&param == this) return true;
  else return false;
}

int main () {
  CDummy a;
  CDummy* b = &a;
  if ( b->isitme(a) )
    cout << "yes, &a is b";
  return 0;
}
```

Program 9:

```
// static members in classes
#include <iostream.h>
using namespace std;

class CDummy {
  public:
    static int n;
    CDummy () { n++; };
    ~CDummy () { n--; };
};

int CDummy::n=0;

int main () {
  CDummy a;
  CDummy b[5];
  CDummy * c = new CDummy;
  cout << a.n << endl;
  delete c;
  cout << CDummy::n << endl;
  return 0;
}
```

Pointers

Program 10:

```
// my first pointer
#include <iostream.h>
using namespace std;

int main ()
{
  int firstvalue, secondvalue;
  int * mypointer;

  mypointer = &firstvalue;
  *mypointer = 10;
  mypointer = &secondvalue;
  *mypointer = 20;
  cout << "firstvalue is " << firstvalue << endl;
  cout << "secondvalue is " << secondvalue << endl;
  return 0;
}
```

Program 11:

```
// more pointers
#include <iostream.h>
using namespace std;

int main ()
{
  int firstvalue = 5, secondvalue = 15;
```

```
  int * p1, * p2;

  p1 = &firstvalue;  // p1 = address of firstvalue
  p2 = &secondvalue; // p2 = address of secondvalue
  *p1 = 10;          // value pointed by p1 = 10
  *p2 = *p1;         // value pointed by p2 = value pointed by p1
  p1 = p2;           // p1 = p2 (value of pointer is copied)
  *p1 = 20;          // value pointed by p1 = 20

  cout << "firstvalue is " << firstvalue << endl;
  cout << "secondvalue is " << secondvalue << endl;
  return 0;
}
```

Program 12:

```
// more pointers
#include <iostream.h>
using namespace std;

int main ()
{
  int numbers[5];
  int * p;
  p = numbers;  *p = 10;
  p++;  *p = 20;
  p = &numbers[2];  *p = 30;
  p = numbers + 3;  *p = 40;
  p = numbers;  *(p+4) = 50;
  for (int n=0; n<5; n++)
    cout << numbers[n] << ", ";
  return 0;
}
```

Program 13:

```
// increaser
#include <iostream.h>
using namespace std;

void increase (void* data, int psize)
{
  if ( psize == sizeof(char) )
  { char* pchar; pchar=(char*)data; ++(*pchar); }
  else if (psize == sizeof(int) )
  { int* pint; pint=(int*)data; ++(*pint); }
}

int main ()
{
  char a = 'x';
  int b = 1602;
  increase (&a,sizeof(a));
  increase (&b,sizeof(b));
  cout << a << ", " << b << endl;
  return 0;
}
```

Program 14:

```
// pointer to functions
#include <iostream.h>
using namespace std;
```



```
int addition (int a, int b)
{ return (a+b); }
```

```
int subtraction (int a, int b)
{ return (a-b); }

int operation (int x, int y, int (*functocall)(int,int))
{
  int g;
  g = (*functocall)(x,y);
  return (g);
}

int main ()
{
  int m,n;
  int (*minus)(int,int) = subtraction;

  m = operation (7, 5, addition);
  n = operation (20, m, minus);
  cout <<n;
  return 0;
}
```

Destructor

```
#include <iostream.h>
#include <cstdlib.h>
using namespace std;

class myclass {
  int *p;
public:
  myclass(int i);
  ~myclass();
  int getval() { return *p; }
};

myclass::myclass(int i)
{
  cout << "Allocating p\n";
  p = new int;
  if(!p) {
    cout << "Allocation failure.\n";
    exit(1); // exit program if out of memory
  }

  *p = i;
}

// use destructor to free memory
myclass::~myclass()
{
  cout << "Freeing p\n";
  delete p;
}

void display(myclass &ob)
{
  cout << ob.getval() << '\n';
}
int main()
{
  myclass a(10);

  display(a);

  return 0;
}
```

Exercise

1. Find out the errors if any:

```
   (a)  #include <iostream.h>
using namespace std;

class cl {
```

```
  int i; // private by default
public:
  int get_i();
  void put_i( j);
};

int cl::get_i()
{
  return i;
}

void cl::put_i(j)
{
  i = j;
}

int main()
{
  cl s;

  s.put_i(10);
  cout << s.get_i() <<endl;

  return 0;
}
```

(b)

```
#include <iostream.h>
using namespace std;

// Class to represent a box
class Box
{
  public:
    double length;
    double breadth;
    double height;

    // Inline initialization
    Box(double lv = 1.0, double bv = 1.0, double hv = 1.0):length(lv),
                                                           breadth(bv),
                                                           height(hv)
    {
      cout << "Box constructor called" << endl;
    }

    // Function to calculate the volume of a box
    double volume()
    {
      return length * breadth * height;
    }
};

int main()
{
  Box firstBox(80.0, 50.0, 40.0);

  // Calculate the volume of the box
  double firstBoxVolume = firstBox.volume()
  cout << endl;
  cout << "Size of first Box object is "
       << firstBox.length  << " by "
       << firstBox.breadth << " by "
        << firstBox.height
       << endl;
  cout << "Volume of first Box object is " << firstBoxVolume
  return 0;
}
```

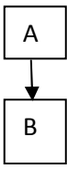
ਪਾਠ-4
ਇਨਹੈਰੀਟੈਂਸ

4.1 ਇਨਹੈਰੀਟੈਂਸ

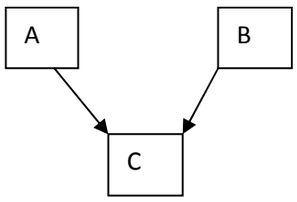
C++ ਮੁੜ-ਵਰਤੋਂ ਜਾਂ ਬਾਰ-ਬਾਰ ਇਸਤੇਮਾਲ ਦੇ ਸਿਧਾਂਤ ਦਾ ਕਠੋਰਤਾ ਨਾਲ ਸਾਥ ਦਿੰਦੀ ਹੈ।
C++ ਦੀਆਂ ਕਲਾਸਿਜ (classes) ਨੂੰ ਵੱਖ-ਵੱਖ ਤਰੀਕਿਆਂ ਨਾਲ ਮੁੜ ਇਸਤੇਮਾਲ ਕੀਤਾ ਜਾ ਸਕਦਾ ਹੈ। ਇਕ ਬਾਰ ਕਿਸੇ ਕਲਾਸ (class) ਨੂੰ ਲਿਖਣ ਅਤੇ ਟੈਸਟ ਕਰਨ ਉਪਰੰਤ ਹੋਰਨਾਂ ਪ੍ਰੋਗਰਾਮਰਜ਼ ਦੁਆਰਾ ਆਪਣੀ ਜ਼ਰੂਰਤ ਅਨੁਸਾਰ ਇਸਤੇਮਾਲ ਕੀਤਾ ਜਾ ਸਕਦਾ ਹੈ ਜਾਂ ਵਰਤੋਂ ਵਿਚ ਲਿਆਇਆ ਜਾ ਸਕਦਾ ਹੈ। ਮੁੱਢਲੇ / ਮੁੱਖ ਤੌਰ ਉੱਤੇ ਇਸ ਤਰ੍ਹਾਂ ਨਵੀਂ ਕਲਾਸ (class) ਤਿਆਰ ਕਰਕੇ ਪਹਿਲਾਂ ਮੌਜੂਦਾ ਕਲਾਸ (class) ਦੀਆਂ ਵਿਸ਼ੇਸ਼ਤਾਵਾਂ (Properties) ਨੂੰ ਮੁੜ ਇਸਤੇਮਾਲ ਕੀਤਾ ਜਾਂਦਾ ਹੈ। ਉਹ ਪ੍ਰਕਿਰਿਆ ਜਿਸ ਰਾਹੀਂ ਨਵੀਂ ਕਲਾਸ (class) ਨੂੰ ਪੁਰਾਣੀ ਕਲਾਸ ਵਿਚੋਂ ਤਿਆਰ ਕੀਤਾ ਜਾਂਦਾ ਹੈ, ਇਨਹੈਰੀਟੈਂਸ ਅਖਵਾਉਂਦੀ ਹੈ।
ਪੁਰਾਣੀ ਕਲਾਸ ਨੂੰ ਬੇਸ (base) ਕਲਾਸ ਆਖਿਆ ਜਾਂਦਾ ਹੈ ਅਤੇ ਨਵੀਂ ਕਲਾਸ ਨੂੰ ਡਿਰਾਈਵਡ/ਸਬ ਕਲਾਸ ਕਿਹਾ ਜਾਂਦਾ ਹੈ।

4.1.1 ਡਿਰਾਈਵਡ ਕਲਾਸ ਵਿਚ ਕੀ ਹੁੰਦਾ ਹੈ ?

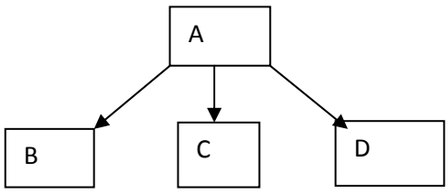
ਡਿਰਾਈਵਡ ਕਲਾਸ ਬੇਸ (Base) ਕਲਾਸ ਤੋਂ ਕੁਝ ਜਾਂ ਸਾਰੀਆਂ ਵਿਸ਼ੇਸ਼ਤਾਵਾਂ ਗ੍ਰਹਿਣ ਕਰਦੀ ਹੈ। ਇਕ ਕਲਾਸ ਵਿਚ ਕਈ ਵਾਰ ਉਸਦੀਆਂ ਇਕ ਤੋਂ ਜ਼ਿਆਦਾ ਵਿਸ਼ੇਸ਼ਤਾਵਾਂ ਪਾਸ ਹੁੰਦੀਆਂ ਹਨ। ਇਸ ਡਿਰਾਈਵਡ ਕਲਾਸ ਜਿਸ ਦੀ ਇਕੋ ਬੇਸ ਕਲਾਸ ਨੂੰ ਸਿੰਹਾਲ ਹੈਰੀਟੈਨਸ (heritance) ਅਤੇ ਇਕ ਦੇ ਨਾਲ ਜਿਆਦਾ ਬੇਸ ਕਲਾਸਾਂ ਨੂੰ ਮਲਟੀਪਲ ਇਨਹੈਰੀਟੈਨਸ (Multiple Inheritance) ਕਿਹਾ ਜਾਂਦਾ ਹੈ। ਇਕ ਕਲਾਸ ਦੇ ਟਰੇਟਸ (traits) ਇਕ ਤੋਂ ਜ਼ਿਆਦਾ ਕਲਾਸ ਤੋਂ ਵੀ ਗ੍ਰਹਿਣ ਕੀਤੇ ਜਾਂਦੇ ਹਨ। ਇਸ ਪ੍ਰੋਸੈਸ ਨੂੰ ਹਾਈਰੈਰੀਕਲ ਇਨਹੈਰੀਟੈਂਸ (hierarchical inheritance) ਕਿਹਾ ਜਾਂਦਾ ਹੈ।
ਉਹ ਵਿਧੀ ਜਿਸ ਵਿਚ ਇਕ ਕਲਾਸ ਨੂੰ ਦੂਸਰੀ ਡਿਰਾਈਵਡ ਕਲਾਸ ਵਿਚ ਡਿਰਾਈਵ ਕੀਤਾ ਜਾਂਦਾ ਹੈ। ਉਸ ਨੂੰ ਮਲਟੀਲੈਵਲ ਇਨਹੈਰੀਟੈਂਸ ਕਿਹਾ ਜਾਂਦਾ ਹੈ।



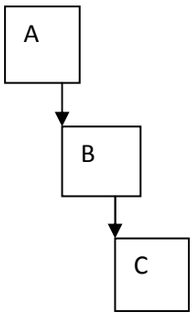
Single Inheritance



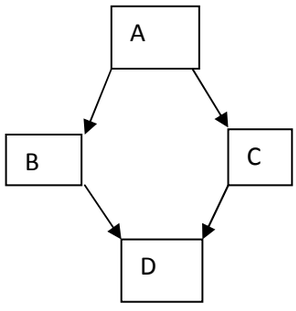
Multiple Inheritance



Hierarchical Inheritance



Multilevel Inheritance



Hybrid Inheritance

ਅਸੀਂ ਇਸ ਬਾਰੇ ਵਿਸਥਾਰ ਨਾਲ ਅੱਗੇ ਪੜਾਗੇ।

4.1.2 ਡਰਾਇਵਡ ਕਲਾਸ (Derived class) ਪਰਿਭਾਸ਼ਿਤ ਕਰਨਾ

ਇਕ ਆਮ ਡਿਰਾਈਵਡ ਕਲਾਸ ਨੂੰ ਪਰਿਭਾਸ਼ਿਤ ਕਰਨ ਦਾ ਤਰੀਕਾ ਇਸ ਤਰ੍ਹਾਂ ਹੈ——

```
class derived-class-name : visibility-mode
                        {
                                .....//
                                .....//  members of derived class
                                .....//
                        };
```

4.4 ਵਿਜੀਬਿਲਟੀ ਮੋਡ ਜਾਂ ਅਸੈਸ ਸਪੈਸੀਫਾਇਰ (Visibility Mode or Access Specifier)

ਇਹ ਸਪੱਸ਼ਟ ਕਰਦਾ ਹੈ ਕਿ ਬੇਸ ਕਲਾਸ ਦੇ ਗੁਣ ਪ੍ਰਾਈਵਟ ਤੌਰ ਤੇ ਡਿਰਾਈਵਡ ਹੁੰਦੇ ਹਨ ਜਾਂ ਪਬਲਿਕ ਤੌਰ ਤੇ ਡਿਰਾਈਵਡ ਹੁੰਦੇ ਹਨ।

1. class ABC : private XYZ
   {
   members of ABC // private derivation
   };
2. class ABC : public XYZ
   {
   members of ABC //public derivation
   };
3. class ABC : XYZ
   {
   members of ABC //private derivation by default
   };

4.2.1 private Derivation

ਜਦੋਂ ਇਕ ਬੇਸ ਕਲਾਸ ਨਿੱਜੀ ਤੌਰ ਤੇ ਡਿਰਾਈਵਡ ਕਲਾਸ ਤੋਂ ਇਨਹੈਰਿਟ ਹੁੰਦੀ ਹੈ ਤਾਂ ਬੇਸ ਕਲਾਸ ਦੇ ਪਬਲਿਕ ਮੈਂਬਰ ਡਿਰਾਈਵਡ ਕਲਾਸ ਦੇ ਪ੍ਰਾਈਵੇਟ ਮੈਂਬਰ ਬਣ ਜਾਂਦੇ ਹਨ। ਇਸ ਲਈ ਬੇਸ ਕਲਾਸ ਦੇ ਪਬਲਿਕ ਮੈਂਬਰ ਡਿਰਾਈਵਡ ਕਲਾਸ ਦੇ ਫੰਕਸ਼ਨ ਮੈਂਬਰ ਤੋਂ ਹੀ ਅਸੈਸ ਕੀਤੇ ਜਾ ਸਕਦੇ ਹਨ।

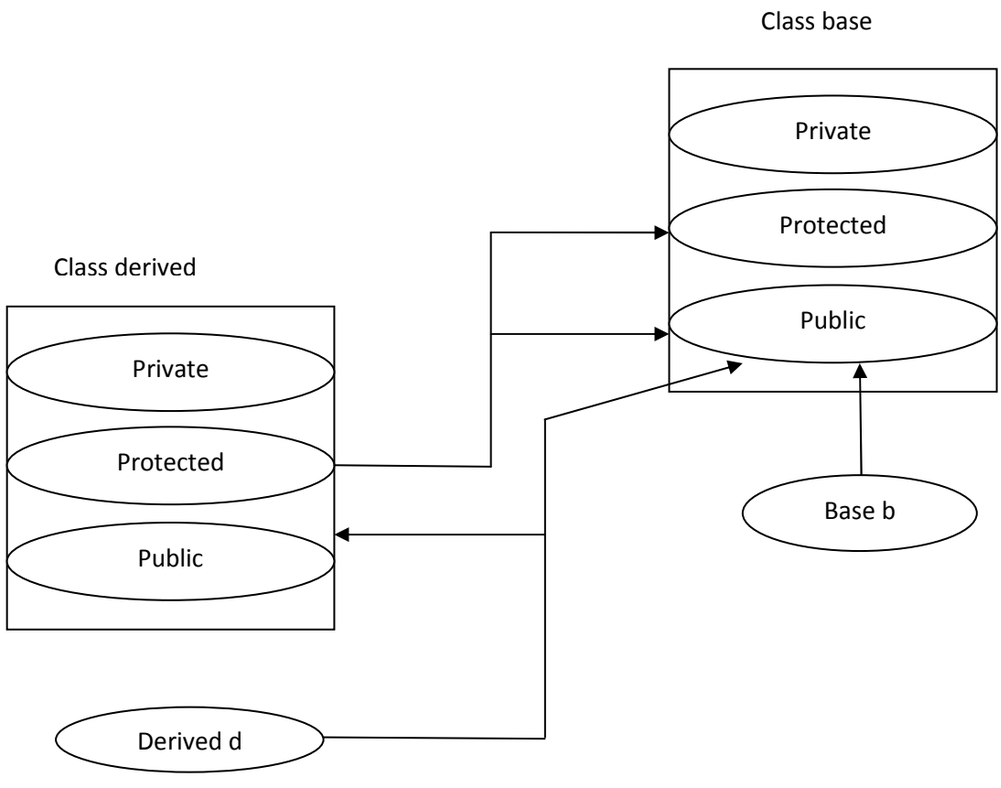
ਉਹ ਡਿਰਾਈਵਡ ਕਲਾਸ ਦੇ ਆਬਜੈਕਟ (Objects) ਨੂੰ ਅਸੈਸ ਨਹੀਂ ਕਰ ਸਕਦੇ।

4.2.2 ਪਬਲਿਕ ਡੈਰੀਵੇਸ਼ਨ (public Derivation)

ਜਦੋਂ ਬੇਸ ਕਲਾਸ ਪਬਲਿਕ ਤੌਰ ਤੇ ਇਨਹੈਰਿਟ ਹੁੰਦੀ ਹੈ ਤਾਂ ਬੇਸ ਕਲਾਸ ਦੇ ਪਬਲਿਕ ਮੈਂਬਰਜ ਡਿਰਾਈਵਡ ਕਲਾਸ ਦੇ ਪਬਲਿਕ ਮੈਂਬਰਜ਼ ਬਣ ਜਾਂਦੇ ਹਨ।

4.2.3 ਪ੍ਰੋਟੈਕਟਿਡ ਡੈਰੀਵੇਸ਼ਨ

ਇਕ ਪ੍ਰੋਟੈਕਟਿਡ ਮੈਂਬਰ ਆਪਣੀ ਹੀ ਕਲਾਸ ਵਿਚ ਆਪਣੇ ਹੀ ਫੰਕਸ਼ਨ ਦੇ ਮੈਂਬਰ ਤੋਂ ਅਸੈਸ ਕੀਤੇ ਜਾ ਸਕਦੇ ਹਨ।



4.2.4 ਵਰਚੂਅਲ ਡੈਰੀਵੇਸ਼ਨ

ਵਰਚੂਅਲ ਡੈਰੀਵੇਸ਼ਨ ਨੂੰ ਅਸੀਂ ਹੇਠਾਂ ਲਿਖੀ ਉਦਾਹਰਣ ਨਾਲ ਸਮਝ ਸਕਦੇ ਹਾਂ——

```
class A        //grand parent
{
     .................
     .................
};
class B1 : virtual public A  //Parent 1
{
     .................
     .................
};
class B2 : public virtual B  //Parent 2
{
     .................
     .................
};
class C : public B1, public B2   //Child
{
     .................  //only one copy of A
     .................  //will be inherited
};
```

ਜਦੋਂ ਇਕ ਕਲਾਸ ਨੂੰ ਵਰਚੂਅਲ ਬੇਸ class ਬਣਾਇਆ ਜਾਂਦਾ ਹੈ ਤਾਂ C++ ਵਿਚ ਧਿਆਨ ਰੱਖਿਆ ਜਾਂਦਾ ਹੈ ਕਿ ਕਲਾਸ ਦੀ ਇਕ ਹੀ ਕਾਪੀ (copy) ਇਨਹੈਰਿਟ (inherit) ਹੋਵੇ ਚਾਹੇ ਵਰਚੂਅਲ ਬੇਸ ਕਲਾਸ ਅਤੇ ਡਿਰਾਈਵਡ ਬੇਸ ਕਲਾਸ ਵਿਚ ਇਕ ਤੋਂ ਵੱਧ ਇਨਹੈਰਿਟੈਂਸ ਪਾਥ (path) ਹੋਵੇ। ਕੀਵਰਡ virtual ਅਤੇ public ਕਿਸੇ ਵੀ ਲੜੀ ਵਿਚ ਵਰਤੇ ਜਾ ਸਕਦੇ ਹਨ।

4.3 ਫੰਕਸ਼ਨ ਓਵਰਰਾਈਡਿੰਗ

ਜਿਵੇਂ ਕਿ ਅਸੀਂ ਪੜ੍ਹ ਚੁੱਕੇ ਹਾਂ ਕਿ ਓਵਰਲੋਡਿੰਗ ਉਸ ਸਮੇਂ ਹੋਈ ਹੈ ਜਦੋਂ ਦੋ ਫੰਕਸ਼ਨ ਇਕੋ ਹੀ ਨਾਮ ਦੇ ਨਾਲ ਬਣਾਏ ਜਾਂਦੇ ਹਨ ਪਰ ਜਾਂ ਤਾਂ ਆਰਗੂਮੈਂਟ (argument) ਟਾਈਪ ਅਲੱਗ ਹੁੰਦੀ ਹੈ ਜਾਂ ਫਿਰ ਨੰਬਰ ਆਫ ਆਰਗੂਮੈਂਟ (argument) ਅਲਗ ਹੁੰਦੇ ਹਨ। ਕੰਪਾਈਲਰ ਨੂੰ ਹਰ ਕੇਸ ਵਿਚ ਕੋਡ ਜਨਰੇਟ (generate) ਕਰਨਾ ਪੈਂਦਾ ਹੈ। ਇਹ ਕੰਮ ਕਲਾਸ ਦੇ ਅੰਦਰ ਵੀ ਹੋ ਸਕਦਾ ਹੈ ਤੇ ਕਲਾਸ ਦੇ ਬਾਹਰ ਵੀ।

ਓਵਰਰਾਈਡਿੰਗ ਉਸ ਸਮੇਂ ਹੁੰਦੀ ਹੈ ਜਦੋਂ ਤੁਸੀਂ ਚਾਈਲਡ (child) ਕਲਾਸ ਨੂੰ ਬੇਸ ਕਲਾਸ ਤੋਂ ਡਰਾਈਵ (Drive) ਕਰਦੇ ਹੋ ਤੇ ਨਾਲ ਹੀ ਬੇਸ ਕਲਾਸ ਦੇ ਮੈਥਡ (method) ਜਾਂ ਫੰਕਸ਼ਨ ਨੂੰ ਚਾਈਲਡ (child) ਕਲਾਸ ਦੇ ਫੰਕਸ਼ਨ (ਜਿਸ ਦੇ ਆਰਗੂਮੈਂਟ (argument) ਇਕੋ ਜਿਹੇ ਨੰਬਰ ਜਾਂ ਇਕੋ ਟਾਈਪ ਦੇ ਹੋਣ) ਤੋਂ ਰੀਪਲੇਸ (replace) ਜਾਂ ਬਦਲ ਦਿੰਦੇ ਹੋ। ਇਹ ਓਵਰਲੋਡਿੰਗ ਨਹੀਂ ਹੁੰਦੀ ਪਰ ਇਹ ਰੀਡੈਕਲੇਰੇਸ਼ਨ ਹੁੰਦੀ ਹੈ।

4.4 ਓਵਰਲੋਡਿੰਗ vs ਓਵਰਰਾਈਡਿੰਗ

1. ਓਵਰਲੋਡਿੰਗ ਆਫ ਫੰਕਸ਼ਨ ਉਸ ਸਮੇਂ ਹੁੰਦੀ ਹੈ ਜਦੋਂ ਇਕ ਕਲਾਸ ਨੂੰ ਦੂਜੀ ਕਲਾਸ ਤੋਂ ਇਨਹੈਰਿਟ (Inherit) ਕੀਤਾ ਜਾਂਦਾ ਹੈ। ਓਵਰਲੋਡਿੰਗ ਬਿਨਾਂ ਇਨਹੈਰੀਟੈਂਸ ਦੇ ਵੀ ਹੋ ਸਕਦੀ ਹੈ।
2. ਓਵਰਲੋਡਿੰਗ ਵਿਚ ਫੰਕਸ਼ਨ ਸਿਗਨੇਚਰ (Signature) ਅਲਗ ਹੋਣੀ ਚਾਹੀਦੀ ਹੈ ਜਿਸ ਦਾ ਅਰਥ ਹੈ ਕਿ ਨੰਬਰ ਆਫ ਆਰਗੂਮੈਂਟ ਜਾਂ ਟਾਈਪ ਆਫ਼ ਆਰਗੂਮੈਂਟ ਅਲਗ ਹੋਣੇ ਚਾਹੀਦੇ ਹਨ। ਓਵਰਰਾਈਡਿੰਗ ਵਿਚ ਫੰਕਸ਼ਨ ਸਿਗਨੇਚਰ ਇਕ ਸਮਾਨ ਹੋਣਾ ਚਾਹੀਦਾ ਹੈ।
3. ਓਵਰਰਾਈਡਿੰਗ ਫੰਕਸ਼ਨ ਸਕੋਪ (scope) ਵਿਚ ਵੀ ਅਲਗ ਹੁੰਦੇ ਹਨ ਪਰ ਓਵਰਲੋਡਿੰਗ ਫੰਕਸ਼ਨ ਇਕ ਹੀ ਸਕੋਪ ਵਿਚ ਹੁੰਦੇ ਹਨ।
4. ਜਦੋਂ ਡਿਰਾਈਵਡ ਕਲਾਸ ਨੇ ਬੇਸ ਕਲਾਸ ਤੇ ਕੁਝ ਹੋਰ ਜਾਂ ਅਲੱਗ ਕੰਮ ਕਰਨਾ ਹੋਵੇ ਤਾਂ ਓਵਰਰਾਈਡਿੰਗ ਦੀ ਜ਼ਰੂਰਤ ਪੈਂਦੀ ਹੈ।
5. ਓਵਰਲੋਡਿੰਗ ਦੀ ਵਰਤੋਂ ਉਸ ਸਮੇਂ ਕੀਤੀ ਜਾਂਦੀ ਹੈ ਜਦੋਂ ਸਾਨੂੰ ਇਕ ਹੀ ਨਾਮ ਦੇ ਫੰਕਸ਼ਨ ਚਾਹੀਦੇ ਹੁੰਦੇ ਹਨ ਪਰ ਉਹਨਾਂ ਦੇ ਆਰਗੂਮੈਂਟ ਅਲਗ ਹੁੰਦੇ ਹਨ।

ਓਵਰਲੋਡਿੰਗ ਉਹ ਪਰਿਭਾਸ਼ਾ ਹੈ ਜਿਸ ਵਿਚ ਇਕ ਨਾਮ ਦੇ ਕਈ ਫੰਕਸ਼ਨ ਹੁੰਦੇ ਹਨ ਪਰ ਉਹਨਾਂ ਦੇ ਆਰਗੂਮੈਂਟ ਜਾਂ ਨੰਬਰ ਆਫ ਆਰਗੂਮੈਂਟ ਅਲਗ ਹੁੰਦੇ ਹਨ।

ਉਦਾਹਰਣ——

```
void Foo(int);
void Foo(int, int) ;
void Foo(char) ;
```

ਓਵਰਰਾਈਡਿੰਗ ਵਿਚ ਇਕ ਫੰਕਸ਼ਨ ਜੋ ਕਿ ਬੇਸ ਕਲਾਸ ਵਿਚ ਪਰਿਭਾਸ਼ਿਤ ਹੈ ਉਸਦੀ ਨਵੀਂ ਪਰਿਭਾਸ਼ਾ ਡਿਰਾਇਵਡ ਕਲਾਸ ਵਿਚ ਲਿਖੀ

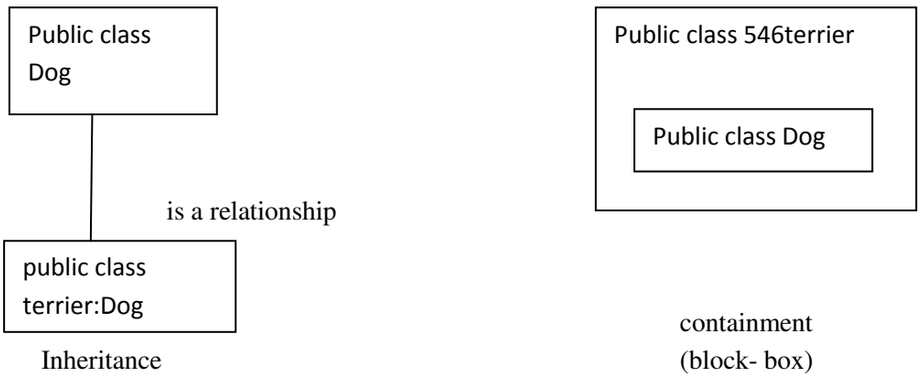
ਜਾਂਦੀ ਹੈ।

ਉਦਾਹਰਣ—

```
class base
{
    public :
    int Foo (  )
    {
        return 1 ;
    }
};
class derived : public base
{
    public :
    int Foo ( )
    {
        return 2;
    }
};
```

ਇਕ ਬੇਸ ਕਲਾਸ ਵਿਚ ਕਈ ਓਵਰਲੋਡਿਡ ਫੰਕਸ਼ਨ ਹੋ ਸਕਦੇ ਹਨ ਜੋ ਕਿ ਡਿਰਾਈਵਡ ਕਲਾਸ ਵਿਚ ਓਵਰਰਾਈਡ ਕੀਤੇ ਜਾ ਸਕਦੇ ਹਨ।

4.5 ਇਨਹੈਰੀਟੈਨਸ v/s ਕਨਟੈਨਮੈਂਟ (Inheritance v/s Containment)



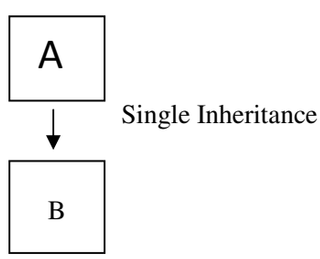
ਅਸੀਂ ਇਨਹੈਰੀਟੈਨਸ ਨੂੰ ਵਾਈਟ ਬਾਕਸ (White Box) ਵੀ ਕਹਿ ਸਕਦੇ ਹਾਂ ਜੋ ਕੋਡ ਦੀ ਮੁੜ ਵਰਤੋਂ (reuse) ਕਰਦਾ ਹੈ ਜਿਸਦਾ ਅਰਥ ਹੈ ਕਿ ਡਿਰਾਈਵਡ ਕਲਾਸ ਵਿਚ ਬੇਸ ਕਲਾਸ ਦੇ ਕੁਝ ਅੰਦਰੂਨੀ ਗੁਣ ਸ਼ਾਮਿਲ ਹੁੰਦੇ ਹਨ। ਇਹ ਐਨਕੇਪਸੂਲੇਸ਼ਨ (Encapsulation) ਦੇ ਸਿਧਾਂਤ ਨੂੰ ਦੱਸਦਾ ਹੈ।

C++ ਵਿਚ ਕੋਡ ਦੀ ਮੁੜ ਵਰਤੋਂ ਦਾ ਇਕ ਹੋਰ ਤਰੀਕਾ ਵੀ ਹੈ ਜਿਸ ਨੂੰ ਕਨਟੈਨਮੈਂਟ (Containment) ਕਹਿੰਦੇ ਹਾਂ ਅਤੇ ਇਹ ਕੋਡ ਦੀ ਮੁੜ ਵਰਤੋਂ (reuse) ਦਾ ਬਲੈਕ ਬਾਕਸ (Black-box) ਕਹਾਉਂਦਾ ਹੈ। ਬੇਸ ਕਲਾਸ ਤੋਂ ਡਰਾਈਵ ਕਰਨ ਦੀ ਥਾਂ ਤੇ, ਇਕ ਨਵੀਂ ਕਲਾਸ ਬੇਸ ਕਲਾਸ ਦਾ ਸਿਰਫ ਇਕ ਇਨਸਟੈਨਸ (Instance) ਬਣਾ (create) ਲੈਂਦੀ ਹੈ ਤੇ ਇਸ ਇਨਸਟੈਨਸ (instance) ਦੇ ਜਰੀਏ ਨਵੀਂ ਕਲਾਸ ਬੇਸ ਕਲਾਸ ਦੇ ਪਬਲਿਕ ਮੈਂਬਰਸ (public Members) ਨੂੰ ਅਸੈਸ (Access) ਕਰ ਸਕਦੀ ਹੈ। ਬੇਸ ਕਲਾਸ ਦੇ ਇਨਸਟੈਨਸ (Instance) ਨੂੰ ਬਲੈਕ ਬਾਕਸ ਕਿਹਾ ਜਾ ਸਕਦਾ ਹੈ ਕਿਉਂਕਿ ਡਰਾਈਵ ਕਲਾਸ ਦੀ ਇੰਪਲੀਮੈਨਟੇਸ਼ਨ (Implementation) ਡੀਟੇਲ (detail) ਉੱਤੇ ਕੋਈ ਅਸੈਸ (access) ਨਹੀਂ ਰਹਿੰਦਾ।

ਨੋਟ : ਅਸੀਂ ਇਕ ਕਲਾਸ ਦੇ ਅੰਦਰ ਆਬਜੈੱਕਟ ਦੇ ਕਈ ਇੰਨਸਟੈਨਸ (Instance) Create ਬਣਾ ਸਕਦੇ ਹਾਂ।

4.6 ਸਿੰਗਲ ਇਨਹੈਰੀਟੈਨਸ (Single Inheritance)

ਇਕ ਡਰਾਈਵ ਕਲਾਸ ਜਿਸ ਦੀ ਸਿਰਫ ਇਕ ਬੇਸ ਕਲਾਸ ਹੁੰਦੀ ਹੈ, ਸਿੰਗਲ ਇਨਹੈਰੀਟੈਨਸ ਕਹਿਲਾਉਂਦੀ ਹੈ।



ਆਓ ਅਸੀਂ ਇਸ ਨੂੰ ਇਕ ਮਿਸਾਲ ਦੁਆਰਾ ਸਮਝਦੇ ਹਾਂ। ਹੇਠਾਂ ਲਿਖੇ ਪ੍ਰੋਗਰਾਮ ਵਿਚ ਇਕ ਬੇਸ ਕਲਾਸ B ਹੈ ਤੇ ਡਰਾਈਵ ਕਲਾਸ D ਹੈ। B ਕਲਾਸ ਵਿਚ ਇਕ ਪ੍ਰਾਈਵੇਟ ਡਾਟਾ ਮੈਂਬਰ, ਇਕ ਪਬਲਿਕ ਡਾਟਾ ਮੈਂਬਰ ਤੇ ਤਿੰਨ ਪਬਲਿਕ ਮੈਂਬਰ ਫੰਕਸ਼ਨ ਹਨ। ਕਲਾਸ D ਵਿਚ ਇਕ ਪ੍ਰਾਈਵੇਟ ਡਾਟਾ ਮੈਂਬਰ ਤੇ ਦੋ ਪਬਲਿਕ ਮੈਂਬਰ ਫੰਕਸ਼ਨ ਹਨ।

```
# include <iostream.h>
class B
{
    int a ;             //private ; note inheritable
    public :
    int b;              //public : ready for inheritance
    void get_ab ( ) ;
    int get_a (void) ;
    void show_a (void) ;
};
class D : public B    //public derivation
{
    int C ;
    public :
        void mul (void) ;
        void display (void) ;
};
void B :: get_ab (void)
{
    a = 5 ; b = 10;
}
int B :: get_a ( )
{
    return a;
}
void B :: show_a ( )
{
cout << ‘‘a = ’’ << a << ‘‘\n’’ ;
}
void D :: mul ( )
{
c = b* get_a ( )
}
void D :: display ( )
{
    cout << ‘‘a =’’ << get_a ( ) << ‘‘\n’’ ;
    cout << ‘‘b =’’ << b << ‘‘\n’’ ;
    cout << ‘‘c =’’ << c << ‘‘\n\n’’
}
//.....
int main ( )
{
    D d;
    d.get_ab ( ) ;
    d.mul ( ) ;
    d.show_a ( )
    d.display ( ) ;
    d.b = 20 ;
    d.mul ( );
    d.display ( );
    return 0 ;
```

Output :

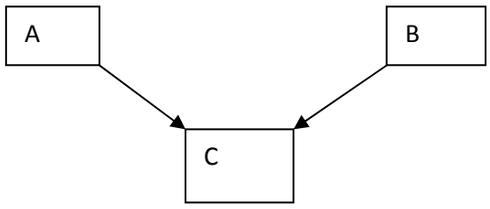
```
a = 5
a = 5
b = 10
c = 50
```

a = 5
b = 20
c = 100

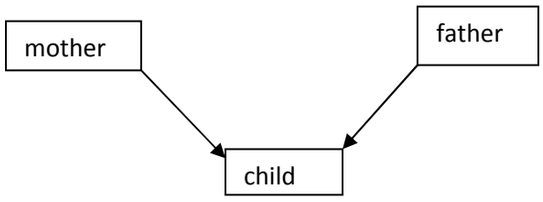
ਇੱਥੇ ਕਲਾਸ D, ਕਲਾਸ B ਦੀ ਪਬਲਿਕ ਡੇਰੀਵੇਸ਼ਨ ਹੈ। ਇਸ ਕਰਕੇ D, B ਦੇ ਸਾਰੇ ਪਬਲਿਕ ਮੈਂਬਰਸ ਨੂੰ ਇਨਹੈਰੀਟ ਕਰ ਰਹੀ ਹੈ। ਇਸ ਤਰ੍ਹਾਂ B ਕਲਾਸ (ਬੇਸ ਕਲਾਸ) ਦਾ ਪਬਲਿਕ ਮੈਂਬਰ, D ਕਲਾਸ (ਡਿਰਾਈਵ ਕਲਾਸ) ਦਾ ਵੀ ਪਬਲਿਕ ਮੈਂਬਰ ਹੈ। B ਕਲਾਸ ਦੇ ਪ੍ਰਾਈਵੇਟ ਮੈਂਬਰ, D ਕਲਾਸ ਦੁਆਰਾ ਇਨਹੈਰੀਟ ਨਹੀਂ ਕੀਤੇ ਜਾ ਸਕਦੇ।

4.7 ਮਲਟੀਪਲ ਇਨਹੈਰੀਟੈਨਸ (Multiple Inheritance)

ਇਕ ਡਰਾਈਵ ਕਲਾਸ ਜਿਸ ਦੀ ਇਕ ਤੋਂ ਜ਼ਿਆਦਾ ਬੇਸ ਕਲਾਸਾਂ ਹੁੰਦੀ ਹੈ। ਮਲਟੀਪਲ ਇਨਹੈਰੀਟਿਡ ਕਹਿਲਾਉਂਦੀ ਹੈ।



ਇਕ ਕਲਾਸ, ਇਕ ਤੋਂ ਜ਼ਿਆਦਾ ਕਲਾਸਾਂ ਦੇ ਐਟਰੀਬਿਊਟਸ (attributes) ਇਨਹੈਰਿਟ ਕਰ ਸਕਦੀ ਹੈ। ਮਲਟੀਪਲ ਇਨਹੈਰੀਟੈਨਸ ਇਹ ਆਗਿਆ ਦਿੰਦੀ ਹੈ ਕਿ ਇਕ ਨਵੀਂ ਕਲਾਸ ਬਣਾਉਣ ਲਈ ਇਕ ਤੋਂ ਜ਼ਿਆਦਾ ਕਲਾਸਾਂ ਦੇ ਫੀਚਰਜ਼ (features) ਨੂੰ ਜੋੜ ਕੇ (combine), ਸ਼ੁਰੂਆਤੀ ਬਿੰਦੂ (starting point) ਦੀ ਤਰ੍ਹਾਂ ਵਰਤ ਸਕਦੇ ਹਾਂ। ਇਹ ਇਕ ਛੋਟੇ ਬੱਚੇ ਦੀ ਤਰ੍ਹਾਂ ਹੈ ਜੋ ਕੁਝ ਗੁਣ ਆਪਣੀ ਮਾਂ ਤੋਂ ਲੈਂਦਾ ਹੈ ਤੇ ਕੁਝ ਪਿਓ ਤੋਂ।



ਮਿਸਾਲ :

```
# include <iostream.h>
class M
{
    protected :
        int m;
    public :
    void get_on (int) ;
};
Class N
{
    protected
        int n ;
    public :
    void get_n (int);
};
Class P : public M, public N
{
    public :
        void display (void) ;
};
void M :: get_on (int X)
{
    m = x;
}
void N :: get_n (int y)
{
    n = y;
```

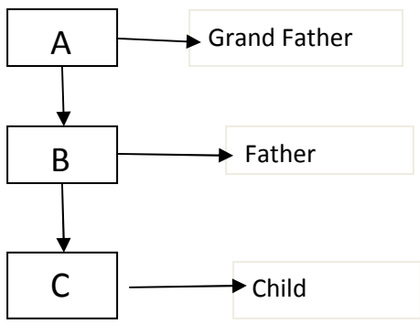
```
}
void P :: display (void)
{
    cout << ‘‘m = ’’ << m << ‘‘\n’’ ;
    cout << ‘‘n = ’’ << n << ‘‘\n’’ ;
    cout << ‘‘m * n=’’ << m * n << ‘‘\n’’ ;
}



int main ( )
{
    P . P ;
    P. get_m (10) ;
    P. get _n (20) ;
    P.display ( ) ;
    return 0 ;
}
```

Output :

m = 10
n = 20
m * n = 200

4.8 ਮਲਟੀਲੈਵਲ ਇਨਹੈਰੀਟੈਂਸ (Multilevel Inheritance)

ਉਹ ਪ੍ਰਕ੍ਰਿਆ ਜਿਸ ਵਿਚ ਇਕ ਕਲਾਸ ਨੂੰ ਡਿਰਾਈਵ ਕਲਾਸ ਤੋਂ ਹੀ ਡਰਾਈਵ ਕੀਤਾ ਜਾਂਦਾ ਹੈ, ਮਲਟੀਲੇਵਲ ਇਨਹੈਰੀਟੈਂਸ ਕਹਾਉਂਦੀ ਹੈ।



ਇਥੇ ਕਲਾਸ A, ਡਿਰਾਈਵ ਕਲਾਸ B ਲਈ ਬੇਸ ਕਲਾਸ ਦਾ ਕੰਮ ਕਰਦੀ ਹੈ ਤੇ ਕਲਾਸ B, ਡਿਰਾਈਵ ਕਲਾਸ C ਲਈ ਬੇਸ ਕਲਾਸ ਦਾ ਕੰਮ ਕਰਦੀ ਹੈ। B ਕਲਾਸ ਨੂੰ ਇੰਟਰਮੀਡੀਏਟ ਬੇਸ ਕਲਾਸ (Intermediate Base class) ਦੇ ਤੌਰ ਤੇ ਜਾਣਿਆ ਜਾਂਦਾ ਹੈ ਕਿਉਂਕਿ ਇਹ A ਤੇ C ਦੇ ਵਿਚ ਇਨਹੈਰੀਟੈਂਸ ਦਾ ਲਿੰਕ ਬਣਾਉਂਦੀ ਹੈ। ਚੇਨ ABC ਨੂੰ ਇਨਹੈਰੀਟੈਨਸ ਪਾਥ ਕਿਹਾ ਜਾਂਦਾ ਹੈ।

```
class A : { ..... };    // Base class
class B : public A {.....} ; //B derived from A
class C : public B {.....} ;  //C derived from B
```

ਮਿਸਾਲ :

```
#include <iostream.h>
class student
{
   protected :
        int roll_number ;
   public :
        void get_number (int) ;
        void put_number (void) ;
};
void student :: get_number (int a)
{
   roll_number = a;
}
void student :: put_number ( )
```

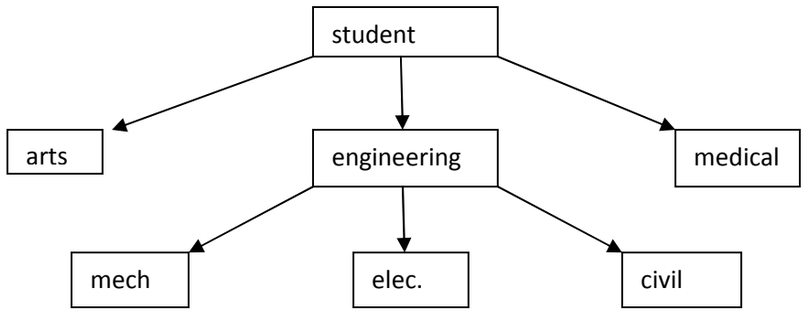
```
{
cout << ‘‘Roll Number : ’’ << roll_number << ‘‘\n’’ ;
}
class test : public student  //first level derivation
{
    protected :
        float sub1 ;
        float sub 2 ;
    public :
    void get_marks (float, float);
    void put_marks (void);
};
void test :: get_marks (float x, float y)
{
    sub1 = x ;
    sub2 = y ;
}
void test :: put_marks ( )
{
    cout << ‘‘Marks in SUB 1 = ’’ << sub1 << ‘‘\n’’ ;
    cout << ‘‘Marks in SUB 2 = ’’ << sub2 << ‘‘\n’’ ;
}
class result : public test  //second level derivation
{
    float total ;   //private by default ;
    public
        void display (void) ;
};
void result :: display (void)
{
    total = sub1 + sub2 ;
    put_number ( ),
    put_marks ( );
    cout << ‘‘Total =’’ << total << ‘‘\n’’ ;
}
int main ( )
{
    result student 1 ;    // student 1 created
    student 1 . get_number (111) ;
    sutdent 2 . get_number (75.0, 59.5) ;
    student 1 . display ( ) ;
    return 0 ;
}
```

Output :

Roll Number : 111
Marks in SUB 1 = 75
Marks in SUB 2 = 59.5
TOTAL = 134.5

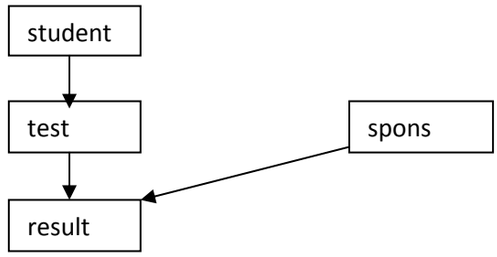
4.9 ਹਿਰਾਰਕੀਕਲ ਇਨਹੈਰੀਟੈਂਸ (Hierarchical Inheritance)

ਜਦੋਂ ਇਕ ਕਲਾਸ ਦੀਆਂ ਵਿਸ਼ੇਸ਼ਤਾਵਾਂ ਇਕ ਤੋਂ ਜ਼ਿਆਦਾ ਕਲਾਸਾਂ ਦੁਆਰਾ ਇਨਹੈਰਿਟ ਕੀਤੀਆਂ ਜਾਂਦੀਆਂ ਹਨ ਤਾਂ ਇਸ ਪ੍ਰਕ੍ਰਿਆ ਨੂੰ ਹਿਰਾਰੀਕਲ ਇਨਹੈਰੀਟੈਂਸ ਕਿਹਾ ਜਾਂਦਾ ਹੈ। ਜਿਵੇਂ ਕਿ ਹੇਠਾਂ ਦਿੱਤੇ ਚਿੱਤਰ ਵਿਚ ਦੱਸਿਆ ਗਿਆ ਹੈ :



4.10 ਹਾਈਬਰਿਡ ਇਨਹੈਰੀਟੈਂਸ (Hybrid Inheritance)

ਜੇਕਰ ਇਸ ਤਰ੍ਹਾਂ ਦੇ ਹਾਲਾਤਾ ਆ ਜਾਣ ਕਿ ਇਕ ਪ੍ਰੋਗਰਾਮ ਨੂੰ ਡਿਜ਼ਾਈਨ ਕਰਨ ਵਾਸਤੇ ਇਕ ਤੋਂ ਜ਼ਿਆਦਾ ਤਰ੍ਹਾਂ ਦੀ ਇਨਹੈਰੀਟੈਂਸ ਦੀ ਵਰਤੋਂ ਕਰਨੀ ਪਵੇ ਤਾਂ ਜੋ ਨਵੀਂ ਇਨਹੈਰੀਟੈਂਸ ਬਣੇਗੀ, ਉਸ ਨੂੰ ਹਾਈਬਰਿਡ ਇਨਹੈਰੀਟੈਂਸ ਕਿਹਾ ਜਾਵੇਗਾ।



4.11 ਬੇਸ ਕਲਾਸ ਤੇ ਡਿਰਾਈਵ ਕਲਾਸ ਵਿਚ ਕਨਵਰਸ਼ਨ (Conversion between Base class and Derived class)

ਜਿੱਥੇ ਤੱਕ C++ ਦੀ ਗੱਲ ਹੈ ਤਾਂ ਇਕ ਬੇਸ ਕਲਾਸ ਦੇ ਇਨਸਟੈਨਸ (instance) ਨੂੰ ਡਿਰਾਈਵ ਕਲਾਸ ਵਿਚ ਬਿਨਾਂ ਕਿਸੀ ਕਨਵਰਜਨ ਅਪਰੇਟਰ (Conversion operator) ਦੇ ਕਨਵਰਟ (Convert) ਨਹੀਂ ਕੀਤਾ ਜਾ ਸਕਦਾ। ਜੇਕਰ ਤੁਹਾਡੇ ਕੋਲ ਡਿਰਾਈਵ ਕਲਾਸ ਦਾ ਇਨਸਟੈਨਸ, ਬੇਸ ਕਲਾਸ ਦੇ ਵੇਰੀਏਬਲ ਦੇ ਤੌਰ ਤੇ ਸਟੋਰ ਹੈ ਤਾਂ ਤੁਸੀਂ ਇਸ ਨੂੰ ਡਿਰਾਈਵ ਕਲਾਸ ਵਿਚ ਕਾਸਟ ਕਰ ਸਕਦੇ ਹੋ।

```
class Base {
    public :
    Base { } { }
    virtual void f ( ) {
    cout << ''f base'' << end l ;
    {
};
class Derived : public Base {
public :
    Derived ( ) { }
    void f ( ) {
    cout << ''f derived'' << end l;
    }
};
int main( )
{
    Base * b = new Derived ( ) ;
    Derived * d = (Derived *) b ;
    Base * b2 = new Base ( );
    Derived * d2 = (Derived *) b2;
    b → f( ) ;      // f derived
    b2 → f ( ) ;   // f base
    d → f ( ) ;    // f derived
    d2 → f ( ) ;  // base
    return 0 ;
}
```

ਯਾਦ ਰੱਖਣ ਯੋਗ ਗੱਲਾਂ (Points to Remember) :

1. ਉਹ ਪ੍ਰਕ੍ਰਿਆ ਜਿਸ ਵਿਚ ਨਵੀਂ ਕਲਾਸ ਨੂੰ ਪੁਰਾਣੀ ਕਲਾਸ ਵਿਚ ਤਿਆਰ ਕੀਤਾ ਜਾਂਦਾ ਹੈ, ਇਨਹੈਰੀਟੈਂਸ ਕਹਿਲਾਉਂਦੀ ਹੈ।
2. ਵਿਜੀਬਿਲਟੀ ਮੋਡ ਜਾਂ ਅਸੈਸ ਸਪੈਸੀਫਾਇਰ ਇਹ ਸਪੱਸ਼ਟ ਕਰਦਾ ਹੈ ਕਿ ਬੇਸ ਕਲਾਸ ਦੇ ਗੁਣ ਪ੍ਰਾਈਵੇਟ ਤੌਰ ਤੇ ਡਿਰਾਈਵਡ ਹੁੰਦੇ ਹਨ ਜਾਂ ਪਬਲਿਕ ਤੌਰ ਤੇ ਡਿਰਾਈਵਰ ਹੁੰਦੇ ਹਨ।

3. ਓਵਰਰਾਈਡਿੰਗ ਵਿਚ ਇਕ ਫੰਕਸ਼ਨ ਜੋ ਕਿ ਬੇਸ ਕਲਾਸ ਵਿਚ ਪਰਿਭਾਸ਼ਿਤ ਹੈ ਉਸਦੀ ਨਵੀਂ ਪਰਿਭਾਸ਼ਾ ਡਿਰਾਈਵਡ ਕਲਾਸ ਵਿਚ ਲਿਖੀ ਜਾਂਦੀ ਹੈ।
4. ਇਨਹੈਰੀਟੈਂਸ ਕਈ ਪ੍ਰਕਾਰ ਦੀ ਹੁੰਦੀ ਹੈ ਜਿਵੇਂ ਕਿ ਸਿੰਗਲ, ਮਲਟੀਪਲ, ਮਲਟੀਲੇਵਲ, ਹਿਰਾਰਕੀਕਲ ਅਤੇ ਹਾਈਬਰਿਡ ਇਨਹੈਰੀਟੈਂਸ।

ਅਭਿਆਸ
(Exericse)

1. ਖਾਲੀ ਥਾਵਾਂ ਭਰੋ—

1. ਕਲਾਸ ਨੂੰ ਲਿਖਣ ਉਪਰੰਤ ਉਸ ਨੂੰ ..... ਕੀਤਾ ਜਾ ਸਕਦਾ ਹੈ।
2. ਪੁਰਾਣੀ ਕਲਾਸ ਨੂੰ ..... ਕਿਹਾ ਜਾਂਦਾ ਹੈ ਤੇ ਨਵੀਂ ਕਲਾਸ ਨੂੰ ..... ਕਿਹਾ ਜਾਂਦਾ ਹੈ।
3. ਇਕ ਡਿਰਾਈਵ ਕਲਾਸ ਜਿਸ ਦੀ ਇਕੋ ਬੇਸ ਕਲਾਸ ਹੋਵੇ ਨੂੰ ..... ਕਿਹਾ ਜਾਂਦਾ ਹੈ।
4. ..... ਆਫ ਫੰਕਸ਼ਨ ਉਸ ਸਮੇਂ ਹੁੰਦੀ ਹੈ ਜਦੋਂ ਇਕ ਕਲਾਸ ਨੂੰ ਦੂਜੀ ਕਲਾਸ ਤੋਂ ਇਨਹੈਰਿਟ ਕੀਤਾ ਜਾਂਦਾ ਹੈ।
5. ..... ਕੋਡ ਦੀ ਮੁੜ ਵਰਤੋਂ ਦਾ ਬਲੈਕ ਬਾਕਸ ਕਹਿਲਾਉਂਦਾ ਹੈ।

2. ਪ੍ਰਸ਼ਨਾਂ ਦੇ ਉੱਤਰ ਦਿਉ—

1. ਇਨਹੈਰੀਟੈਂਸ ਦੀ ਪ੍ਰਕ੍ਰਿਆ ਦਾ ਵਰਣਨ ਕਰੋ।
2. ਇਨਹੈਰੀਟੈਂਸ ਦੀਆਂ ਕਿਸਮਾਂ ਦੱਸੋ।
3. ਅਸੈਸ ਸਪੈਸੀਫਾਇਰ ਤੇ ਇਕ ਨੋਟ ਲਿਖੋ।
4. ਫੰਕਸ਼ਨ ਓਵਰਰਾਈਡਿੰਗ ਦਾ ਕੀ ਅਰਥ ਹੈ ?
5. ਮਲਟੀਪਲ ਤੇ ਮਲਟੀਲੇਵਲ ਇਨਹੈਰੀਟੈਂਸ ਵਿਚ ਅੰਤਰ ਦੱਸੋ।
6. ਸਿੰਗਲ ਇਨਹੈਰੀਟੈਂਸ ਦੀ ਉਦਾਹਰਣ ਸਹਿਤ ਵਿਆਖਿਆ ਕਰੋ।

LAB ACTIVITY

1.
```
#include <iostream.h>
using namespace std;

class A {
int data;
public:
void f(intarg) { data = arg; }
int g() { return data; }
};
class B : public A { };

int main() {
   B obj;
obj.f(20);
cout<<obj.g() <<endl;
}
```

2.
```
#include<iostream.h>
usingnamespacestd;
class A
{
public:
void display()
{
cout<<"Base class content.";
}
};

class B :public A
{

};

class C :public B
{

};

int main()
{
    C c;
```

```
c.display();
return0;
}
```

3.

```
#include <iostream.h>

using namespace std;

// Base class
class Shape
{
public:
voidsetWidth(int w)
      {
width = w;
      }
voidsetHeight(int h)
      {
height = h;
      }
protected:
int width;
int height;
};

// Derived class
class Rectangle: public Shape
{
public:
intgetArea()
      {
return (width * height);
      }
};
int main(void)
{
   Rectangle Rect;

Rect.setWidth(5);
Rect.setHeight(7);

   // Print the area of the object.
cout<< "Total area: " <<Rect.getArea() <<endl;

return 0;
}
```

When the above code is compiled and executed, it produces following result:

```
Total area: 35
```

4.

```
#include<iostream.h>

usingnamespacestd;

// Base class Shape
classShape
{
public:
voidsetWidth(int w)
{
width= w;
}
voidsetHeight(int h)
{
height= h;
}
protected:
int width;
int height;
};
```

```
// Base class PaintCost
classPaintCost
{
public:
intgetCost(int area)
{
return area *70;
}
};

// Derived class
classRectangle:publicShape,publicPaintCost
{
public:
intgetArea()
{
return(width * height);
}
};

int main(void)
{
RectangleRect;
int area;

Rect.setWidth(5);
Rect.setHeight(7);

area=Rect.getArea();

// Print the area of the object.
cout<<"Total area: "<<Rect.getArea()<<endl;

// Print the total cost of painting
cout<<"Total paint cost: $"<<Rect.getCost(area)<<endl;

return0;
}
```

When the above code is compiled and executed, it produces following result:

```
Total area:35
Total paint cost: $2450
```

5.

```
        #include<iosteram.h>
        #include<conio.h>

        Class M
{
                Protected:
                Int m;
                Public :
                Void get_M();
        };
        Class N
{
                Protected:
                Int n;
                Public:
                Void get_N();
        };
        Class p:public M, public N
        {
                Public:
                Void disply(void);
        };
        Void M ::get_m(int x)
        {
                m=x;
        }
        Void N::get_n(int y)
        {
                n=y;
        }
```

```
Void P::disply(void)
{
Cout<<”m=”<<m<<endl;
Cout<<”n=”<<n<<endl;
Cout<<”m*n=”<<m*n<<endl;
}
int main()
{
        P p;
        p.get_m(10);
        p.get_n(20);
        p.display();
        return0;
}
```

OUTPUT

m=10
n=20
m*n=200

6.

```
#include<iostream.h>
#include<conio.h>
class B
{
int a;
public:
int b;
voidget_ab();
intget_a();
voidshow_a();
};
class D:private B
{
int c;
public:
voidmul();
void display();
};
void B::get_ab()
{
cout<<"Enter Values for a and b";
cin>>a>>b;
}
int B::get_a()
{
return a;
}
void B::show_a(){
                        cout<<"a= "<<a<<“\n”;
                }
void D::mul()
{
                        get_ab();
                        c=b*get_a();
        }
void D::display()
{
show_a();
cout<<"b= "<<b<<“\n”;
cout<<"c= "<<c<<"\n\n";
}
void main()
{
                        clrscr();
                        D d;
                        d.mul();
                        d.display();
                        d.mul();
                        d.display();
                        getch();
                }
```

OUTPUT

A=5
A=5
B=10
C=50

A=5
B=20
C=100

7.

```
#include<iostream.h>
#include<conio.h>

class student
{




        public:
        intrno;
        //float per;
        char name[20];
        voidgetdata()
        {
                cout<<"Enter RollNo :- \t";
                cin>>rno;
                cout<<"Enter Name :- \t";
                cin>>name;

        }

};
class marks : public student
{
public:
        int m1,m2,m3,tot;
        float per;
        voidgetmarks()
        {
                getdata();
                cout<<"Enter Marks 1 :- \t";
                cin>>m1;
                cout<<"Enter Marks 2 :- \t";
                cin>>m2;
                cout<<"Enter Marks 2 :- \t";
                cin>>m3;
        }
        void display()
        {
                getmarks();
                cout<<"Roll Not \t Name \t Marks1 \t marks2 \t Marks3 \t Total \t
Percentage";

        cout<<rno<<"\t"<<name<<"\t"<<m1<<"\t"<<m2<<"\t"<<m3<<"\t"<<tot<<"\t"<<per;
        }
};
void main()
{
        studentstd;
        clrscr();
        std.getmarks();
        std.display();
        getch();
}
```

8.

```
#include <iostream.h>
using namespace std;
class A {
int data;
public:
void f(intarg) { data = arg; }
int g() { return data; }
};

class B {
public:
A x;
};

int main() {
   B obj;
obj.x.f(20);
cout<<obj.x.g() <<endl;
//   cout<<obj.g() <<endl;
}
```

9.

```
#include<iostream.h>
usingnamespacestd;
classArea
{
public:
floatarea_calc(floatl,float b)
{
return l*b;
}
};

classPerimeter
{
public:
floatperi_calc(floatl,float b)
{
return2*(l+b);
}
};

/* Rectangle class is derived from classes Area and Perimeter. */
classRectangle:privateArea,privatePerimeter
{
private:
float length, breadth;
public:
Rectangle(): length(0.0), breadth(0.0){}
voidget_data()
{
cout<<"Enter length: ";
cin>>length;
cout<<"Enter breadth: ";
cin>>breadth;
}

floatarea_calc()
{
/* Calls area_calc() of class Area and returns it. */

returnArea::area_calc(length,breadth);
}

floatperi_calc()
{
/* Calls peri_calc() function of class Perimeter and returns it. */

returnPerimeter::peri_calc(length,breadth);
}
};

int main()
{
Rectangle r;
r.get_data();
```

```
cout<<"Area = "<<r.area_calc();
cout<<"\nPerimeter = "<<r.peri_calc();
return0;
}
```

10.

PROGRAM: PAYROLL SYSTEM USING SINGLE INHERITANCE

```
#include<iostream.h>
#include<conio.h>

class emp
{
   public:
     int eno;
     char name[20],des[20];
     void get()
     {
               cout<<"Enter the employee number:";
               cin>>eno;
               cout<<"Enter the employee name:";
               cin>>name;
               cout<<"Enter the designation:";
               cin>>des;
     }
};

class salary:public emp
{
     float bp,hra,da,pf,np;
   public:
     void get1()
     {
               cout<<"Enter the basic pay:";
               cin>>bp;
               cout<<"Enter the Humen Resource Allowance:";
               cin>>hra;
               cout<<"Enter the Dearness Allowance :";
               cin>>da;
               cout<<"Enter the Profitablity Fund:";
               cin>>pf;
     }
     void calculate()
     {
               np=bp+hra+da-pf;
     }
     void display()
     {

cout<<eno<<"\t"<<name<<"\t"<<des<<"\t"<<bp<<"\t"<<hra<<"\t"<<da<<"\t"<<pf<<"\t"<<np<
<"\n";
     }
};

void main()
{
    int i,n;
    char ch;
    salary s[10];
    clrscr();
    cout<<"Enter the number of employee:";
    cin>>n;
    for(i=0;i<n;i++)
    {
               s[i].get();
               s[i].get1();
               s[i].calculate();
    }
    cout<<"\ne_no \t e_name\t des \t bp \t hra \t da \t pf \t np \n";
    for(i=0;i<n;i++)
    {
               s[i].display();
    }
    getch();
}
```

```
Output:
Enter the Number of employee:1
Enter the employee No: 150
Enter the employee Name: ram
Enter the designation: Manager
Enter the basic pay: 5000
Enter the HR allowance: 1000
Enter the Dearness allowance: 500
Enter the profitability Fund: 300

E.No   E.name   des      BP    HRA   DA   PF     NP
150    ram      Manager  5000  1000  500  300    62
```

Exercise

Show the output of the following programs:

Program 1:

```
// friend functions
#include <iostream.h>
usingnamespacestd;

class CRectangle {
int width, height;
public:
voidset_values (int, int);
int area () {return (width * height);}
friendCRectangle duplicate (CRectangle);
};

void CRectangle::set_values (int a, int b) {
width = a;
height = b;
}

CRectangle duplicate (CRectanglerectparam)
{
CRectanglerectres;
rectres.width = rectparam.width*2;
rectres.height = rectparam.height*2;
return (rectres);
}

int main () {
CRectanglerect, rectb;
rect.set_values (2,3);
rectb = duplicate (rect);
cout<<rectb.area();
return 0;
}
```

Program 2:

```
// friend class
#include <iostream.h>
using namespacestd;

class CSquare;
```

```
class CRectangle {
int width, height;
public:
int area ()
      {return (width * height);}
void convert (CSquare a);
};

class CSquare {
private:




int side;
public:
voidset_side (int a)
      {side=a;}
friend class CRectangle;
};

void CRectangle::convert (CSquare a) {
width = a.side;
height = a.side;
}

int main () {
CSquaresqr;
CRectanglerect;
sqr.set_side(4);
rect.convert(sqr);
cout<<rect.area();
return 0;
}
```

Program 3:

```
// derived classes
#include <iostream.h>
usingnamespacestd;

classCPolygon {
protected:
int width, height;
public:
voidset_values (int a, int b)
{ width=a; height=b;}
  };

classCRectangle: publicCPolygon {
public:
int area ()
{ return (width * height); }
  };

classCTriangle: publicCPolygon {
public:
int area ()
{ return (width * height / 2); }
  };

int main () {
CRectanglerect;
CTriangletrgl;
rect.set_values (4,5);
trgl.set_values (4,5);
cout<<rect.area() <<endl;
cout<<trgl.area() <<endl;
return 0;
}
```

Program 4:

```
// constructors and derived classes
#include <iostream.h>
usingnamespacestd;

class mother {
```

```
public:
mother ()
{ cout<< "mother: no parameters\n"; }
mother (int a)
{ cout<< "mother: int parameter\n"; }
};



class daughter : public mother {
public:
daughter (int a)
{ cout<< "daughter: int parameter\n\n"; }
};

class son : public mother {
public:
son (int a) : mother (a)
{ cout<< "son: int parameter\n\n"; }
};

int main () {
daughtercynthia (0);
sondaniel(0);

return 0;
}
```

Program 5:

```
// multiple inheritance
#include <iostream.h>
usingnamespacestd;

classCPolygon {
protected:
int width, height;
public:
voidset_values (int a, int b)
{ width=a; height=b;}
  };

classCOutput {
public:
void output (int i);
  };

voidCOutput::output (int i) {
cout<< i <<endl;
  }

classCRectangle: publicCPolygon, publicCOutput {
public:
int area ()
{ return (width * height); }
  };

classCTriangle: publicCPolygon, publicCOutput {
public:
int area ()
{ return (width * height / 2); }
  };
int main () {
CRectanglerect;
CTriangletrgl;
rect.set_values (4,5);
trgl.set_values (4,5);
rect.output (rect.area());
trgl.output (trgl.area());
return 0;
}
```

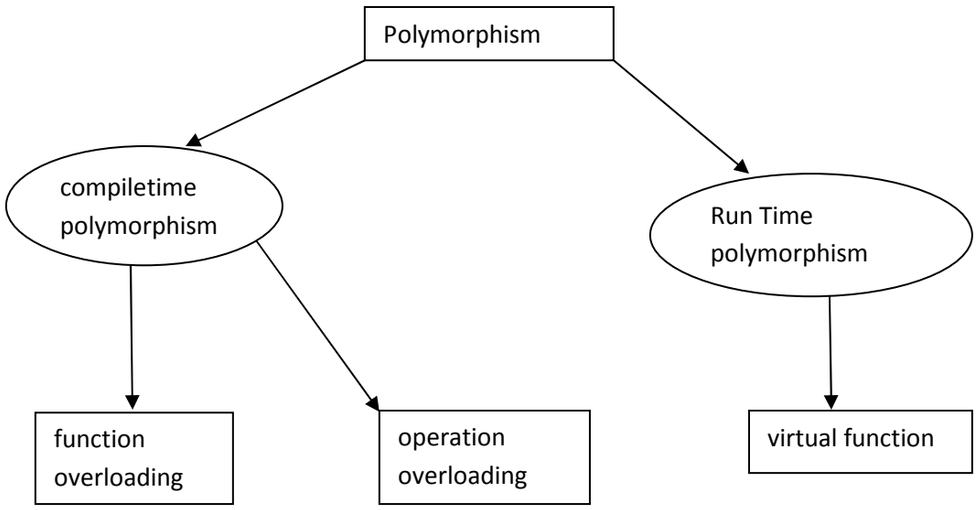
ਪਾਠ 5

ਪੋਲੀਮਾਰਫਿਜ਼ਮ
(Polymorphism)

ਇਸ ਪਾਠਕ੍ਰਮ ਵਿਚ ਅਸੀਂ ਪੋਲੀਮਾਰਫਿਜ਼ਮ (Polymorphism) ਵਰਚੂਅਲ ਫੰਕਸ਼ਨ, ਸਟੈਟਿਕ ਅਤੇ ਡਾਇਨਾਮਿਕ ਬਾਈਂਡਿੰਗ ਅਬਸਟਰੈਕਟ ਬੇਸ ਕਲਾਸ ਅਤੇ ਵਰਚੂਅਲ ਡਿਸਟਰਕਟਰ (virtual Destructor), ਵਰਚੂਲ ਟੇਬਲ ਬਾਰੇ ਜਾਣਾਂਗੇ।

5.1 ਪੋਲੀਮਾਰਫਿਜ਼ਮ (Polymorphism)

ਪੋਲੀਮਾਰਫਿਜ਼ਮ ਦਾ ਅਰਥ ਹੈ ਇਕ ਨਾਂ, ਕਈ ਕਿਸਮਾਂ (One Name, Multiple Forms)। ਇਹ ਸਾਨੂੰ ਇਕ ਨਾਮ ਦੇ ਇਕ ਤੋਂ ਵੱਧ ਫੰਕਸ਼ਨ ਇਕ ਪ੍ਰੋਗਰਾਮ (Function Over loading) ਵਿਚ ਬਣਾਉਣ ਦੀ ਆਗਿਆ ਦਿੰਦਾ ਹੈ।
ਇਹ ਸਾਨੂੰ ਅਪਰੇਟਰ ਓਵਰਲੋਡਿੰਗ (Overloading) ਦੀ ਵੀ ਆਗਿਆ ਦਿੰਦਾ ਹੈ, ਜਿਸ ਦਾ ਅਰਥ ਹੈ ਅਲੱਗ-ਅਲੱਗ ਹਲਾਤਾਂ (instances) ਵਿਚ ਇਕ ਅਪਰੇਟਰ ਦੁਆਰਾ ਅਲੱਗ-ਅਲੱਗ ਵਿਵਹਾਰ ਦਿਖਾਣਾ।
ਪੋਲੀਮਾਰਫਿਜ਼ਮ ਦੋ ਤਰ੍ਹਾਂ ਦਾ ਹੁੰਦਾ ਹੈ——
* ਕੰਪਾਈਲ ਟਾਈਮ ਪੋਲੀਮਾਰਫਿਜ਼ਮ (Compile Time Polymorphism)
* ਰਨ-ਟਾਈਮ ਪੋਲੀਮਾਰਫਿਜ਼ਮ (Run Time Polymorphism)



ਚਿੱਤਰ 5.1 Polymorphism

5.1.1 ਸਟੈਟਿਕ ਬਾਈਂਡਿੰਗ

ਇਸ ਵਿਚ ਕਿਸੇ ਵੀ ਫੰਕਸ਼ਨ (Procedure) ਨਾਲ ਸੰਬੰਧਿਤ ਕੋਡ ਕੰਪਾਇਲਰ ਨੂੰ ਕੰਪਾਈਲ ਟਾਈਮ ਤੇ ਪਤਾ ਚੱਲਦਾ ਹੈ।

5.1.2 ਡਾਇਨਾਮਿਕ ਬਾਈਂਡਿੰਗ

ਇਸ ਵਿਚ ਕਿਸੇ ਵੀ ਫੰਕਸ਼ਨ ਦੇ ਨਾਲ ਸੰਬੰਧਿਤ ਰੇਡ ਕੰਪਾਇਲਰ ਨੂੰ ਰਨ ਟਾਈਮ ਤੇ ਪਤਾ ਚੱਲਦਾ ਹੈ।

5.2 ਵਰਚੂਅਲ ਫੰਕਸ਼ਨ (virtual function)

ਜਿਵੇਂ ਕਿ ਪਹਿਲਾਂ ਹੀ ਦਰਸਾਇਆ ਗਿਆ ਹੈ ਕਿ ਪੋਲੀਮਾਰਫਿਜ਼ਮ ਦੀ ਇਕ ਵਿਸ਼ੇਸ਼ਤਾ ਇਹ ਵੀ ਹੈ ਕਿ ਇਸ ਵਿੱਚ ਅਲੱਗ-ਅਲੱਗ ਕਲਾਸਾਂ ਤੋਂ ਸੰਬੰਧਿਤ ਆਬਜੈਕਟ (Object) ਇਕ ਹੀ ਮੈਸੇਜ (Message) ਦਾ ਜਵਾਬ ਅਲੱਗ-ਅਲੱਗ ਤਰੀਕਿਆਂ ਨਾਲ ਦੇ ਸਕਦੇ ਹਨ।
ਇਸ ਲਈ ਇਕ ਅਜਿਹੇ ਸਿੰਗਲ ਪੁਆਇੰਟਰ ਵੇਰੀਏਬਲ (Single Pointer Variable) ਦੀ ਜ਼ਰੂਰਤ ਮਹਿਸੂਸ ਹੁੰਦੀ ਹੈ ਜੋ ਅਲੱਗ-ਅਲੱਗ ਕਲਾਸਾਂ ਦੇ ਆਬਜੈਕਟ (Objects) ਨੂੰ ਇਕੱਠਾ ਰੈਫਰ (refer) ਕਰ ਸਕੇ।
ਇੱਥੇ ਅਸੀਂ ਬੇਸ ਕਲਾਸ (Base class) ਦੇ ਪੁਆਇੰਟਰ ਨੂੰ ਡਿਰਾਈਵਡ ਆਬਜੈਕਟ (derived objects) ਨੂੰ ਰੈਫਰ ਕਰਨ ਲਈ ਇਸਤੇਮਾਲ ਕਰਦੇ ਹਾਂ। ਜਦੋਂ ਅਸੀਂ ਇਕੋ ਫੰਕਸ਼ਨ name ਬੇਸ ਅਤੇ ਡਿਰਾਈਵਡ ਕਲਾਸ ਦੋਹਾਂ ਵਿਚ ਵਰਤਦੇ ਹਾਂ ਤਾਂ ਬੇਸ ਕਲਾਸ ਵਿਚ ਘੋਸ਼ਿਤ ਕੀਤੇ ਗਏ ਫੰਕਸ਼ਨ ਦੇ ਅੱਗੇ ਵਰਚੂਅਲ (virtual) ਸ਼ਬਦ ਇਸਤੇਮਾਲ ਕਰਦੇ ਹਾਂ। ਜਦੋਂ ਇਕ ਫੰਕਸ਼ਨ ਨੂੰ ਵਰਚੂਅਲ ਬਣਾਇਆ ਜਾਂਦਾ ਹੈ ਤਾਂ C++ ਦੱਸਦੀ ਹੈ ਕਿ ਪੁਆਇੰਟਰ ਦੀ ਟਾਈਪ ਵੱਲ ਧਿਆਨ ਦੇਣ ਦੀ ਥਾਂ ਉਸ ਫੰਕਸ਼ਨ ਨੂੰ ਰਨ ਟਾਈਮ ਤੇ ਵਰਤੋ ਜਿਸ ਨੂੰ ਬੇਸ ਪੁਆਇੰਟਰ (Base Pointer) ਫੰਕਸ਼ਨ ਦੇ ਆਬਜੈਕਟ ਨਾਲ ਪੁਆਇੰਟ ਕਰਦਾ ਹੈ।
ਇਸ ਤਰ੍ਹਾਂ ਅਸੀਂ ਵਰਚੂਅਲ ਫੰਕਸ਼ਨ ਦੇ ਅਲੱਗ-ਅਲੱਗ ਵਰਜਨ (version) ਰਨ (run) ਕਰਵਾ ਸਕਦੇ ਹਾਂ।

ਪ੍ਰੋਗਰਾਮ 5.2

VIRTUAL FUNCTIONS

```
# include <iostream.h>
class Base
{

    public :
    void display( ) {cout << ‘‘\n Display base’’ : }
    virtual void show ( ) {cout << ‘‘\n showbase’’ ;}
};
class Derived : public Base
{
    public :
    void display ( ) {cout << ‘‘\n Display derived’’ ;}
    void show ( ) {cout << ‘‘\n Show derived’’ ;}
};

    int main( )
{
    Base B;
    Derived D;
    Base * btpr ;
cout << ‘‘\n btpr Points to Base \n’’;
btpr = & B ;
btrp -> display ( ) ;   //Calls Base Version
btpr -> show ( ) ;     //Calls Base Version
cout << ‘‘\n \n btpr Points to Derived \n’’;
btpr = & 0;
btpr -> display ( );   //Calls Base Version
btpr -> show ( ) ;     //Calls Derived Version
return 0;
}
```

The output of Program would be

```
bptr Points to Base
Display base
Show base
btpr Points to Derived
Display base
Show derived
```

5.2.1 ਪਿਓਰ ਵਰਚੁਅਲ ਫੰਕਸ਼ਨ (Pure virtual Function)

ਇਕ Pure virtual Function ਉਹ ਫੰਕਸ਼ਨ ਹੈ ਜੋ ਘੋਸ਼ਿਤ ਤਾਂ base class ਵਿਚ ਕੀਤਾ ਜਾਂਦਾ ਹੈ ਪਰ base classs ਨਾਲ ਸੰਬੰਧਿਤ ਉਸ ਦੀ ਕੋਈ ਪਰਿਭਾਸ਼ਾ ਨਹੀਂ ਹੁੰਦੀ। ਇਕ ਵਰਚੁਅਲ ਫੰਕਸ਼ਨ ਜੋ ਜ਼ੀਰੋ ਦੇ ਬਰਾਬਰ ਹੁੰਦਾ ਹੈ, ਉਹ ਪਿਓਰ ਵਰਚੁਅਲ ਫੰਕਸ਼ਨ ਕਹਾਉਂਦਾ ਹੈ।

5.3 ਅਬਸਟਰੈਕਟ ਬੇਸ ਕਲਾਸ (Abstract Base classes)

ਇਕ ਕਲਾਸ ਜਿਸ ਵਿਚ Pure virtual Function ਹੁੰਦੇ ਹਨ, ਉਹ ਅਬਸਟੈਕਟ ਬੇਸ ਕਲਾਸ ਕਹਾਉਂਦੀ ਹੈ। ਇੱਥੇ ਇਹ ਯਾਦ ਰੱਖਣ ਯੋਗ ਹੈ ਕਿ ਜਿਸ ਕਲਾਸ ਵਿਚ Pure virtual function ਹੁੰਦੇ ਹਨ, ਉਸ class ਦਾ ਇਸਤੇਮਾਲ ਕਿਸੇ ਵੀ ਆਬਜੈਕਟ ਨੂੰ ਘੋਸ਼ਿਤ ਕਰਨ ਲਈ ਨਹੀਂ ਕੀਤਾ ਜਾ ਸਕਦਾ।

5.3.1 ਇੰਟਰਫੇਸਿਸ ਇਨ C++ (Interfaces in C++)

ਇਕ ਇੰਟਰਫੇਸ C++ ਕਲਾਸ ਨੂੰ ਲਾਗੂ ਕੀਤੇ ਜਾਣ ਜਾਂ ਨਾਂ ਲਾਗੂ ਕੀਤੇ ਜਾਣ ਬਾਰੇ ਜਾਣੇ ਬਿਨਾਂ ਕਲਾਸ ਦੀ ਸਮਰੱਥਾ ਜਾਂ ਉਸ ਦੇ ਵਿਵਹਾਰ ਨੂੰ ਪਰਿਭਾਸ਼ਿਤ ਕਰਦਾ ਹੈ।

C++ ਇੰਟਰਫੇਸ ਅਬਸਟੈਕਟ (abstract class) ਕਲਾਸ ਨੂੰ ਪ੍ਰਯੋਗ (using) ਕਰਕੇ ਲਾਗੂ ਕੀਤੇ ਜਾਂਦੇ ਹਨ। ਇਕ ਕਲਾਸ ਦੇ ਕਿਸੇ ਵੀ ਇਕ ਫੰਕਸ਼ਨ ਨੂੰ Pure virtual Function ਡਿਕਲੇਅਰ ਕਰਕੇ ਕਲਾਸ ਨੂੰ ਅਬਸਟਰੈਕਟ ਬਣਾਇਆ ਜਾ ਸਕਦਾ ਹੈ।

Pure virtual Function ਦਰਸਾਉਣਾ ''= 0''

ਅਤੇ ਇਸ ਦੀ ਡਿਕਲੇਅਰੇਸ਼ਨ ਇਸ ਤਰ੍ਹਾਂ ਹੈ।

Declaration

```
class Box
{
 public :
    // Pure virtual function
    virtual double get volume ( ) = 0;
    private :
    double length;   //length of a box
    double breadth;  //breadth of a box
    double height;   //height of a box
};
```

ਅਬਸਟਰੈਕਟ ਕਲਾਸ ਉਦਾਹਰਣ——

ਉਦਾਹਰਣ 5.3

Abstract class Example

Consider the following example where parent class provides an interface to the base implement a function called getArea ( ) :

```
# include <iostream.h>
using namespace std;
// Base class
class Shape
{
public :
     // pure virtual function providing interface framework,
     virtual int getArea ( ) = 0 ;
     void setWidth (int w)
     {
          Width = w;
     }
     void setHeight (int h)
     {
          height = h'
     }
protected :
     int width ;
     int height ;
};
//Derived classes
class Rectangle : public Shape
{
public :
     int getArea ( )
     {
          return (width * height) ;
     }
};
class Triangle : public Shape
{
public :
     int getArea (*)
     {
          return (width * height)/2;
```

```
    }
};
int main (void)
{
    Rectangle Rect;
    Triangle Tri;
    Rect.setWidth (3) ;
    Rect.setHeight (7) ;
    // Print the area of the object.
    cout << ‘‘Total Rectangle area : ’’ << Rect.get Area ( ) << end1;
    Tri.setWidth (5) ;
    Tri.setHeight (7) ;
    // Print the area of the object.
    cout << ‘‘Total Triangle area : ‘‘Tri.getArea ( ) << endl;
    return 0;
}
```

When the above code is compiled and executed, it produces following result :
Total Rectangle area : 35
Total Triangle area : 17

5.4 ਵਰਚੂਅਲ ਟੇਬਲ (virtual Table)

ਵਰਚੂਅਲ ਟੇਬਲ ਨੂੰ ਅਸੀਂ ਕਈ ਵਾਰੀ Vtable ਦੇ ਨਾਮ ਨਾਲ ਵੀ ਜਾਣਦੇ ਹਾਂ। ਅਸਲ ਵਿਚ ਵਰਚੂਅਲ ਟੇਬਲ ਬਹੁਤ ਸਧਾਰਨ ਹੈ ਲੇਕਿਨ ਇਸ ਨੂੰ ਸ਼ਬਦਾਂ ਵਿੱਚ ਬਿਆਨ ਕਰਨਾ ਥੋੜ੍ਹਾ ਜਿਹਾ ਔਖਾ ਹੈ।

ਸਭ ਤੋਂ ਪਹਿਲਾਂ ਹਰੇਕ ਕਲਾਸ (class) ਜੋ ਵਰਚੂਅਲ ਫੰਕਸ਼ਨ ਵਰਤ ਰਹੀ ਹੈ, ਨੂੰ ਇਸ ਦਾ ਆਪਣਾ ਵਰਚੂਅਲ ਟੇਬਲ ਦਿੱਤਾ ਜਾਂਦਾ ਹੈ। ਇਹ Table ਸਧਾਰਨ ਤੌਰ ਤੇ ਇਕ ਸਟੈਟਿਕ ਐਰੇ ਹੁੰਦਾ ਹੈ ਜਿਸ ਨੂੰ ਕੰਪਾਇਲਰ ਕੰਪਾਇਲ ਟਾਈਮ ਤੇ ਬਣਾਉਂਦਾ ਹੈ। ਜਦੋਂ ਕਲਾਸ (class) ਦੇ ਆਬਜੈਕਟ ਦੁਆਰਾ ਵਰਚੂਅਲ ਫੰਕਸ਼ਨ ਨੂੰ call ਕੀਤਾ ਜਾਂਦਾ ਹੈ ਤਾਂ ਵਰਚੂਅਲ ਟੇਬਲ ਵਿਚ ਉਸ ਦੀ ਇਕ ਐਂਟਰੀ (Entry) ਲੈਂਦਾ ਹੈ। ਇਸ ਟੇਬਲ ਦੀ ਹਰੇਕ ਐਂਟਰੀ ਇਕ ਫੰਕਸ਼ਨ ਪੁਆਇੰਟਰ ਹੁੰਦੀ ਹੈ ਜੋ ਕਿ ਕਲਾਸ ਦੇ ਡਿਰਾਈਵਡ ਫੰਕਸ਼ਨ ਦੇ ਵੱਲ ਇਸ਼ਾਰਾ ਕਰਦੀ ਹੈ।

ਉਦਾਹਰਣ 5.4

```
 1
 2 class Base
 3 {
 4 public :
 5 virtual void : function1 ( ) { };
 6 virtual void function2 ( ) { };
 7 };
 8 class D1 : public Base
 9 {
10 public :
11 virtual void function ( ) { } ;
12 };
13 class D2 : public Base
14 {
15 public :
16 virtual void function2 ( ) { } ;
17 };
18
```

ਇਸ ਉਦਾਹਰਣ ਵਿਚ 3 ਕਲਾਸਾਂ ਹਨ। ਇਸ ਕਰਕੇ ਕੰਪਾਇਲਰ 3 ਵਰਚੂਅਲ ਟੇਬਲ ਬਣਾਏਗਾ। ਇਕ Base ਵਾਸਤੇ, ਇਕ D1 ਲਈ, ਇਕ D2 ਲਈ।

5.5 ਵਰਚੂਅਲ ਡਿਸਟਰਕਟਰ (virtual Destructor)

ਜਿਵੇਂ ਕਿ ਅਸੀਂ ਜਾਣਦੇ ਹਾਂ ਕਿ C++ ਵਿਚ ਡਿਸਟਰਕਟਰ ਮੈਮਰੀ ਨੂੰ ਖਾਲੀ ਕਰਵਾਉਣ ਲਈ ਵਰਤੇ ਜਾਂਦੇ ਹਨ, ਡਿਸਟਰਕਟਰ ਨੂੰ ਅਸੀਂ Tiled ~ ਚਿੰਨ੍ਹ ਨਾਲ ਦਰਸਾਉਂਦੇ ਹਾਂ। ਵਰਚੂਅਲ ਫੰਕਸ਼ਨ ਪਰਿਭਾਸ਼ਿਤ ਕਰਦੇ ਸਮੇਂ ਅਸੀਂ ਸਿਰਫ ਵਰਚੂਅਲ Tiled ~ ਚਿੰਨ੍ਹ ਦੇ ਅੱਗੇ ਲਗਾਉਂਦੇ ਹਾਂ।

C++ ਵਿਚ ਵਰਚੂਅਲ ਡਿਸਟਰਕਟਰ ਦੀ ਲੋੜ ਹੇਠਾਂ ਲਿਖੀ ਉਦਾਹਰਣ ਦੁਆਰਾ ਸਮਝੀ ਜਾ ਸਕਦੀ ਹੈ।

ਉਦਾਹਰਣ 5.5.1 ਵਰਚੂਅਲ ਡਿਸਟਰਕਟਰ ਤੋਂ ਬਿਨਾਂ

```
# include <iostrean.h>
class Base
{
    public :
    Base ( ) {cout << ‘‘Constructing Base’’;}
    //this is a destructor;
    ~ Base ( ) {cout << ‘‘Destroying Base’’; }
};
class Derive : public Base
{
    public :
    Derive ( ) {cout << ‘‘Constructing Derive’’;}
    ~ Derive ( ) {cout <<‘‘Destroying Derive’’;}
};
void main ( )
{
    Bass * baseptr = new Derive ( ) ;
    delete base Ptr ;
}
```

Output :

```
Constructing Base
Constructing Derive
Destroying Base
```

ਇਸ ਉਦਾਹਰਣ ਤਹਿਤ ਅਸੀਂ ਦੇਖ ਸਕਦੇ ਹਾਂ ਕਿ ਕੰਸਟਰਕਟਰ ਨੂੰ ਸਹੀ ਆਰਡਰ ਵਿਚ ਬੁਲਾਏ ਜਾਂਦੇ ਹਨ ਜਦੋਂ ਡਿਰਾਈਵ ਕਲਾਸ ਆਬਜੈਕਟ ਪੁਆਇੰਟਰ ਮੇਨ ਫੰਕਸ਼ਨ ਵਿਚ ਕਰੀਏਟ ਕੀਤੇ ਜਾਂਦੇ ਹਨ।

ਪਰ ਇੱਥੇ ਕੋਡ ਨਾਲ ਇਕ ਵੱਡੀ ਸਮੱਸਿਆ ਹੈ ਕਿ ਜਦੋਂ ਅਸੀਂ ‘base ptr’ ਡਿਲੀਟ ਕਰ ਦਿੰਦੇ ਹਾਂ ਤਾਂ "Derive" class ਲਈ ਜਿਹੜੀ ਡਿਸਟਰਕਟਰ ਹੈ Call ਨਹੀਂ ਕੀਤੀ ਜਾਂਦੀ।

ਉਦਾਹਰਣ 5.5.2 (ਵਰਚੂਅਲ ਡਿਸਟਰਕਟਰ ਦੇ ਨਾਲ)

```
class Base
{
    public :
    Base ( ) {cout << ‘‘Constructing Base’’;}
    //this is a virtual destructor ;
    virtual ~ Base ( ) {cout << ‘‘Destroying Base’’ ;}
};
```

ਨੋਟ : ਡਿਰਾਈਵਡ ਕਲਾਸ destructor ਨੂੰ base class ਤੋਂ ਪਹਿਲਾਂ call ਕੀਤਾ ਜਾਂਦਾ ਹੈ।

ਇਸ ਤਰ੍ਹਾਂ ਹੁਣ ਸਪੱਸ਼ਟ ਦਿਖਾਈ ਦਿੰਦਾ ਹੈ ਕਿ virtual destruction ਦੀ ਕਿਉਂ ਲੋੜ ਹੁੰਦੀ ਹੈ ਅਤੇ ਇਹ ਵੀ ਕਿ ਇਹ ਕਿਵੇਂ ਕੰਮ ਕਰਦੇ ਹਨ।

5.6 ਵੈਕਟਰ (Vector)

ਵੈਕਟਰਜ਼ ਉਹ ਲੜੀਵਾਰ ਕੰਟੇਨਰਜ਼ (Containers) ਹਨ ਜੋ ਕਿ ਐਰੇਜ (Arrays) ਨੂੰ ਦਰਸਾਉਂਦੇ ਹਨ ਅਤੇ ਆਪਣਾ ਆਕਾਰ (size) ਬਦਲਦੇ ਹਨ।

ਵੈਕਟਰ ਆਪਣੇ ਐਲੀਮੈਂਟ ਨੂੰ ਸਟੋਰ ਕਰਨ ਲਈ ਡਾਇਨਾਮਿਕ ਐਲੋਕੇਟਿਡ (Dynamically allocated) ਐਰੇ ਨੂੰ ਇਸਤੇਮਾਲ ਕਰਦਾ ਹੈ। ਇਸ ਐਰੇ ਨੂੰ ਰੀਐਲੋਕੇਟ (Reallocate) ਕਰਨ ਦੀ ਜ਼ਰੂਰਤ ਮਹਿਸੂਸ ਹੁੰਦੀ ਹੈ ਕਿਉਂਕਿ ਜਦੋਂ ਵੀ ਨਵੇਂ ਐਲੀਮੈਂਟਸ ਦਾਖਲ ਕੀਤੇ ਜਾਂਦੇ ਹਨ ਤਾਂ ਇਸ ਦੇ ਆਕਾਰ (size) ਵਿਚ ਵਾਧਾ ਹੁੰਦਾ ਹੈ ਅਤੇ ਇਕ ਨਵਾਂ ਐਰੇ (array) ਐਲੋਕੇਟ (allocate) ਕੀਤਾ ਜਾਂਦਾ ਹੈ ਤਾਂ ਜੋ ਉਸ ਵਿਚ ਸਾਰੇ ਐਲੀਮੈਂਟ ਸ਼ਿਫਟ (move) ਕੀਤੇ ਜਾ ਸਕਣ।

ਇਸ ਸਾਰੇ ਕੰਮ ਵਿਚ ਬਹੁਤ ਜ਼ਿਆਦਾ ਸਮਾਂ ਲੱਗਦਾ ਹੈ। ਇਸ ਲਈ ਇਸ ਦੇ ਬਜਾਇ ਵੈਕਟਰ ਕੰਟੇਨਰਜ਼ ਨੂੰ ਫਾਲਤੂ (etc.) ਸਟੋਰੇਜ ਐਲੋਕੇਟ ਕਰ ਦਿੱਤੀ ਜਾਂਦੀ ਹੈ ਤਾਂ ਜੋ ਜਿੰਨਾ ਐਲੀਮੈਂਟ ਦਾ ਵਾਧਾ ਉਸ ਵਿਚ ਸਮਾਇਆ ਜਾ ਸਕੇ। ਇਸ ਤਰ੍ਹਾਂ ਜੇ ਐਰੇ ਨਾਲ ਤੁਲਨਾ ਕਰੀਏ ਤਾਂ ਵੈਕਟਰਜ਼ ਜ਼ਿਆਦਾ ਮੈਮਰੀ ਦੀ ਵਰਤੋਂ ਕਰਦੇ ਹਨ।

ਉਦਾਹਰਣ 5.6

```
// constructing vectors
# include <iostream.h>
# include <vector>

int main ( )
{
unsigned int i ;
// constructors used in the same user as described above :
std :: vector < int > first ;    //empty vector of  ints
std :: vector < int > second (4,100) ;   //four ints with value 100
std :: vector < int > third (second.begin ( ), second.end( ) );  // iterating through second
std :: vector < int > fourth (third) ;   // a copy of third   the iterator constructor can also be
                                         // used to construct from arrays :
int myints [ ] = {16, 2, 77, 29};
std :: vector < int > fifth (myints, myints + size of (myints) / sizeof (int) );

std :: cout << ‘‘The contents of fifth are :’’;
for (std :: vector < int > :: iterator :: fifth.begain) ; it ! - fifth.end ( );++it)
std :: cont << ‘ ’ << * it ;
std :: cout << ‘\n’ ;
return 0 ;
}
```

Output :

The contents of fifth are : 16 2 77 2

ਯਾਦ ਰੱਖਣ ਯੋਗ ਗੱਲਾਂ (Points to be remembered) :

1. ਪੋਲੀਮਾਰਫਿਜ਼ਮ (Polymorphism) ਦਾ ਅਰਥ ਹੈ ਇਕ ਨਾਂ, ਕਈ ਕਿਸਮਾਂ (one name, multiple forms)
2. ਪੋਲੀਮਾਰਫਿਜ਼ਮ ਦੋ ਤਰ੍ਹਾਂ ਦਾ ਹੁੰਦਾ ਹੈ।
3. ਇਕ ਇੰਟਰਫੇਸ C++ ਕਲਾਸ ਦੀਆਂ ਸਮਰੱਥਾ ਜਾਂ ਉਸ ਦੇ ਵਿਵਹਾਰ ਨੂੰ ਪਰਿਭਾਸ਼ਿਤ ਕਰਦਾ ਹੈ।
4. virtual table ਨੂੰ ਅਸੀਂ vtable ਦੇ ਨਾਂ ਨਾਲ ਵੀ ਜਾਣਦੇ ਹਨ।
5. ਡਿਸਟਰਕਟਰ (Destructor) ਦੀ ਵਰਤੋਂ Memory ਨੂੰ ਖਾਲੀ ਕਰਵਾਉਣ ਲਈ ਕੀਤੀ ਜਾਂਦੀ ਹੈ।

ਅਭਿਆਸ
(Exercise)

1. ਖਾਲੀ ਥਾਵਾਂ ਭਰੋ—

1. ਪੋਲੀਮਾਰਫਿਜ਼ਮ ਦਾ ਅਰਥ ਹੈ ਇਕ ਨਾਂ, ...।
2. ਹਰੇਕ ਕਲਾਸ ਜੋ virtual function ਵਰਤ ਰਹੀ ਹੈ, ਉਸਦਾ ਆਪਣਾ ..... ਹੁੰਦਾ ਹੈ।
3. ..... ਨੂੰ ਅਸੀਂ ~ ਚਿੰਨ੍ਹ ਨਾਲ ਦਰਸਾਉਂਦੇ ਹਾਂ।
4. ਉਹ ਫੰਕਸ਼ਨ ਜੋ ਪਰਿਭਾਸ਼ਿਤ ਤਾਂ base class ਵਿਚ ਕੀਤਾ ਜਾਂਦਾ ਹੈ ਪਰ ਉਸ ਦਾ base class ਨਾਲ ਕੋਈ ਸੰਬੰਧ ਨਹੀਂ ਹੁੰਦਾ, ... ਅਖਵਾਉਂਦਾ ਹੈ।
5. ..... ਲੜੀਵਾਰ ਕੰਟੇਨਰਜ਼ ਹੁੰਦੇ ਹਨ।

2. ਛੋਟੇ ਉੱਤਰਾਂ ਵਾਲੇ ਪ੍ਰਸ਼ਨ——

1. ਪੋਲੀਮਾਰਫਿਜ਼ਮ ਬਾਰੇ ਇਕ ਨੋਟ ਲਿਖੋ।
2. ਅਬਸਟਰੈਕਟ ਬੇਸ ਕਲਾਸਾਂ ਕੀ ਹੁੰਦੀਆਂ ਹਨ ?
3. Pure virtual Function ਕਿਸ ਨੂੰ ਆਖਦੇ ਹਨ ?
4. ਸਟੈਟਿਕ ਬਾਈਂਡਿੰਗ ਅਤੇ ਡਾਇਨਾਮਿਕ ਬਾਈਂਡਿੰਗ ਵਿਚ ਕੀ ਅੰਤਰ ਹੈ ?
5. ਵੈਕਟਰ ਕੀ ਹੁੰਦੇ ਹਨ ?
6. ਵਰਚੂਅਲ ਫੰਕਸ਼ਨ ਦਾ ਉਦਾਹਰਨ ਸਹਿਤ ਵਰਣਨ ਕਰੋ।

Lab Activity

Program 1:

```
// pointers to base class
#include <iostream.h>
using namespace std;

class CPolygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
  };

class CRectangle: public CPolygon {
  public:
    int area ()
      { return (width * height); }
  };

class CTriangle: public CPolygon {
  public:
    int area ()
      { return (width * height / 2); }
  };

int main () {
  CRectangle rect;
  CTriangle trgl;
  CPolygon * ppoly1 = &rect;
  CPolygon * ppoly2 = &trgl;
  ppoly1->set_values (4,5);
  ppoly2->set_values (4,5);
  cout << rect.area() << endl;
  cout << trgl.area() << endl;
  return 0;
}
```

Program 2:

```
// virtual members
#include <iostream.h>
using namespace std;

class CPolygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
    virtual int area ()
      { return (0); }
  };

class CRectangle: public CPolygon {
  public:
    int area ()
      { return (width * height); }
  };
```

```
class CTriangle: public CPolygon {
  public:
    int area ()
      { return (width * height / 2); }
  };

int main () {
  CRectangle rect;
  CTriangle trgl;
  CPolygon poly;
  CPolygon * ppoly1 = &rect;
  CPolygon * ppoly2 = &trgl;
  CPolygon * ppoly3 = &poly;
  ppoly1->set_values (4,5);
  ppoly2->set_values (4,5);
  ppoly3->set_values (4,5);
  cout << ppoly1->area() << endl;
  cout << ppoly2->area() << endl;
  cout << ppoly3->area() << endl;
  return 0;
}
```

Program 3:

```
// abstract class CPolygon
class CPolygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
    virtual int area () =0;
};
```

Program 4:

```
// abstract base class
#include <iostream.h>
using namespace std;

class CPolygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
    virtual int area (void) =0;
  };

class CRectangle: public CPolygon {
  public:
    int area (void)
      { return (width * height); }
  };

class CTriangle: public CPolygon {
  public:
    int area (void)
      { return (width * height / 2); }
  };

int main () {
  CRectangle rect;
  CTriangle trgl;
  CPolygon * ppoly1 = &rect;
  CPolygon * ppoly2 = &trgl;
  ppoly1->set_values (4,5);
  ppoly2->set_values (4,5);
  cout << ppoly1->area() << endl;
  cout << ppoly2->area() << endl;
  return 0;
}
```

Program 5:

```
// pure virtual members can be called
// from the abstract base class
#include <iostream.h>
using namespace std;

class CPolygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
    virtual int area (void) =0;
    void printarea (void)
      { cout << this->area() << endl; }
  };

class CRectangle: public CPolygon {
  public:
    int area (void)
      { return (width * height); }
  };

class CTriangle: public CPolygon {
  public:
    int area (void)
      { return (width * height / 2); }
  };

int main () {
  CRectangle rect;
  CTriangle trgl;
  CPolygon * ppoly1 = &rect;
  CPolygon * ppoly2 = &trgl;
  ppoly1->set_values (4,5);
  ppoly2->set_values (4,5);
  ppoly1->printarea();
  ppoly2->printarea();
  return 0;
}
```

Program 6:

```
// dynamic allocation and polymorphism
#include <iostream.h>
using namespace std;

class CPolygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
    virtual int area (void) =0;
    void printarea (void)
      { cout << this->area() << endl; }
  };

class CRectangle: public CPolygon {
  public:
    int area (void)
      { return (width * height); }
  };
class CTriangle: public CPolygon {
  public:
    int area (void)
      { return (width * height / 2); }
  };
int main () {
  CPolygon * ppoly1 = new CRectangle;
  CPolygon * ppoly2 = new CTriangle;
  ppoly1->set_values (4,5);
  ppoly2->set_values (4,5);
  ppoly1->printarea();
  ppoly2->printarea();
  delete ppoly1;
  delete ppoly2;
  return 0;
}
```

ਪਾਠ 6

ਇਨਪੁਟ/ਆਊਟਪੁਟ ਫਾਈਲਜ਼
(Input/Output with Files)

6.1 C++ ਹੇਠਾਂ ਲਿਖੀਆਂ ਕਲਾਸਾਂ ਪ੍ਰਧਾਨ ਕਰਦੀ ਹੈ ਜੋ ਫਾਈਲਾਂ ਤੋਂ/ਤੇ ਇਨਪੁਟ ਅਤੇ ਆਊਟਪੁਟ ਪ੍ਰਫੋਰਮ (Perform) ਕਰਦੀਆਂ ਹਨ।

* ofstream : stream class ਫਾਈਲਾਂ ਤੇ ਲਿਖਣ ਲਈ
* ifstream : stream class ਫਾਈਲਾਂ ਤੋਂ ਪੜ੍ਹਣ ਲਈ
* fstream : stream class ਜੋ ਕਿ ਫਾਈਲਾਂ ਤੇ/ਤੋਂ ਪੜ੍ਹਦੀਆਂ ਵੀ ਹਨ ਤੇ ਲਿਖਦੀਆਂ ਵੀ ਹਨ।

ਇਹ ਕਲਾਸਾਂ (classes) ਸਿੱਧੇ ਅਤੇ ਅਸਿੱਧੇ ਤੌਰ ਤੇ ਕਲਾਸਾਂ istream ਅਤੇ ostream ਤੇ ਪ੍ਰਾਪਤ ਹੁੰਦੀਆਂ ਹਨ।ਅਸੀਂ ਪਹਿਲਾਂ ਹੀ ਉਹ ਆਬਜੈਕਟ, ਜਿਨ੍ਹਾਂ ਦੀਆਂ ਟਾਈਪਸ ਇਹ ਕਲਾਸਾਂ ਸਨ ਵਰਤ ਚੁੱਕੇ ਹਾਂ : cin ਇਕ class istream ਦਾ ਆਬਜੈਕਟ ਹੈ ਅਤੇ cout ਇਕ class ostream ਦਾ ਆਬਜੈਕਟ ਹੈ ਇਸ ਕਰਕੇ ਅਸੀਂ ਉਹ ਕਲਾਸਾਂ ਵਰਤ ਰਹੇ ਹਾਂ ਜੋ ਕਿ ਸਾਡੀਆਂ ਫਾਈਲ stream ਨਾਲ ਸੰਬੰਧਿਤ ਹਨ।

ਆਓ ਦੇਖੀਏ ਉਦਾਹਰਣ 6.1 :

```
 1 // basic file operations
 2 # include <iostream.h>
 3 # include <fstream.h>
 4 using namespace std;
 5
 6 int main ( ) {        [file example.txt] writing this to a file
 7 of stream my file;
 8 myfile.open (‘‘example.txt’’);
 9 myfile <<‘‘writing this to a file. \n’’;
10 myfile . close ( ) ;
11 return 0;
12 }
```

ਇੱਕ ਕੋਡ ਇੱਕ ਫਾਈਲ ਬਣਾਉਂਦਾ ਹੈ।ਇਸ ਨੂੰ example.txt ਕਹਿੰਦੇ ਹਨ ਜੋ ਕਿ ਇਕ ਵਾਰ ਇਨਸਰਟ ਕਰਦਾ ਹੈ, ਠੀਕ ਉਸ ਤਰ੍ਹਾਂ ਜਿਵੇਂ ਅਸੀਂ cout ਨਾਲ ਕਰਦੇ ਹਾਂ ਪਰ ਇਸ ਵਿਚ ਫਾਈਲ stream my file ਵਰਤਿਆ ਹੁੰਦਾ ਹੈ।

ਹੁਣ ਅਸੀਂ ਸਟੈਪ-ਸਟੈਪ ਅੱਗੇ ਚੱਲਦੇ ਹਾਂ।

6.2 Open a File

ਪਹਿਲਾਂ ਆਪਰੇਸ਼ਨ (Operation) ਇਹਨਾਂ 'ਚੋਂ ਹੀ ਇਕ ਕਲਾਸ ਦੇ ਆਬਜੈਕਟ ਉੱਪਰ ਹੁੰਦਾ ਹੈ ਜੋ ਕਿ ਫਾਈਲ ਰੀਡ (read) ਕਰਨ ਲਈ ਜੋੜਿਆ ਜਾਂਦਾ ਹੈ।ਇਸ ਪ੍ਰੋਸੈਸ ਨੂੰ Open a file ਕਿਹਾ ਜਾਂਦਾ ਹੈ।

ਇਕ ਫਾਈਲ ਜੋ ਕਿ stream object ਵਾਲੀ ਹੈ, ਨੂੰ ਖੋਲ੍ਹਣ ਲਈ ਅਸੀਂ member function open ( ) : ਵਰਤਦੇ ਹਾਂ।

Open (file name, mode) ;

ਜਿੱਥੇ filename ਇਕ null-terminated character ਜੋ ਕਿ type const char* ਹੈ ਅਤੇ ਕਿ ਫਾਈਲ ਦਾ ਨਾਮ ਦਰਸਾਉਂਦਾ ਹੈ ਜਿਸ ਨੂੰ ਖੋਲ੍ਹਣਾ (open) ਹੈ, ਹੇਠ ਲਿਖੇ flags ਦਾ mode ਆਪਸ਼ਨਲ ਪੈਰਾਮੀਟਰ ਹੈ।

ios :: in open for input operations
ios :: out open for output operations
ios :: binary open in binary mode
ios :: ate set the initial position at the end of file
ios : app the current content of the file.
ios :: trunc if the file opened for output operations already existed before

ਇਹ ਸਾਰੇ Flags bitwise operator OR ( ) ਨਾਲ ਜੋੜੇ ਜਾ ਸਕਦੇ ਹਨ।

ਮਿਸਾਲ :

```
1 of stream myfile ;
2 my file . open (‘‘example.bin’’, ios :: out / ios :: app ios :: binary) ;
```

6.3 Closing a file

ਜਦੋਂ ਅਸੀਂ ਆਪਣੇ input ਅਤੇ output operations on file ਖ਼ਤਮ ਕਰ ਲੈਂਦੇ ਹਾਂ ਤਾਂ ਅਸੀਂ ਇਹਨਾਂ ਨੂੰ ਬੰਦ ਕਰ ਸਕਦੇ ਹਾਂ। ਇਸ ਨੂੰ ਕਰਨ ਲਈ streams member function close ( ) ਨੂੰ ਬੁਲਾਇਆ ਜਾਂਦਾ ਹੈ ਤਾਂ ਕਿ ਇਹਨਾਂ ਦੇ ਰੀਸੋਰਸ ਦੁਬਾਰਾ ਉਪਲਬਧ ਹੋ ਜਾਣ।

```
my file.close ( ) ;
```

ਇਕ ਵਾਰ ਜਦੋਂ ਮੈਂਬਰ ਫੰਕਸ਼ਨ ਨੂੰ ਬੁਲਾਇਆ ਜਾਂਦਾ ਹੈ ਤਾਂ stream object ਦੂਸਰੀ file ਨੂੰ ਖੋਲ੍ਹਣ ਲਈ ਵਰਤੇ ਜਾਂਦੇ ਹਨ ਅਤ, ਦੂਸਰੇ ਪ੍ਰੋਸੈਸ ਨੂੰ ਖੋਲ੍ਹਣ ਲਈ , ਫਾਈਲ ਦੁਬਾਰਾ ਉਪਲਬਧ ਹੋ ਜਾਂਦੀ ਹੈ ।

6.4 Text Files

Text file stream ਉਹ ਹਨ ਜਿੱਥੇ ਅਸੀਂ ios :: binary flag ਓਪਨ ਮੋਡ ਵਿਚ ਨਹੀਂ ਵਰਤੇ ਜਾਂਦੇ।

ਇਹ ਫਾਈਲਾਂ ਟੈਕਸਟ ਸਟੋਰ ਕਰਨ ਲਈ ਡਿਜ਼ਾਈਨ ਕੀਤੀਆਂ ਜਾਂਦੀਆਂ ਹਨ ਅਤੇ ਸਾਡੇ ਰਾਹੀਂ ਇਨਪੁਟ ਜਾਂ ਆਉਟਪੁਟ ਕੀਤੀਆਂ ਸਾਰੀਆਂ ਕੀਮਤਾਂ ਫਾਰਮੈਟਿੰਗ ਟ੍ਰਾਂਸਫੋਰਮੇਸ਼ਨ ਤੋਂ ਪ੍ਰਭਾਵਿਤ ਹੁੰਦੀਆਂ ਹਨ। ਡਾਟਾ ਆਊਟਪੁਟ operation ਟੈਕਸਟ ਫਾਈਲਜ਼ ਨਾਲ ਉਸੇ ਤਰ੍ਹਾਂ ਵਰਤੇ ਜਾਂਦੇ ਹਨ, ਜਿਸ ਤਰ੍ਹਾਂ ਅਸੀਂ cout: ਨਾਲ ਵਰਤਦੇ ਹਾਂ।

```
1. // 6.4 Writing on a text file
2. # include <iostream.h>          [file example.txt]
3. # include <fstream.h>          This is a line. This is another line.
4. using namespace std;
5.
6. int main ( ) {
7. ofstream my file (‘‘example txt’’) ;
8. if (my fle.is-open ( ) )
9. {
10. my file << ‘‘This is a line \n’’ ;
11. my file << ‘‘This is another line \n’’;
12. my file close ( ) ;
13. }
14. else cout << ‘‘unable to open file’’ ;
15. return 0 ;
16. }
```

Data input ਫਾਈਲ ਤੋਂ ਉਸ ਤਰ੍ਹਾਂ ਹੀ ਚੱਲਦਾ ਹੈ ਜਿਸ ਤਰ੍ਹਾਂ ਅਸੀਂ cin: ਨਾਲ

```
1. // reading a text file
2. # include <iostream.h>
3. # include <fstream.h>
4. # include <string.h>
5. using namespace std;
6.
7. int main ( ) {
8. string line ;
9. ifstream myfile (‘‘example.txt’’);
10. if (my file is – open ( ) )
11. {
12. While (my file good ( ) )  This is line.
13. {                This is another line.
14. getline (my file, line) ;
15. cout << line << endl ;
16. }
17. my file, close ( ) ;
18. }
19.
20. else cout << ‘‘unable to open file’’ ;
21.
22. return 0 ;
23. }
```

ਇਹ ਆਖਰੀ ਉਦਾਹਰਣ ਇਕ ਟੈਕਸਟ File read ਕਰਦੀ ਹੈ ਅਤੇ ਇਸ ਦੇ ਕਾਨਟੈਂਟ ਸਕਰੀਨ ਤੇ ਛਾਪਦੀ ਹੈ। ਇਹ ਨੋਟ ਕੀਤਾ ਜਾਵੇ ਕਿ ਕਿਸ ਤਰ੍ਹਾਂ ਅਸੀਂ ਇਕ ਨਵਾਂ ਮੈਂਬਰ ਫੰਕਸ਼ਨ good ( ) ਵਰਤਿਆ ਹੈ ਜਿਹੜਾ Stream Input/Output Operation ਲਈ ਤਿਆਰ ਕੇਸ ਵਿਚ true ਹੈ ।

6.5 Checking State Flags

ਇਕ ਮੈਂਬਰ ਫੰਕਸ਼ਨ good ( ) ਜੋ ਕਿ ਚੈੱਕ ਕਰਦਾ ਹੈ ਕਿ ਸਟਰੀਮ (stream) input/output operations ਦੇ ਲਈ ਤਿਆਰ ਹੈ, ਨਾਲ ਇਕ ਹੋਰ ਮੈਂਬਰ ਫੰਕਸ਼ਨ ਮੌਜੂਦ ਹੈ ਜੋ ਕਿ ਚੈੱਕ ਕਰਦਾ ਹੈ ਕਿ ਸਾਰੇ states of stream boal value ਵਾਪਿਸ ਕਰਦੇ ਹਨ।

bad ( )

ਇਹ true ਵਾਪਿਸ ਕਰਦਾ ਹੈ ਜੇਕਰ ਸਾਰੇ ਰੀਡਿੰਗ (reading) ਅਤੇ ਰਾਈਟਿੰਗ (writing) ਅਪਰੇਸ਼ਨ ਫੇਲ੍ਹ ਹੋਣ।

fail ( )

ਇਹ true ਵਾਪਿਸ ਕਰਦਾ ਹੈ ਉਸ ਤਰ੍ਹਾਂ ਜਿਸ ਤਰ੍ਹਾਂ bad ( ),ਪਰ ਜਦੋਂ ਉਸ ਵਿੱਚ ਫਾਰਮੈਂਟ ਐਰਰ ਆਵੇ ਤਾਂ ਇੰਟੀਜਰ ਨਬਰ ਪੜ੍ਹਦੇ ਹੋਏ ਇਕ ਐਲਫਾਬੈਟੀਕਲ (alphabetical character) ਐਕਸਟਰੈਕਟ ਕੀਤਾ ਜਾਂਦਾ ਹੈ।

eof ( )

ਇਹ true ਵਾਪਿਸ ਕਰਦਾ ਹੈ ਜੇਕਰ ਫਾਈਲ ਪੜ੍ਹਣ ਲਈ ਖੁੱਲ੍ਹੀ ਹੈ ਅਤੇ ਅਖ਼ੀਰ ਵਿਚ ਪਹੁੰਚ ਚੁੱਕੀ ਹੈ।

good ( )

ਇਹ ਇਕ ਅਤਿਅੰਤ ਜੈਨਰਿਕ (generic) state flag : ਹੈ ਜੋ ਕਿ false ਵਾਪਿਸ ਕਰਦਾ ਹੈ ਜਿਨ੍ਹਾਂ ਕੇਸਾਂ ਵਿਚ ਪਹਿਲੇ ਫੰਕਸ਼ਨ ਨੇ true ਵਾਪਿਸ ਕੀਤਾ ਹੋਵੇ।

State flags ਨੂੰ reset ਕਰਨ ਲਈ ਜੋ ਕਿ ਕੋਈ ਵੀ ਮੈਂਬਰ ਫੰਕਸ਼ਨ ਦੁਆਰਾ ਚੈਕ ਕੀਤੇ ਗਏ ਹਨ ਅਸੀਂ member ਫੰਕਸ਼ਨ clear( ) ਵਰਤ ਸਕਦੇ ਹਾਂ ਜੋ ਕਿ ਕੋਈ ਵੀ Parameter ਨਹੀਂ ਲੈਂਦੇ।

6.6 get and Put stream Pointers

ਸਾਰੇ i/o stream ਆਬਜੈਕਟ ਕੋਲ ਘੱਟ ਤੋਂ ਘੱਟ ਇਕ internal stream pointer ਹੁੰਦਾ ਹੈ।

ifstream, istream ਦੀ ਤਰ੍ਹਾਂ ਕੋਲ ਇਕ Pointer get Pointer ਹੁੰਦਾ ਹੈ ਜੋ ਕਿ ਅਗਲੇ input operation ਵਿੱਚ ਪੜ੍ਹੇ ਜਾਣ ਵਾਲੇ ਐਲੀਮੈਂਟ ਵੱਲ ਇਸ਼ਾਰਾ ਕਰਦਾ ਹੈ।

ofstream, ostream ਦੀ ਤਰ੍ਹਾਂ ਕੋਲ ਇਕ Put Pointer ਹੁੰਦਾ ਹੈ ਜੋ ਕਿ ਉਸ ਜਗ੍ਹਾ ਵੱਲ ਇਸ਼ਾਰਾ ਕਰਦਾ ਹੈ, ਜਿੱਥੇ ਅਗਲਾ ਐਲੀਮੈਂਟ ਲਿਖਿਆ ਜਾਣਾ ਹੈ।

ਅਖ਼ੀਰ ਵਿਚ fstream ਜੋ ਕਿ ਦੋਨਾਂ get ਅਤੇ Put Pointer ਤੋਂ ਆਇਆ ਹੈ ਜੋ ਕਿ ਅੱਗੋਂ iostream ਤੋਂ ਆਇਆ ਹੈ। ਇਹ ਅੰਦਰੂਨੀ Stream Pointer Stream ਦੇ ਅੰਦਰ ਰੀਡਿੰਗ ਅਤੇ ਰਾਈਟਿੰਗ ਲੋਕੇਸ਼ਨ ਵੱਲ Point ਕਰਦੀਆਂ ਹਨ, ਨੂੰ ਹੇਠ ਲਿਖੇ ਮੈਂਬਰ ਫੰਕਸ਼ਨ ਦੁਆਰਾ ਮੈਨੀਪੁਲੇਟ (manipulate) ਕੀਤਾ ਜਾ ਸਕਦਾ ਹੈ।

6.6.1 tellg ( ) and tellp ( )

ਇਹਨਾਂ ਦਹਾਂ ਮੈਂਬਰ ਫੰਕਸ਼ਨ ਦਾ ਕੋਈ ਪੈਰਾਮੀਟਰ ਨਹੀਂ ਹੁੰਦਾ ਅਤੇ POS-type ਦੇ ਮੈਂਬਰ ਟਾਈਪ (type) ਦੀ ਵੈਲਿਊ (Value) ਵਾਪਿਸ ਕਰਦੀਆਂ ਹਨ ਜੋ ਕਿ integer data type ਹੁੰਦਾ ਹੈ। ਇਸ ਪੋਜੀਸ਼ਨ (Current Position) ਵਿਚ ਇਹ integer data type tell g ਦੇ ਕੇਸ ਵਿਚ get steam pointer ਹੁੰਦਾ ਹੈ ਅਤੇ ਉਸ ਕੇਸ 'ਚ tell P ਦਾ Put Stream Pointer ਹੁੰਦਾ ਹੈ।

6.6.2 seekg( ) and seekp( )

ਇਹ ਫੰਕਸ਼ਨ ਸਾਨੂੰ get ਅਤੇ Put stream pointer ਦੀ ਪੋਜੀਸ਼ਨ (Position) ਬਦਲਣ ਦੀ ਆਗਿਆ ਦਿੰਦੇ ਹਨ। ਦੋਵੇਂ ਫੰਕਸ਼ਨਜ ਅਲੱਗ-ਅਲੱਗ Prototypes ਦੁਆਰਾ ਓਵਰਲੋਡਿਡ (overloaded) ਹੁੰਦੇ ਹਨ।

* ਪਹਿਲਾ Prototype ਹੈ।
  seek g (Position) ;
  seek P (Position) ;
* ਦੂਸਰਾ Prototype ਹੈ।
  seek g (offset, direction) ;
  seek P (offset, direction) ;

ਹੇਠਲੀ ਉਦਾਹਰਣ 6.6.2 ਮੈਂਬਰ ਫੰਕਸ਼ਨ ਵਰਤਦੀ ਹੈ ਜੋ ਫਾਈਲ ਦਾ ਸਾਈਜ਼ ਪ੍ਰਾਪਤ ਕਰਨ ਲਈ ਹੈ।

```
// obtaining file size
# include <iostream.h>
# include <fstream.h>
using namespace std;
int main ( )
{
long begin end ;
if stream my file (''example.txt''); size is : 40 bytes
begin = mgfile.tellg ( );
my file.seekg (0, ios :: end) ;
end = my file.tell g ( ) ;
myfile.close ( ) ;
cout << ''size is :'' << (end-begin) << ''bytes \n'' ;
return 0;
}
```

6.7 ਸਟਰੀਮ ਮੇਨੀਪੁਲੇਟਰਸ (Stream Manipulators)

ਸਟਰੀਮ ਮੇਨੀਪੁਲੇਟਰਸ (Stream Manipulators) ਦੀ ਵਰਤੋਂ ਫਾਰਮੇਟਿੰਗ (Formating) ਕਰਨ ਲਈ ਕੀਤੀ ਜਾਂਦੀ ਹੈ, ਜਿਵੇਂ ਕਿ——

* ਫੀਲਡ ਵਿਡਥ (Field width) ਨੂੰ ਸੈੱਟ ਕਰਨਾ।
* ਪਰੀਸਿਜ਼ਨ (Precision)
* ਫਾਰਮੈਟ ਫਲੇਗਸ (Format Flags) ਨੂੰ ਅਨਸੈਟ (Unset) ਕਰਨਾ।
* ਆਊਟਪੁਟ ਸਟਰੀਮ (Output stream) ਵਿਚ ਨਵੀਂ ਲਾਈਨ ਦਾਖਲ ਕਰਨਾ ਅਤੇ ਸਟਰੀਮ ਨੂੰ flush ਕਰਨਾ।
* ਆਊਟਪੁਟ ਸਟਰੀਮ (Output stream) ਵਿਚ ਨੱਲ (NULL) ਕਰੈਕਟਰ (character) ਨੂੰ ਦਾਖਲ ਕਰਨਾ।
* ਵਾਈਟਸਪੇਸ (whitespace) ਨੂੰ ਸਕਿਪ (skip) ਕਰਨਾ।
* ਫੀਲਡ ਵਿਚ ਫੀਲਡ ਕਰੈਕਟਰ ਨੂੰ ਸੈਟ ਕਰਨਾ।

6.8 ਸਟਰੀਮ ਬੇਸ (Stream Base)

6.8.1 hex. ਬੇਸ (Base) ਨੂੰ ਹੈਕਸਾਡੈਸੀਮਲ (hexadecimal) ਬੇਸ 16 ਤੇ ਸੈੱਟ ਕਰਨ ਲਈ।

6.8.2 Oct. ਬੇਸ ਨੂੰ Octal (ਆਕਟਲ) ਬੇਸ 8 ਤੇ ਸੈੱਟ ਕਰਨ ਲਈ।

6.8.3 dec. ਸਟਰੀਮ (stream) ਨੂੰ ਡੈਸੀਮਲ (decimal) ਤੇ reset ਕਰਨ ਲਈ।

6.8.4 set base ( ). ਇਹ ਇਕ ਪੈਰਾਮੀਟਰਾਈਜ਼ਡ (Parameterized) ਮੈਨੀਪੁਲੇਟਰ (manipulator) ਹੈ ਜੋ ਕਿ 10, 8 ਜਾਂ ਫਿਰ 16 ਨੂੰ Parameter (ਪੈਰਾਮੀਟਰ) ਦੇ ਤੌਰ ਤੇ ਲੈਂਦਾ ਹੈ ਤਾਂ ਜੋ ਇੰਨਟੀਜਰ (Integer) ਉਸ ਬੇਸ ਤੇ Print ਹੋ ਸਕੇ ਜਿਵੇਂ ਕਿ set base (16) ਵੀ hex operator ਦੀ ਤਰ੍ਹਾਂ ਹੀ ਕੰਮ ਕਰੇਗਾ।

6.8.5 flush. ਇਹ Processing ਦੇ ਸ਼ੁਰੂ ਹੋਣ ਤੋਂ ਪਹਿਲਾਂ Output buffer ਨੂੰ flush ਕਰਨ ਦੇ ਕੰਮ ਆਉਂਦਾ ਹੈ।

6.8.6 endl. ਇਹ ਇਕ ਨਵੀਂ ਲਾਈਨ ਪ੍ਰਿੰਟ ਕਰਦਾ ਹੈ ਤੇ Output buffer ਨੂੰ flush ਕਰ ਦਿੰਦਾ ਹੈ।

6.8.7 width( ). ਇਹ ਫੀਲਡ width ਨੂੰ ਸੈਟ ਕਰਦਾ ਹੈ। ਜੇਕਰ enter ਕੀਤੀ ਗਈ Value, Field Width ਨਾਲੋਂ ਛੋਟੀ ਤੇ ਤਾਂ ਇਹ fill characters ਨੂੰ Padding ਦੇ ਤੌਰ ਤੇ ਦਾਖਲ ਕਰਦਾ ਹੈ।

6.8.8 fill( ). ਇਹ ਫੰਕਸ਼ਨ fill character ਨੂੰ ਭਰਨ ਦੇ ਕੰਮ ਆਉਂਦਾ ਹੈ (ਜਦੋਂ ਅਸੀਂ width( ) ਫੰਕਸ਼ਨ ਨੂੰ ਵਰਤ ਰਹੇ ਹੁੰਦੇ ਹਾਂ)।

6.8.9 setw( ). ਇਹ ਇਕ ਪੈਰਾਮੀਟਰਾਈਜਡ ਮੈਨੀਪੁਲੇਟਰ (Parameterized Manipulator) ਹੈ ਤੇ width( ) ਫੰਕਸ਼ਨ ਦੀ ਤਰ੍ਹਾਂ ਹੀ ਕੰਮ ਕਰਦਾ ਹੈ।

6.8.10 set file( ). ਇਹ ਇਕ Parameterized Manipulator ਹੈ ਤੇ file ( ) ਫੰਕਸ਼ਨ ਦੀ ਤਰ੍ਹਾਂ ਹੀ ਕੰਮ ਕਰਦਾ ਹੈ।

6.9 ਬਾਇਨਰੀ ਫਾਈਲਜ਼ (Binary Files)

ਬਾਇਨਰੀ ਫਾਈਲਜ਼ ਵਿਚ ਡਾਟਾ ਨੂੰ ਐਕਸਟਰੈਕਸ਼ਨ (extraction) ਇਨਸਰਸ਼ਨ ਅਤੇ ਫੰਕਸ਼ਨ ਓਪਰੇਟਰ (ਜਿਵੇਂ ਕਿ getline ਨਾਲ ਇਨਪੁਟ ਅਤੇ ਆਊਟਪੁਟ) ਕਰਨਾ ਠੀਕ ਨਹੀਂ ਹੈ। ਇਸ ਕਰਕੇ ਸਾਨੂੰ ਕਿਸੇ ਵੀ ਡਾਟਾ ਵਿਚ ਤਬਦੀਲੀ ਕਰਨ ਦੀ ਲੋੜ ਨਹੀਂ ਹੈ। ਡਾਟਾ ਨੂੰ ਜਿਵੇਂ ਕਿ Space, newline code ਅਲੱਗ ਤੋਂ ਦੇਣ ਦੀ ਲੋੜ ਨਹੀਂ ਹੈ।

ਫਾਈਲ ਸਟਰੀਮ (File stream) ਵਿਚ ਦੋ ਮੈਂਬਰ ਫੰਕਸ਼ਨ ਡੀਜ਼ਾਈਨ ਕੀਤੇ ਗਏ ਹਨ ਜਿਹੜੇ ਇਨਪੁਟ ਅਤੇ ਆਊਟਪੁਟ ਬਾਇਨਰੀ ਡਾਟਾ ਨੂੰ ਕ੍ਰਮਵਾਰ ਲੈਂਦੇ ਹਨ।

* Write and Read.

* ਪਹਿਲਾਂ (write) ostream ਦਾ ਮੈਂਬਰ ਫੰਕਸ਼ਨ ਹੈ ਜੋ ਕਿ ofstream ਤੋਂ ਮਿਲਿਆ ਹੈ ਅਤੇ (read) istream ਦਾ ਮੈਂਬਰ ਫੰਕਸ਼ਨ ਹੈ ਜੋ ਕਿ ifstream ਤੋਂ ਮਿਲਿਆ ਹੈ। ਕਲਾਸ ifstream ਦੇ ਆਬਜੈਕਟ ਕੋਲ ਦੋਵੇਂ ਮੈਂਬਰਜ਼ ਹੁੰਦੇ ਹਨ ਅਤੇ ਉਹਨਾਂ ਦੇ ਪ੍ਰੋਟੋਟਾਈਪ (Prototype) ਹੇਠਾਂ ਹਨ :

```
write (memory_block, size);
read (memory_block, size);
```

ਉਦਾਹਰਣ 6.9

```
// reading a complete binary file
# include <iostream.h>
# include <fstream.h>
using namespace std;
ifstream :: POS-type size ;
char * memblock ;
int main ( )
 {
     ifstream file (‘‘example.bin’’,
     ios :: in/ios :: binary / ios :: ate) ;
      if (file . is - open ( ) )
{
size = file.tell g ( ) ;     the complete file content is in memory
memblock = new char [size] ;
file.seekg (0, ios :: beg) ;
file.read (memblock, size) ;
file.close ( );
```

```
cout << ''the complete file content is in
memory'';
delete [ ] memblock ;
}
else cout << ''unable to open file'';
return 0;
}
```

ਇਸ ਉਦਾਹਰਣ ਵਿਚ ਸਾਰੀ ਫਾਈਲ ਮੈਮਰੀ ਬਲਾਕ ਵਿਚ ਪੜ੍ਹੀ (read) ਅਤੇ ਸਟੋਰ ਕੀਤੀ ਜਾਂਦੀ ਹੈ।

6.10 ਬਫਰਸ ਅਤੇ ਸਿਨਕਰੋਨਾਈਜੇਸ਼ਨ (Buffers and Synchronization)

ਜਦੋਂ ਅਸੀਂ ਫਾਈਲ ਸਟਰੀਮ (file stream) ਨੂੰ ਓਪਰੇਟ ਕਰਦੇ ਹਾਂ ਤਾਂ ਇਹ ਇਨਟਰਨਲ ਬਫਰ ਜਿਸ ਦੀ ਟਾਈਪ (type) streambuf. ਨਾਲ ਜੁੜੇ ਹੁੰਦੇ ਹਨ।

ਇਹ ਬਫਰ ਇਕ ਮੈਮਰੀ ਦਾ ਬਲਾਕ ਹੁੰਦੇ ਹਨ ਜੋ ਕਿ stream ਅਤੇ ਫਿਜ਼ੀਕਲ ਫਾਈਲ (Physical file) ਦੇ ਵਿਚਕਾਰ ਇੰਟਰਮੀਡੀਏਟ (intermediary) ਹੁੰਦੇ ਹਨ।

ਉਦਾਹਰਣ ਵਜੋਂ ਜਦੋਂ ofstring ਨਾਲ ਸੰਬੰਧਤ ਮੈਂਬਰ ਫੰਕਸ਼ਨ put ਨੂੰ call ਕੀਤਾ ਜਾਂਦਾ ਹੈ ਤਾਂ ਅੱਖਰ ਸਿੱਧੇ ਤੌਰ ਤੇ ਸਟਰਿੰਗ (string) ਨਾਲ ਸੰਬੰਧਿਤ ਫਿਜ਼ੀਕਲ ਫਾਈਲ (physical file) ਉੱਤੇ ਨਹੀਂ ਲਿਖੇ ਜਾਂਦੇ ਸਗੋਂ ਇਸ ਦੇ ਅੱਖਰ stream ਦੇ ਇੰਟਰਮੀਡੀਏਟ ਬੱਫਰ ਦੇ ਵਿਚ ਦਾਖਲ ਹੋ ਜਾਂਦੇ ਹਨ।

ਜਦੋਂ ਬੱਫਰ (Buffer) ਨੂੰ ਫਲੱਸ਼ (flush) ਕੀਤਾ ਜਾਂਦਾ ਹੈ ਤਾਂ ਜੋ ਇਸ ਦੇ ਅੰਦਰ ਪਿਆ ਡਾਟਾ ਜੇਕਰ Output stream ਹੋਵੇ ਤਾਂ ਇਸ ਨੂੰ ਫਿਜ਼ੀਕਲ ਮਾਧਿਅਮ (Physical medium) ਵਿੱਚ ਲਿਖਿਆ ਜਾਂਦਾ ਹੈ। ਜੇਕਰ ਇਹ ਡਾਟਾ Input stream ਹੈ ਜਾਂ ਫਿਰ ਬਿਲਕੁਲ ਫਰੀ (free) ਹੋ ਜਾਂਦਾ ਹੈ, ਇਸ ਪ੍ਰਕਿਰਿਆ ਨੂੰ synchronisation ਕਹਿੰਦੇ ਹਨ ਅਤੇ ਇਹ ਹੇਠ ਲਿਖੀਆਂ ਪਰਿਸਥਿਤੀਆਂ ਵਿਚ ਹੁੰਦੀ ਹੈ—

1. ਜਦੋਂ ਫਾਈਲ ਨੂੰ ਬੰਦ ਕੀਤਾ ਜਾਂਦਾ ਹੈ।
2. ਜਦੋਂ ਬੱਫਰ ਭਰ ਜਾਂਦਾ ਹੈ।
3. ਮੈਨੀਪੁਲੇਟਰਜ਼ ਨਾਲ ਬਾਹਰੀ ਤੌਰ ਤੇ (Explicity) ਜਦੋਂ ਕੁਝ ਮੈਨੀਪੁਲੇਟਰਜ਼ stream ਦੇ ਉੱਤੇ ਇਸਤੇਮਾਲ ਕੀਤੇ ਜਾਂਦੇ ਹਨ ਉਸ ਸਮੇਂ ਬਾਹਰੀ synchronization ਹੁੰਦੀ ਹੈ। ਇਹ ਮੈਨੀਪੁਲੇਟਰਜ਼ ਹਨ (Flush and end)
4. ਮੈਂਬਰ ਫੰਕਸ਼ਨਜ਼ ਨਾਲ ਬਾਹਰੀ ਤੌਰ ਤੇ ਜਦੋਂ stream ਦੇ ਮੈਂਬਰ ਫੰਕਸ਼ਨ sync ( ) ਨੂੰ call ਕੀਤਾ ਜਾਂਦਾ ਹੈ ਤਾਂ ਉਸੇ ਦੌਰਾਨ synchronization ਹੁੰਦੀ ਹੈ।

ਯਾਦ ਰੱਖਣ ਯੋਗ ਗੱਲਾਂ (Points to be Remember) :

1. ofstream ਕਲਾਸ ਫਾਈਲਾਂ ਤੇ ਲਿਖਣ ਲਈ, ifstream class ਫਾਈਲਾਂ ਤੋਂ ਪੜ੍ਹਣ ਲਈ ਅਤੇ fstream ਕਲਾਸ ਤੋਂ ਪੜ੍ਹਣ ਤੇ ਲਿਖਣ ਲਈ ਵਰਤੀਆਂ ਜਾਂਦੀਆਂ ਹਨ।
2. ਮੈਂਬਰ ਫੰਕਸ਼ਨ (Member function) Open ( ) ਅਤੇ Close ( ) ਫਾਈਲ ਨੂੰ ਖੋਲ੍ਹਣ ਅਤੇ ਬੰਦ ਕਰਨ ਲਈ ਵਰਤਿਆ ਜਾਂਦਾ ਹੈ।
3. ਸਾਰੇ I/O stream ਆਬਜੈਕਟ ਕੋਲ ਘੱਟ ਤੋਂ ਘੱਟ ਇਕ Internal Stream Pointer ਹੁੰਦਾ ਹੈ।
4. Stream Manipulators ਫਾਰਮੇਟਿੰਗ ਕਰਨ ਲਈ ਵਰਤੇ ਜਾਂਦੇ ਹਨ।
5. Write ਅਤੇ Read ਦੋ ਅਜਿਹੇ ਮੈਂਬਰ ਫੰਕਸ਼ਨ ਹਨ ਜੋ ਕਿ ਇਨਪੁਟ ਅਤੇ ਆਉਟਪੁਟ ਬਾਈਨਰੀ ਡਾਟਾ ਨੂੰ ਕ੍ਰਮਵਾਰ ਲੈਂਦੇ ਹਨ।
6. Buffer ਇਕ ਮੈਮਰੀ ਦਾ ਬਲਾਕ (Block) ਹੁੰਦੇ ਹਨ ਜੋ ਕਿ stream ਅਤੇ ਫਿਜ਼ੀਕਲ ਫਾਈਲ ਦੇ ਵਿਚਕਾਰ ਇੰਟਰਮੀਡੀਏਟ (Intermediate) ਹੁੰਦੇ ਹਨ।

ਅਭਿਆਸ
(Exercise)

1. ਖਾਲੀ ਥਾਵਾਂ ਭਰੋ—

1. ਫਾਈਲਾਂ ਤੇ ਲਿਖਣ ਲਈ ..... stream class ਵਰਤੀ ਜਾਂਦੀ ਹੈ।
2. ਫਾਈਲ ਨੂੰ ਖੋਲ੍ਹਣ ਲਈ ..... member function ਦੀ ਵਰਤੋਂ ਕੀਤੀ ਜਾਂਦੀ ਹੈ।
3. Stream Manipulators ..... ਕਰਨ ਲਈ ਵਰਤੇ ਜਾਂਦੇ ਹਨ।
4. ..... Manipulator ਇਕ ਨਵੀਂ ਲਾਈਨ ਨੂੰ ਪ੍ਰਿੰਟ ਕਰਦਾ ਹੈ।
5. ..... ਅਤੇ ..... ਫੰਕਸ਼ਨ ਸਾਨੂੰ get ਅਤੇ Put Stream Pointer ਦੀ Position ਬਦਲਣ ਦੀ ਆਗਿਆ ਦਿੰਦਾ ਹੈ।

2. ਸਹੀ ਗਲਤ ਦਸੋ—

1. SetW( ) ਅਤੇ Width( ) ਫੰਕਸ਼ਨ ਦਾ ਕੰਮ ਫੀਲਡ Width ਨੂੰ ਸੈਟ ਕਰਨਾ ਹੈ।
2. hex ਬੇਸ ਨੂੰ 8 ਤੇ ਸੈਟ ਕਰ ਦਿੰਦਾ ਹੈ।
3. ਜੇਕਰ ਸਾਰੇ ਰੀਡਿੰਗ (reading) ਅਤੇ ਰਾਈਟਿੰਗ (writing) ਉਪਰੇਸ਼ਨ ਫੇਲ੍ਹ ਹੋਵੇ ਤਾਂ bad ( ) False Value ਦਿੰਦਾ ਹੈ।
4. ਜਦੋਂ Buffer ਭਰ ਜਾਂਦਾ ਹੈ ਤਾਂ Synchronization ਦੀ ਪ੍ਰਕ੍ਰਿਆ ਹੁੰਦੀ ਹੈ।

3. ਛੋਟੇ ਉੱਤਰ ਵਾਲੇ ਪ੍ਰਸ਼ਨ—

1. bad( ) ਅਤੇ good( ) ਵਿਚ ਅੰਤਰ ਲਿਖੋ।
2. Text ਫਾਈਲਜ ਉੱਤੇ ਸੰਖੇਪ ਨੋਟ ਲਿਖੋ।
3. Stream Manipulators ਕੀ ਹਨ ?
4. ਬੱਫਰ (Buffer) ਤੋਂ ਕੀ ਭਾਵ ਹੈ ?
5. hex, oct ਅਤੇ Dec ਮੈਨੀਪੁਲੇਟਰਜ਼ (manipulators) ਬਾਰੇ ਦੱਸੋ।

Lab Activity

Example 1

```
* Program to create a file and write some data on the file */
#include <stdio.h>
#include <stdio.h>
main( )
{
     FILE *fp;
char stuff[25];
int index;
fp = fopen("TENLINES.TXT","w"); /* open for writing */
strcpy(stuff,"This is an example line.");
for (index = 1; index <= 10; index++)
       fprintf(fp,"%s Line number %d\n", stuff, index);
fclose(fp); /* close the file before ending program */
}
```

Example 2

```
/* Program to display the contents of a file on screen */
#include <stdio.h>
void main()
{
   FILE *fopen(), *fp;
int c;
fp = fopen("prog.c","r");
   c = getc(fp) ;
while (c!= EOF)
   {
              putchar(c);
              c = getc(fp);
   }
```

Example 3

```
#include <stdio.h>
int main()
{
 FILE *fp;
file = fopen("file.txt","w");
 /*Create a file and add text*/
fprintf(fp,"%s","This is just an example :)"); /*writes data to the file*/
fclose(fp); /*done!*/
return 0;
}
```

Example 4

```
#include <stdio.h>
int main()
{
    FILE *fp
file = fopen("file.txt","a");
fprintf(fp,"%s","This is just an example :)"); /*append some text*/
fclose(fp);
return 0;
}
```

Example 5

```
#include <stdio.h>
main( )
   {
     FILE *fp;
char c;
funny = fopen("TENLINES.TXT", "r");
if (fp == NULL)
              printf("File doesn't exist\n");
else {
do {
       c = getc(fp); /* get one character from the file
       */
putchar(c); /* display it on the monitor
```

Example related to iostream

Program 1

```
#include <iostream.h>

using namespace std;

int main( )
{
charstr[] = "Hello C++";

cout<< "Value of str is : " <<str<<endl;
}
```

Program 2

```
#include <iostream.h>
intmain()
{
    usingnamespacestd;
    // First we'll use the insertion operator on cout to print text to the
monitor
    cout<< "Enter your age: "<<endl;

    // Then we'll use the extraction operator on cin to get input from the
user
    intnAge;
    cin>>nAge;

    if(nAge<= 0)
    {
        // In this case we'll use the insertion operatior on cerr to print an
error message
        cerr<< "Oops, you entered an invalid age!"<<endl;
        exit(1);
    }

    // Otherwise we'll use insertion again on cout to print a result
    cout<< "You entered "<<nAge<< " years old"<<endl;
return0;

}
```

Streamoutput programexample:

Program 3

```
//string output using <<
#include <stdlib.h>
#include <iostream.h>
void main(void)
{
cout<<"Welcome to C++ I/O module!!!"<<endl;
cout<<"Welcome to ";
cout<<"C++ module 18"<<endl;
//endl is end line stream manipulator
//issue a new line character and flushes the output buffer
//output buffer may be flushed by
cout<<flush;
commandsystem("pause");
}
```

Program 4

```
//concatenating <<
#include <stdlib.h>
//for system(), if compiled in some compiler
//such as Visual Studio, no need this stdlib.h
#include <iostream.h>
void main(void)
{
int p = 3, q = 10;
cout<< "Concatenating using << operator.\n"
<<"--------------------------------"<<endl;
cout<< "70 minus 20 is "<<(70 - 20)<<endl;
cout<< "55 plus 4 is "<<(55 + 4)<<endl;
cout<<p<<" + "<<q<<" = "<<(p+q)<<endl;
system("pause");
}
```

Stream input program example:

Program 5

```
#include <stdlib.h>
#include <iostream.h>
void main(void)
{
int p, q, r;
cout<< "Enter 3 integers separated by space: \n";
cin>>p>>q>>r;
//the >> operator skips whitespace characters such as tabs,
//blank space and newlines. When eof is encountered, zero (false)
//is returned.
cout<<"Sum of the "<<p<<","<<q<<" and "<<r<<" is = "<<(p+q+r)<<endl;
system("pause");
}
```

Program 6

```
//using hex, oct, dec and setbase stream manipulator
#include <stdlib.h>
#include <iostream.h>
#include <iomanip.h>
void main(void)
{
int p;
cout<<"Enter a decimal number:"<<endl;
cin>>p;
```

```
cout<<p<< " in hexadecimal is: "
<<hex<<p<<'\n'
<<dec<<p<<" in octal is: "
<<oct<<p<<'\n'
<<setbase(10) <<p<<" in decimal is: "
<<p<<endl;
cout<<endl;
system("pause");
}
```

Floating-point Precision
Program 7

```
//using precision and setprecision
#include <stdlib.h>
#include <iostream.h>
#include <iomanip.h>
#include <math.h>
void main(void)
{
doubletheroot = sqrt(11.55);
cout<<"Square root of 11.55 with various"<<endl;
cout<<" precisions"<<endl;
cout<<"--------------------------------"<<endl;
cout<<"Using 'precision':"<<endl;
for(intpoinplace=0; poinplace<=8; poinplace++)
{
cout.precision(poinplace);
cout<<theroot<<endl;
}
cout<<"\nUsing 'setprecision':"<<endl;
for(intpoinplace=0; poinplace<=8; poinplace++)
cout<<setprecision(poinplace)<<theroot<<endl;
system("pause");
}
```

Field Width
Program 8

```
//using width member function
#include <iostream.h>
#include <stdlib.h>
void main(void)
{
int p = 6;
char string[20];
cout<<"Using field width with setw() or width()"<<endl;
cout<<"---------------------------------------"<<endl;
cout<<"Enter a line of text:"<<endl;
cin.width(7);
while (cin>>string)
{
cout.width(p++);
cout<<string<<endl;
cin.width(7);
//use ctrl-z followed by return key or ctrl-d to exit
}
system("pause");
}
```

Trailing Zeroes and Decimal Points
Program 9

```
///Using showpoint




//controlling the trailing zeroes and floating points
#include <iostream.h>
#include <iomanip.h>
#include <stdlib.h>
void main(void)
{
cout<<"Before using the ios::showpoint flag\n"
<<"-----------------------------------"<<endl;
cout<<"cout prints 88.88000 as: "<<88.88000
<<"\ncout prints 88.80000 as: "<<88.80000
<<"\ncout prints 88.00000 as: "<<88.00000
<<"\n\nAfter using the ios::showpoint flag\n"
<<"----------------------------------"<<endl;
cout.setf(ios::showpoint);
cout<<"cout prints 88.88000 as: "<<88.88000
<<"\ncout prints 88.80000 as: "<<88.80000
<<"\ncout prints 88.00000 as: "<<88.00000<<endl;
system("pause");
}
```

Manipulators

Program 10

```
//using setw(), setiosflags(), resetiosflags() manipulators
//and setf and unsetf member functions
#include <iostream.h>
#include <iomanip.h>
#include <stdlib.h>
void main(void)
{
long p = 123456789L;
//L - literal data type qualifier for long...
//F - float, UL unsigned integer...
cout<<"The default for 10 fields is right justified:\n"
<<setw(10)<<p
<<"\n\nUsing member function\n"
<<"---------------------\n"
<<"\nUsingsetf() to set ios::left:\n"<<setw(10);
cout.setf(ios::left,ios::adjustfield);
cout<<p<<"\nUsingunsetf() to restore the default:\n";
cout.unsetf(ios::left);
cout<<setw(10)<<p
<<"\n\nUsing parameterized stream manipulators\n"
<<"---------------------------------------\n"
<<"\nUsesetiosflags() to set the ios::left:\n"
<<setw(10)<<setiosflags(ios::left)<<p
<<"\nUsingresetiosflags() to restore the default:\n"
<<setw(10)<<resetiosflags(ios::left)
<<p<<endl;
system("pause");
}
```

Program 11

```
/using setw(), setiosflags(), showpos and internal
```

```
#include <iostream.h>
#include <iomanip.h>
#include <stdlib.h>
void main(void)
{
cout<<setiosflags(ios::internal | ios::showpos)
<<setw(12)<<12345<<endl;
system("pause");




}
```

Exercise

Point out the errors if any

```
   1. // writing on a text file
#include <iostream.h>
#include <fstream.h>
usingnamespacestd;

int main () {
ofstreammyfile ("example.txt");
if (myfile.is_open())
  {
myfile<< "This is a line.\n";
myfile<< "This is another line.\n";

  }
elsecout<< "Unable to open file";
return 0;
}


2.// obtaining file size
#include <iostream.h>
#include <fstream.h>
usingnamespacestd;

int main () {
longbegin,end;
ifstreammyfile ("example.txt");
begin = myfile.tellg();
myfile.seekg (0, ios::end);
end = myfile.tellg();
myfile.close();
cout<< "size is: " << (end-begin) << " bytes.\n";
return 0;
}
3./ basic file operations
#include <iostream.h>
#include <fstream.h>
usingnamespacestd;

int main () {
ofstreammyfile;
myfile.open ("example.txt");
myfile<< "Writing this to a file.\n";
myfile.close();

}
```

ਪਾਠ 7

ਟੈਂਪਲੇਟਸ ਅਤੇ ਐਕਸੈਪਸ਼ਨ ਹੈਂਡਲਿੰਗ
(Templates and Exception Handling)

ਇਸ ਪਾਠ ਵਿਚ ਅਸੀਂ ਹੇਠ ਲਿਖੇ ਵਿਸ਼ਿਆਂ ਬਾਰੇ ਜਾਣਕਾਰੀ ਹਾਸਲ ਕਰਾਂਗੇ——

* ਟੈਂਪਲੇਟਸ
* ਕਲਾਸ ਟੈਂਪਲੇਟਸ
* ਫੰਕਸ਼ਨ ਟੈਂਪਲੇਟਸ
* ਓਵਰਲੋਡਿੰਗ ਆਫ ਟੈਂਪਲੇਟਸ ਫੰਕਸ਼ਨ
* ਐਕਸੈਪਸ਼ਨ ਹੈਂਡਲਿੰਗ
* ਨੇਮਸਪੇਸ, ਕਾਮਟ ਓਪਰੇਟਰ

7.1 ਟੈਂਪਲੇਟਸ

ਇਹ ਇਕ ਅਜਿਹਾ ਵਿਸ਼ਾ ਹੈ ਜਿਸ ਨਾਲ ਅਸੀਂ ਜੈਨਰਿਕ (Generic classes) ਕਲਾਸਾਂ ਅਤੇ ਜੈਨਰਿਕ ਫੰਕਸ਼ਨ ਪਰਿਭਾਸ਼ਿਤ ਕਰ ਸਕਦੇ ਹਾਂ।

ਜੈਨਰਿਕ ਪ੍ਰੋਗਰਾਮਿੰਗ (General Programming) ਉਹ ਪ੍ਰਕਿਰਿਆ ਹੈ ਜਿਸ ਵਿਚ Generic ਡਾਟਾ ਟਾਈਪ ਪੈਰਾਮੀਟਰ ਦੇ ਤੌਰ ਤੇ ਵਰਤ ਸਕਦੇ ਹਾਂ।

ਟੈਂਪਲੇਟਸ ਦੀ ਵਰਤੋਂ ਕਲਾਸ ਜਾਂ ਫੰਕਸ਼ਨ ਦੇ ਗਰੁੱਪ (family) ਬਣਾਉਣ ਲਈ ਕੀਤੀ ਜਾਂਦੀ ਹੈ। ਟੈਂਪਲੇਟ ਨੂੰ ਮੈਕਰੋ (Macro) ਵੀ ਕਿਹਾ ਜਾ ਸਕਦਾ ਹੈ।

ਉਦਾਹਰਣ 7.1 ਪ੍ਰੋਗਰਾਮ (ਟੈਂਪਲੇਟ)

```
template < class T >
class vector
{
   T * V ;       // Type T vector
   int size ;
public :
      vector (int m)
   {

      V = new T [size = m] ;
      fon (int i = 0; i<size, i++)
      V[i] = 0;
   }
   vector (T * a)
   {
      for (int i = 0, i < size ; i ++)
          v[i] = a[i];
   }
T operator * (vector & y)
{
      T sum = 0;
   for (int i = 0 ; i < size ; i++)
   sum + = this -> v[i] + y – v[i] ;
   return sum ;
   }
};
```

7.1.1 ਕਲਾਸ ਟੈਂਪਲੇਟਸ (class Templates)

ਜਿਵੇਂ ਕਿ ਪਹਿਲਾਂ ਦਰਸਾਇਆ ਹੈ ਕਿ ਟੈਂਪਲੇਟਸ ਸਾਨੂੰ ਜੈਨਰਿਕ (Generic) ਕਲਾਸਾਂ ਬਣਾਉਣ ਦੀ ਆਗਿਆ ਦਿੰਦਾ ਹੈ।

ਇਸ ਨੂੰ ਬਣਾਉਣ ਦਾ ਤਰੀਕਾ ਹੇਠਾਂ ਅਨੁਸਾਰ ਹੈ——

```
template < class T >
class classname
{
   //.....

   //class member specification
   //with anonymous type T
```

```
    //wherever appropriate
    //.....
};
```

ਇਕ ਕਲਾਸ (class) ਜੋ ਕਲਾਸ (class) ਟੈਂਪਲੇਟ ਦੀ ਵਰਤੋਂ ਕਰਕੇ ਬਣਾਈ ਜਾਂਦੀ ਹੈ ਉਸ ਨੂੰ ਟੈਂਪਲੇਟ ਕਲਾਸ ਕਹਿੰਦੇ ਹਨ।

7.1.2 ਫੰਕਸ਼ਨ ਟੈਂਪਲੇਟ (Function Templates)

ਇਕ ਕਲਾਸ ਟੈਂਪਲੇਟ (class Template) ਦੀ ਤਰ੍ਹਾਂ ਅਸੀਂ ਫੰਕਸ਼ਨ ਟੈਂਪਲੇਟ ਵੀ ਪਰਿਭਾਸ਼ਿਤ ਕਰ ਸਕਦੇ ਹਾਂ ਜੋ ਕਿ ਅਲੱਗ-ਅਲੱਗ ਆਰਗੂਮੈਂਟ ਟਾਈਪ ਦੇ ਨਾਲ ਫੰਕਸ਼ਨ ਦੇ ਗਰੁੱਪ ਬਣਾਉਂਦੇ ਹਨ।

ਫਾਰਮੈਟ——

```
    template < class T >
    returntype functioname (arguments of type T)
{
    //.....
    //Body of function
    //With type T
    //Wherever appropriate
    //.....
}
```

7.2 ਓਵਰਲੋਡਿੰਗ ਟੈਂਪਲੇਟ ਫੰਕਸ਼ਨ (Overloading of Template Function)

ਇਕ ਟੈਂਪਲੇਟ ਫੰਕਸ਼ਨ ਨੂੰ ਜਾਂ ਤਾਂ ਟੈਂਪਲੇਟ ਫੰਕਸ਼ਨ ਰਾਹੀਂ ਜਾਂ ਸਧਾਰਨ ਫੰਕਸ਼ਨ ਰਾਹੀਂ ਓਵਰਲੋਡ ਕੀਤਾ ਜਾ ਸਕਦਾ ਹੈ। ਇਸ ਸਥਿਤੀ ਵਿਚ ਓਵਰਲੋਡਿੰਗ ਰੈਸੂਲਿਊਸ਼ਨ (Overloading Resolution) ਦੀ ਪੂਰਤੀ ਹੇਠਾਂ ਦਿੱਤੇ ਅਨੁਸਾਰ ਹੁੰਦੀ ਹੈ——

1. ਇਕ ਸਧਾਰਨ (function) ਫੰਕਸ਼ਨ ਨੂੰ call ਕੀਤਾ ਜਾਵੇ ਜਿਸ ਦਾ ਬਿਲਕੁਲ (exact) ਮਿਲਾਣ (Match) ਹੋਵੇ।
2. ਇਕ ਟੈਂਪਲੇਟ ਫੰਕਸ਼ਨ (function) ਨੂੰ call ਕੀਤਾ ਜਾਵੇ ਜੋ ਕਿ ਠੀਕ ਮਿਲਾਣ (exact match) ਤੋਂ ਬਣਿਆ ਹੋਵੇ।
3. ਨਾਰਮਲ (Normal) ਓਵਰਲੋਡਿੰਗ ਰੈਸੋਲਿਊਸ਼ਨ (Resolution) ਦੀ ਵਰਤੋਂ ਕੀਤੀ ਜਾਵੇ।

ਜੇਕਰ ਮਿਲਾਣ ਸਹੀ ਨਾ ਹੋਵੇ ਤਾਂ ਗਲਤੀ (Error) ਦਾ ਮੈਸੇਜ ਆਵੇਗਾ।

ਮਿਸਾਲ :

```
# include <iostream.h>
# include <string.h>
using namespace std,
template <class T>
void display (T  x)
{
    cout << ‘‘Template display :’’ << x << \n’’;
}
void display (int x)     //overloads the generic display( )
{
    cout << ‘‘Explicit display :’’ << x << ‘‘\n’’ ;
}
    int main ( )
    {
        display (100) ;
        display (12.34) ;
        display (‘C’) ;
        return 0;
    }
```

The output of Program is

```
Explicit display : 100
Template display : 12.34
Template display : C
```

7.3 ਐਕਸੈਪਸ਼ਨ ਹੈਂਡਲਿੰਗ (Exception Handling)

ਐਕਸੈਪਸ਼ਨਜ਼ ਕੀ ਹਨ ?

ਐਕਸੈਪਸ਼ਨਜ਼ (Exceptions) ਉਹ ਸਮੱਸਿਆਵਾਂ ਹਨ ਜੋ ਕਿ ਪ੍ਰੋਗਰਾਮ ਨੂੰ ਰਨ (Run) ਕਰਦੇ ਹੋਏ ਆਉਂਦੀਆਂ ਹਨ।

ਬੇਸਿਕਸ ਆਫ ਐਕਸੈਪਸ਼ਨ ਹੈਂਡਲਿੰਗ (Basics of Exception Handling) ਐਕਸੈਪਸ਼ਨਜ਼ (exceptions) ਦੋ ਤਰ੍ਹਾਂ ਦੀਆਂ ਹੁੰਦੀਆਂ ਹਨ :

* ਸਿਨਕਰੋਨਸ ਐਕਸੈਪਸ਼ਨਜ਼ (Synchronous Exception)
* ਅਸਿਨਕਰੋਨਸ ਐਕਸੈਪਸ਼ਨਜ਼ (asynchronous exception)

Synchronous ਐਕਸੈਪਸ਼ਨ ਵਿਚ ‘‘out-of-range index’’ ਅਤੇ ‘‘over-flow’’ ਐਰਰਜ਼ ਆਉਂਦੇ ਹਨ।

asynchronous ਐਕਸੈਪਸ਼ਨਜ਼ ਵਿਚ ਉਹ ਐਰਰਜ਼ ਆਉਂਦੇ ਹਨ ਜੋ ਕਿ ਉਹਨਾਂ ਈਵੈਂਟਸ (Events) ਤੋਂ ਵਾਪਰਦੇ ਹਨ ਜੋ ਕਿ ਪ੍ਰੋਗਰਾਮ ਦੇ ਕੰਟਰੋਲ ਤੋਂ ਪਰ੍ਹੇ (beyond) ਹੁੰਦੇ ਹਨ।

7.4 ਐਕਸੈਪਸ਼ਨ ਕਲਾਸ (try, throw, catch keywords)

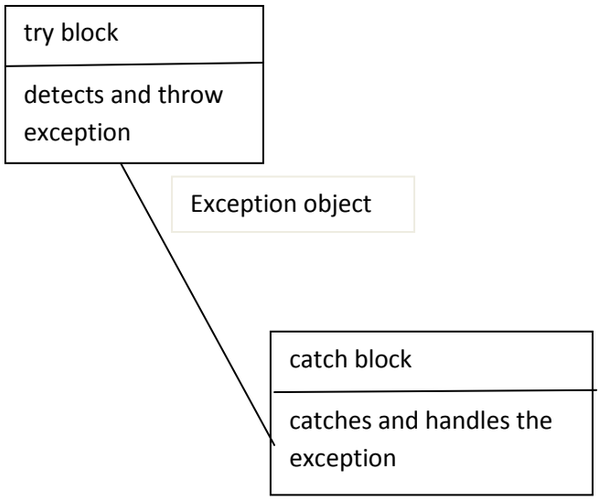
C++ ਦੇ ਵਿਚ ਐਕਸੈਪਸ਼ਨ ਹੈਂਡਲਿੰਗ ਪ੍ਰਕਿਰਿਆ (mechanism) ਤਿੰਨ ਕੀਵਰਡਜ਼ ਤੋਂ ਬਣਿਆ ਹੈ—— try, throw, catch.

ਟਰਾਈ (try)—— try ਸਟੇਟਮੈਂਟਸ ਦੇ ਬਲਾਕ (block) ਵਿਚ ਗਲਤੀਆਂ (errors) ਜਾਂ ਐਕਸੈਪਸ਼ਨ ਦਾ ਪਤਾ ਲਗਾਉਂਦਾ ਹੈ।

throw—— ਜਦੋਂ ਇਕ ਐਕਸੈਪਸ਼ਨ ਦਾ ਪਤਾ ਲੱਗਦਾ ਹੈ ਤਾਂ ਇਸ ਨੂੰ ਦੂਰ ਕਰਨ ਲਈ throw statement ਦੀ ਵਰਤੋਂ ਕਰਕੇ ਭੇਜਿਆ ਜਾਂਦਾ ਹੈ।

catch—— throw ਸਟੇਟਮੈਂਟ ਦੁਆਰਾ ਭੇਜੀ ਗਈ ਸਟੇਟਮੈਂਟ catch ਵਿਚ ਆਉਂਦੀ ਹੈ ਅਤੇ ਇਸ ਨੂੰ ਦੂਰ ਕਰਨ ਲਈ catch ਕੋਸ਼ਿਸ਼ ਕਰਦਾ ਹੈ।

```
.............
.............
try
{
   .............
   throw exception;      // Block of statements which
   .............         // detects and throws on exception
   .............
}
catch (type arg)         // catches exception
{
   .............
.............            // Block of statements that
   .............         // handles the exception
}
.............
.............
```



Try block throwing exceptio

Fig. Try block throwing exception

7.5 ਨੇਮ ਸਪੇਸ ਕਾਨਸੈਪਟ (Name Space Concept)

C++ ਵਿਚ ਅਸੀਂ ਵੇਰੀਏਬਲਜ਼ ਅਲੱਗ-ਅਲੱਗ ਥਾਵਾਂ ਤੇ ਪਰਿਭਾਸ਼ਿਤ (Define) ਕਰ ਸਕਦੇ ਹਾਂ ਜਿਵੇਂ ਕਿ ਕਲਾਸ, ਫੰਕਸ਼ਨ, ਬਲਾਕ (Block) ਆਦਿ। C++ ਵਿਚ ਇਕ ਨਵਾਂ ਕੀਵਰਡ ਪਾਇਆ ਗਿਆ ਹੈ—— ਨੇਮ ਸਪੇਸ ਜੋ ਕਿ ਇਕ ਵੇਰੀਏਬਲ ਨੂੰ ਗਲੋਬਲ ਪਰਿਭਾਸ਼ਿਤ ਕਰਦਾ ਹੈ ਜਿਸਦਾ ਅਰਥ ਹੈ ਕਿ ਨੇਮਸਪੇਸ ਵਿਚ ਪਰਿਭਾਸ਼ਿਤ ਕੀਤਾ ਹੋਇਆ ਵੇਰੀਏਬਲ ਅਸੀਂ ਕਿਤੇ ਵੀ ਪ੍ਰੋਗਰਾਮ ਵਿਚ ਵਰਤ ਸਕਦੇ ਹਾਂ।

Namespace ਦੀ ਸਭ ਤੋਂ ਵਧੀਆ ਮਿਸਾਲ C++ standard library ਹੈ।

Using namespace std;

ਨੇਮਸਪੇਸ ਨੂੰ ਅਸੀਂ ਕਿਸੇ ਕਲਾਸ ਦੀ ਤਰ੍ਹਾਂ ਵੀ ਪਰਿਭਾਸ਼ਿਤ ਕਰ ਸਕਦੇ ਹਾਂ। ਅਸੀਂ ਆਪਣਾ ਨੇਮਸਪੇਸ ਵੀ ਪ੍ਰੋਗਰਾਮ ਵਿਚ ਬਣਾ ਸਕਦੇ ਹਾਂ। ਨੇਮਸਪੇਸ ਬਣਾਉਣ ਦਾ ਤਰੀਕਾ ਹੇਠ ਲਿਖੇ ਅਨੁਸਾਰ ਹੈ——

```
namespace namespace_name
{
    // Declaration of
    // Variables, functions, class etc.
}
```

ਮਿਸਾਲ 7.5

```
# include < iostream>
using namespace std ;
// Defining a namespace
namespace Name 1
{
    double x = 4.56 ;
    int m = 100 ;
    name space  Name 2  // Nesting namespace
{
    double  = 1.23 ;
    }
}
name space


{
    int m = 200 ;
}
int main ( )
{
    cout << ‘‘x = ’’ << Name1 :: x << ‘‘\n’’ ; // x is qualified
    cout << ‘‘m = ’’ << Name1 :: m << ‘‘\n’’ ;
    cout << ‘‘y =’’ << Name1 :: Name2 :: y << ‘‘\n’’ ;
                                                    // T is fully qualified.
```

7.6 ਕਾਸਟ ਓਪਰੇਟਰ (Cast operator)

ਕਾਸਟ ਓਪਰੇਟਰ ਦੀ ਵਰਤੋਂ ਕਿਸੇ ਇੱਕ ਡਾਟਾ ਟਾਈਪ ਦੀ ਕੀਮਤ (value) ਦੂਸਰੀ ਡਾਟਾ ਟਾਈਪ ਵਿਚ ਬਦਲਣ ਲਈ ਕੀਤੀ ਜਾਂਦੀ ਹੈ। ਇਹ ਉਸ ਹਾਲਤ ਵਿਚ ਜ਼ਰੂਰੀ ਹੈ ਜਦੋਂ ਆਟੋਮੈਟਿਕ (automatic) ਬਦਲਾਵ (conversion) ਸੰਭਵ ਨਹੀਂ ਹੁੰਦਾ।

```
double x = doube (m)   // C++ type casting.
```

7.6.1 ਸਟੈਟਿਕ ਕਾਸਟ ਓਪਰੇਟਰ (The Static-Cast operator)

ਸਟੈਟਿਕ ਕਾਸਟ ਓਪਰੇਟਰ ਵੀ ਕਾਸਟ ਅਪਰੇਟਰ ਦੀ ਤਰ੍ਹਾਂ ਡਾਟਾ ਟਾਈਪ ਦੇ ਲਈ ਵਰਤਿਆ ਜਾਂਦਾ ਹੈ। ਇਸ ਦਾ ਇਸਤੇਮਾਲ ਅਸੀਂ ਬੇਸ ਕਲਾਸ ਪੁਆਇੰਟਰ (Base class Pointer) ਤੋਂ ਡਿਰਾਈਵਡ ਕਲਾਸ ਪੁਆਇੰਟਰ (Derived class Pointer) ਵਿਚ ਬਦਲਣ ਲਈ ਕਰ ਸਕਦੇ ਹਾਂ।

```
static_cast (type) (object)
```

ਮਿਸਾਲ——

```
int m = 10 ;
double x = static_cast < double > (m) ;
```

7.6.2 ਕਾਨਸਟੈਂਟ-ਕਾਸਟ ਓਪਰੇਟਰ (The const_Cost operator)

ਕਾਨਸਟੈਂਟ ਕਾਸਟ ਓਪਰੇਟਰ ਦੀ ਵਰਤੋਂ ਕਿਸੇ ਵੀ ਵੇਰੀਏਬਲ ਦੀ ਡਾਟਾ ਟਾਈਪ ਨੂੰ ਬਾਹਰੀ ਤੌਰ ਤੇ override ਕਰਨ ਲਈ ਕੀਤੀ ਜਾਂਦੀ ਹੈ ਜਿਸ ਦਾ ਅਰਥ ਹੈ ਕਿ ਓਪਰੇਟਰ ਡਾਟਾ ਟਾਈਪ ਦੇ ਕਾਨਸਟੈਂਟ (attribute) ਐਟਰੀਬਿਊਟ ਨੂੰ ਬਦਲ ਦਿੰਦਾ ਹੈ।

```
const_cost <type> (object)
```

7.6.3 ਰੀ-ਇੰਟਰਪਰੈਟ ਕਾਸਟ ਓਪਰੇਟਰ (The reinterpret-cast operator)

ਜੇਕਰ ਅਸੀਂ ਪੁਆਇੰਟਰ (Pointre) ਟਾਈਪ ਆਬਜੈਕਟ (Object) ਨੂੰ ਇੰਟੀਜਰ (Integer) ਟਾਈਪ ਵਿਚ ਬਦਲਣਾ ਹੋਵੇ ਤਾਂ ਅਸੀਂ ਰੀ ਇੰਟਰਪਰੈਟ ਕਾਸਟ ਓਪਰੇਟਰ ਦੀ ਵਰਤੋਂ ਕਰਦੇ ਹਾਂ।

```
reinterpret_cast <type> (object)
```

7.6.4 ਡਾਇਨਾਮਿਕ ਕਾਸਟ ਓਪਰੇਟਰ (The dynamic-cast operator)

ਇਸ ਦੀ ਵਰਤੋਂ ਰਨ ਟਾਈਮ (run time) ਤੇ ਇਕ ਡਾਟਾ ਟਾਈਪ ਤੋਂ ਦੂਸਰੀ ਡਾਟਾ ਟਾਈਪ ਵਿਚ ਬਦਲਣ ਲਈ ਕੀਤੀ ਜਾਂਦੀ ਹੈ।

```
dynamic_cast <type> (object)
```

7.7 ਐਕਸੈਪਸ਼ਨ ਹੈਂਡਲਿੰਗ/ਟਰਡੀਸ਼ਨਲ ਐਰਰ ਹੈਂਡਲਿੰਗ (Exception handling vs Traditional Error Handling)

ਪਹਿਲਾਂ ਸਮੇਂ C++ ਕੋਲ ਰਨ ਟਾਈਮ ਐਰਰਜ (error) ਨੂੰ ਨਜਿੱਠਣ ਲਈ ਕੋਈ ਬਿਲਟ ਇਨ ਸੁਵਿਧਾ ਨਹੀਂ ਸੀ। ਇਸ ਕੰਮ ਲਈ ਟਰਡੀਸ਼ਨਲ ਐਰਰ ਹੈਂਡਲਿੰਗ ਮੈਥਡ (Traditional Error Handling Method) ਪ੍ਰਯੋਗ ਕੀਤਾ ਜਾਂਦਾ ਸੀ ਜਿਸ ਵਿਚ ਐਰਰ ਹੈਂਡਲ ਕਰਨ ਲਈ ਹੇਠ ਲਿਖਿਆ ਤਰੀਕਾ ਵਰਤਿਆ ਜਾਂਦਾ ਸੀ——

1. ਪ੍ਰੋਗਰਾਮ ਦਾ ਲੋਜਿਕ ਬਣਾਉਣਾ
2. ਸੂਡੋਕੋਡ ਲਿਖਣਾ (To write Pseudocode)

ਜਿਵੇਂ ਕਿ

2(1) ਕੋਈ ਇਕ ਕੰਮ ਕਰੋ (Perform a Task)

2(2) ਜੇਕਰ ਪ੍ਰੋਸੀਡਿੰਗ (Proceeding) ਵਾਲਾ ਕੰਮ ਜਾਂ ਟਾਸਕ ਸਹੀ ਤਰੀਕੇ ਨਾਲ ਨਾ ਚੱਲੇ ਤਾਂ ਐਰਰ ਪਤਾ ਲਗਾਉਣ ਦੀ ਪ੍ਰਕਿਰਿਆ ਕਰਨ।

2(3) ਅਗਲਾ ਕੰਮ ਕਰੋ ਜੇਕਰ ਪ੍ਰੋਸੀਡਿੰਗ ਵਾਲਾ ਕੰਮ ਜਾਂ ਟਾਸਕ ਸਹੀ ਤਰੀਕੇ ਨਾਲ ਨਾ ਚੱਲੇ ਤਾਂ ਐਰਰ ਪਤਾ ਲਗਾਉਣ ਦੀ ਪ੍ਰਕਿਰਿਆ ਕਰਨ.....

3. ਇਹ ਪ੍ਰੋਗਰਾਮ ਨੂੰ ਪੜ੍ਹਣ, ਸੋਧਣ ਅਤੇ ਸਾਂਭ ਕੇ ਰੱਖਣ ਅਤੇ ਡੀਬੱਗ ਕਰਨ ਵਿਚ ਮੁਸ਼ਕਿਲ ਪੈਦਾ ਕਰਦਾ ਹੈ।
4. ਇਮਪੈਕਟ ਕਾਰਜ ਪ੍ਰਣਾਲੀ (Impact Performance) ਨੂੰ ਪ੍ਰਭਾਵਿਤ ਕਰਦਾ ਹੈ।

ਇਹ ਤਰੀਕਾ (Traditional Error Handling) ਵੱਡੇ ਸਕੇਲ 'ਤੇ ਐਪਲੀਕੇਸ਼ਨ ਬਣਾਉਣ ਲਈ ਬਿਲਕੁਲ ਵੀ ਕੰਮ ਨਹੀਂ ਕਰ ਸਕਦਾ ਅਤੇ ਨਾ ਹੀ ਇਹ ਆਬਜੈਕਟ ਓਰੀਐਂਟਿਡ ਇਨਵਾਇਰਮੈਂਟ (OOP) ਦਾ ਸਮਰਥਨ ਕਰਦਾ ਹੈ।

ਯਾਦ ਰੱਖਣ ਯੋਗ ਗੱਲਾਂ (Points to Remember) :

1. ਜੈਨਰਿਕ ਪ੍ਰੋਗਰਾਮਿੰਗ (Generic Programming) ਉਹ ਪ੍ਰਕਿਰਿਆ ਹੈ ਜਿਸ ਵਿਚ ਜੈਨਰਿਕ (generic) ਡਾਟਾ ਟਾਈਪ ਪੈਰਾਮੀਟਰ ਦੇ ਤੌਰ ਤੇ ਵਰਤ ਸਕਦੇ ਹਾਂ।
2. ਟੈਂਪਲੇਟਸ ਦੀ ਵਰਤੋਂ ਕਲਾਸ ਜਾਂ ਫੰਕਸ਼ਨ ਦੇ ਗਰੁੱਪ (family) ਬਣਾਉਣ ਲਈ ਕੀਤੀ ਜਾਂਦੀ ਹੈ।
3. ਐਕਸੈਪਸ਼ਨਜ਼ ਉਹ ਸਮੱਸਿਆਵਾਂ ਹਨ ਜੋ ਕਿ ਪ੍ਰੋਗਰਾਮ ਨੂੰ ਰਨ (RUN) ਕਰਦੇ ਹੋਏ ਆਉਂਦੀਆਂ ਹਨ।
4. ਨੇਮਸਪੇਸ ਇਕ ਵੇਰੀਏਬਲ ਨੂੰ ਗਲੋਬਲੀ ਪਰਿਭਾਸ਼ਿਤ ਕਰਦਾ ਹੈ।
5. ਕਾਸਟ (cast) ਓਪਰੇਟਰ ਦੀ ਵਰਤੋਂ ਕਿਸੀ ਇਕ ਡਾਟਾ ਟਾਈਪ ਦੀ ਕੀਮਤ (value) ਦੂਸਰੀ ਡਾਟਾ ਟਾਈਪ ਵਿਚ ਬਦਲਣ ਲਈ ਕੀਤੀ ਜਾਂਦੀ ਹੈ।

ਅਭਿਆਸ
(Exercise)

1. ਖਾਲੀ ਥਾਵਾਂ ਭਰੋ——

1. ਟੈਂਪਲੈਟਸ ਸਾਨੂੰ ..... ਕਲਾਸਾਂ ਬਣਾਉਣ ਦੀ ਆਗਿਆ ਦਿੰਦਾ ਹੈ।
2. ਐਕਸੈਪਸ਼ਨਜ਼ ..... ਤਰ੍ਹਾਂ ਦੀਆਂ ਹੁੰਦੀਆਂ ਹਨ।
3. C++ ਵਿਚ ਐਕਸੈਪਸ਼ਨਜ਼ ਹੈਂਡਲਿੰਗ ....., ... ਅਤੇ ..... ਕੀਵਰਡਜ਼ ਤੋਂ ਬਣਿਆ ਹੈ।
4. ..... ਉਪਰੇਟਰ ਦੀ ਵਰਤੋਂ ਅਸੀਂ ਬੇਸ ਕਲਾਸ ਪੁਆਇੰਟਰ ਤੇ ਡਿਰਾਈਵਡ ਕਲਾਸ ਪੁਆਇੰਟਰ ਵਿਚ ਬਦਲਣ ਲਈ ਕਰ ਸਕਦੇ ਹਾਂ।

2. ਸਹੀ ਗਲਤ ਦੱਸੋ——

1. ਨੇਮਸਪੇਸ ਵਿਚ ਪਰਿਭਾਸ਼ਿਤ ਕੀਤਾ ਗਿਆ ਵੇਰੀਏਬਲ ਅਸੀਂ ਪ੍ਰੋਗਰਾਮ ਵਿਚ ਨਹੀਂ ਵਰਤ ਸਕਦੇ।
2. ਟੈਂਪਲੇਟਸ ਦੀ ਵਰਤੋਂ ਕਲਾਸ ਜਾਂ ਫੰਕਸ਼ਨ ਦੇ ਗਰੁੱਪ ਬਣਾਉਣ ਲਈ ਕੀਤੀ ਜਾਂਦੀ ਹੈ।
3. ਨੇਮਸਪੇਸ (namespace) ਦੀ ਸਭ ਤੋਂ ਵਧੀਆ ਮਿਸਾਲ C++ Standard Library ਹੈ।
4. ਇਕ ਟੈਂਪਲੇਟਸ ਫੰਕਸ਼ਨ ਨੂੰ ਓਵਰਲੋਡ ਨਹੀਂ ਕੀਤਾ ਜਾ ਸਕਦਾ।

3. ਛੋਟੇ-ਉੱਤਰਾਂ ਵਾਲੇ ਪ੍ਰਸ਼ਨ——

1. ਜੈਨਰਿਕ ਪ੍ਰੋਗਰਾਮਿੰਗ ਕੀ ਹੈ ?
2. ਐਕਸੈਪਸ਼ਨ ਕੀ ਹਨ ?ਇਸ ਨੂੰ ਕਿਤਨੇ ਭਾਗਾਂ ਵਿਚ ਵੰਡਿਆ ਜਾਂਦਾ ਹੈ ?
3. ਫੰਕਸ਼ਨ ਟੈਂਪਲੇਟ ਦੀ ਪਰਿਭਾਸ਼ਾ ਦਿਉ।
4. ਕਾਸਟ ਉਪਰੇਟਰ ਉੱਤੇ ਸੰਖੇਪ ਨੋਟ ਲਿਖੋ।
5. ਨੇਮਸਪੇਸ ਬਾਰੇ ਜਾਣਕਾਰੀ ਦਿਉ।

4. ਵੱਡੇ ਉੱਤਰਾਂ ਵਾਲੇ ਪ੍ਰਸ਼ਨ——

1. ਨੇਮਸਪੇਸ ਬਣਾਉਣ ਦੀ ਪ੍ਰਕਿਰਿਆ ਦਾ ਵਰਣਨ ਕਰੋ।
2. ਐਕਸੈਪਸ਼ਨ ਹੈਂਡਲਿੰਗ ਪ੍ਰਕਿਰਿਆ ਦੀ ਉਦਾਹਰਣ ਨਾਲ ਵਰਣਨ ਕਰੋ।
3. ਟੈਂਪਲੇਟਸ ਕਿਵੇਂ ਬਣਾਏ ਜਾਂਦੇ ਹਨ ?ਵਰਣਨ ਕਰੋ।

Lab Activity

Templates

Program 1:

```
template <class myType>
myType GetMax (myType a, myType b) {
 return (a>b?a:b);
}
```

Program 2:

```
// function template
#include <iostream.h>
using namespace std;

template <class T>
T GetMax (T a, T b) {
  T result;
  result = (a>b)? a : b;
  return (result);
}

int main () {
  int i=5, j=6, k;
  long l=10, m=5, n;
  k=GetMax<int>(i,j);
  n=GetMax<long>(l,m);
  cout << k << endl;
  cout << n << endl;
  return 0;
}
```

Program 3:

```
// function template II
#include <iostream.h>
using namespace std;

template <class T>
T GetMax (T a, T b) {
  return (a>b?a:b);
}

int main () {
  int i=5, j=6, k;
  long l=10, m=5, n;
```



```
  k=GetMax(i,j);
  n=GetMax(l,m);
  cout << k << endl;
  cout << n << endl;
  return 0;
}
```

Program 4:

```
template <class T>
class mypair {
    T values [2];
  public:
    mypair (T first, T second)
    {
      values[0]=first; values[1]=second;
    }
};
```

Program 5:

```
// class templates
#include <iostream.h>
using namespace std;

template <class T>
class mypair {
    T a, b;
  public:
    mypair (T first, T second)
      {a=first; b=second;}
    T getmax ();
};

template <class T>
T mypair<T>::getmax ()
{
  T retval;
  retval = a>b? a : b;
  return retval;
}

int main () {
  mypair <int> myobject (100, 75);
  cout << myobject.getmax();
  return 0;
}
```

Program 6

```
// template specialization
#include <iostream.h>
using namespace std;

// class template:
template <class T>
class mycontainer {

    T element;
  public:
    mycontainer (T arg) {element=arg;}
    T increase () {return ++element;}
};

// class template specialization:
```

```
template <>
class mycontainer <char> {
    char element;
  public:
    mycontainer (char arg) {element=arg;}
    char uppercase ()
    {
      if ((element>='a')&&(element<='z'))
      element+='A'-'a';
      return element;
    }
};

int main () {
  mycontainer<int> myint (7);
  mycontainer<char> mychar ('j');
  cout << myint.increase() << endl;
  cout << mychar.uppercase() << endl;
  return 0;
}
```

Program 7:

```
// sequence template
#include <iostream.h>
using namespace std;

template <class T, int N>
class mysequence {
    T memblock [N];
  public:
    void setmember (int x, T value);
    T getmember (int x);
};

template <class T, int N>
void mysequence<T,N>::setmember (int x, T value) {
  memblock[x]=value;
}

template <class T, int N>
T mysequence<T,N>::getmember (int x) {
  return memblock[x];
}

int main () {
  mysequence <int,5> myints;
  mysequence <double,5> myfloats;
  myints.setmember (0,100);
  myfloats.setmember (3,3.1416);
  cout << myints.getmember(0) << '\n';
  cout << myfloats.getmember(3) << '\n';
  return 0;
}
```

Exception Handling

Program 8:

```
// exceptions

#include <iostream.h>
using namespace std;

int main () {
  try
  {
    throw 20;
```

```
  }
  catch (int e)
  {
    cout << "An exception occurred. Exception Nr. " << e << endl;
  }
  return 0;
}
```

Program 9:

```
try {
  try {
      // code here
  }
  catch (int n) {
      throw;
  }
}
catch (...) {
  cout << "Exception occurred";
}
```

Program 10:

```
// standard exceptions
#include <iostream.h>
#include <exception>
using namespace std;

class myexception: public exception
{
  virtual const char* what() const throw()
  {
    return "My exception happened";
  }
} myex;

int main () {
  try
  {
    throw myex;
  }
  catch (exception& e)
  {
    cout << e.what() << endl;
  }
  return 0;
}
```

Program 11:

```
// bad_alloc standard exception
#include <iostream.h>
#include <exception>
using namespace std;

int main () {
  try
  {
    int* myarray= new int[1000];
  }
  catch (exception& e)
```

```
  {
    cout << "Standard exception: " << e.what() << endl;
  }
  return 0;
}
```

Namespaces

Program 12:

```
// namespaces
#include <iostream.h>
using namespace std;

namespace first
{
  int var = 5;
}

namespace second
{
  double var = 3.1416;
}

int main () {
  cout << first::var << endl;
  cout << second::var << endl;
  return 0;
}
```

Program 13:

```
// using
#include <iostream.h>
using namespace std;

namespace first
{
  int x = 5;
  int y = 10;
}

namespace second
{
```

```
  double x = 3.1416;
  double y = 2.7183;
}

int main () {
  using first::x;
  using second::y;
  cout << x << endl;
  cout << y << endl;
  cout << first::y << endl;
  cout << second::x << endl;
  return 0;
}
```

Program 14:

```
// using
#include <iostream.h>
using namespace std;

namespace first
{
  int x = 5;
  int y = 10;
}

namespace second
{
  double x = 3.1416;
  double y = 2.7183;
}

int main () {
  using namespace first;
  cout << x << endl;
  cout << y << endl;
  cout << second::x << endl;
  cout << second::y << endl;
  return 0;
}
```

Program 15:

```
// using namespace example
#include <iostream.h>
using namespace std;

namespace first
{
  int x = 5;
}

namespace second
{
  double x = 3.1416;
}

int main () {
  {
    using namespace first;
    cout << x << endl;
  }
  {
    using namespace second;
    cout << x << endl;
  }
  return 0;
}
```