



महाराष्ट्र राज्य तंत्रशिक्षण मंडळ, मुंबई
(स्वायत्त्व) (ISO 9001:2015) (ISO/IEC 27001:2013)

अभियांत्रिकी आणि तंत्रज्ञान पदविका

शिक्षण पुस्तिका

(Learning Material)

**OBJECT ORIENTED
PROGRAMMING USING C++**

(313304)

K Scheme

ऑब्जेक्ट ओरिएंटेड प्रोग्रामिंग
युजिंग सी ++

संगणक अभियांत्रिकी गट
(के स्कीम)

मराठी-इंग्रजी (द्विभाषिक) माध्यम
(अभियांत्रिकी व तंत्रज्ञानातील तृतीय सत्र पदविका)

शिक्षण पुस्तिका
(Learning Material)

ऑब्जेक्ट ओरिएंटेड प्रोग्रामिंग
युजिंग सी ++

**OBJECT ORIENTED
PROGRAMMING USING C++**

(313304)

K Scheme

संगणक अभियांत्रिकी गट
मराठी-इंग्रजी (द्विभाषिक) माध्यम
(अभियांत्रिकी व तंत्रज्ञानातील तृतीय सत्र पदविका)



महाराष्ट्र राज्य तंत्रशिक्षण मंडळ, मुंबई

(स्वायत्व) (ISO 9001:2015) (ISO/IEC 27001:2013)

(313304)

मार्गदर्शक

प्रा. कुकरे विजय नामदेवराव

विभागप्रमुख, संगणक अभियांत्रिकी

प्रा. काळे गोरक्षनाथ भागवतराव

उपप्राचार्य व विभागप्रमुख, संगणक तंत्रज्ञान

लेखक

प्रा. वाकचौरे एस. एल.

(अधिव्याख्याता, संगणक तंत्रज्ञान)

प्रा. घुगे जी.डी.

(अधिव्याख्याता, संगणक तंत्रज्ञान)

प्रा. मुसळे डी. एस.

(अधिव्याख्याता, संगणक तंत्रज्ञान)

प्रा. शिंदे पी.बी.

(अधिव्याख्याता, माहिती तंत्रज्ञान विभाग)



महाराष्ट्र राज्य तंत्र शिक्षण मंडळ

(स्वायत्त) (ISO: ९००१:२०१५) (ISO/IES: २७००१-२०१३)

शासकीय तंत्रनिकेतन इमारत, चौथा मजला, ४९, खेरवाडी, बांद्रा (पूर्व), मुंबई - ४०० ०५१.

दूरध्वनी क्र.: ०२२-६२५४२१७० / १६१

Email : director@msbte.com

Web : www.msbte.org.in



प्रास्ताविक

महाराष्ट्र राज्यातील पदविका स्तरावरील तंत्रशिक्षणामध्ये विद्यार्थ्यांचे रोजगार कौशल्य विकसित करून विद्यार्थ्यांचा सर्वांगीण विकास घडवून आणण्याकरिता महाराष्ट्र राज्य तंत्रशिक्षण मंडळ कटिबद्ध आहे. उद्योगधंद्यातील बदलत्या तंत्रज्ञानाशी संबंधित गरजा लक्षात घेऊन महाराष्ट्र राज्य तंत्र शिक्षण मंडळाकडून पदविका अभ्यासक्रम वेळोवेळी अद्यावत करण्यात येतो. अभियांत्रिकी पदविका अभ्यासक्रम शिकत असतांना संकल्पनात्मक ज्ञान, सुसंगत संदर्भ, प्रश्न विचारणे, विश्वसनीय पुरावे, कारणमीमांसा आणि सुस्पष्ट निकष यांचा वापर करून अर्थाची उकल करण्याची, विश्लेषण व मूल्यमापन करण्याची तसेच तर्काने अनुमान काढण्याची क्षमता म्हणजेच चिकित्सक विचार विद्यार्थ्यांमध्ये अधिक दृढ होतील असा मला विश्वास आहे. जेव्हा विद्यार्थी ज्ञान मिळवण्याच्या माध्यमाशी पूर्णपणे परिचित आणि सोयीस्कर असतात, तेव्हा त्यांच्यासाठी वर्गातील चर्चेत भाग घेणे सोपे होते, संकल्पनात्मक व सैद्धांतिक बाबींचे आकलन परिपूर्ण होते, संज्ञानात्मक क्षमता सुधारते आणि त्यांचा आत्मविश्वास देखील वाढतो या सर्व गोष्टींचा विचार करून मंडळाकडून शैक्षणिक सामुग्रीची निर्मिती करण्यात आलेली आहे. भारत देश हा खेड्यापाड्यातून विकसित झालेला देश असून ग्रामीण भागातील विद्यार्थ्यांना तांत्रिक शिक्षण घेतांना भाषेचा अडसर न येता तांत्रिक बाबींचा आशय समजून घेणे शक्य होईल या दृष्टिकोनातून महाराष्ट्र राज्य तंत्र शिक्षण मंडळाने पदविका स्तरावरील तांत्रिक शिक्षणाकरीता विद्यार्थ्यांना मराठी-इंग्रजी द्विभाषिक माध्यमाचा पर्याय शैक्षणिक वर्ष २०२१-२२ पासून उपलब्ध करून दिलेला आहे.

राष्ट्रीय शैक्षणिक धोरण-२०२० प्रादेशिक भाषेतील शिक्षणास प्रोत्साहन देते, ज्यामुळे विद्यार्थ्यांना तांत्रिक अभ्यासक्रमांसाठी प्रादेशिक भाषांतुन शिक्षणाचे माध्यम निवडता येते. सदर धोरणामुळे प्रादेशिक भाषांमध्ये तांत्रिक सामग्री आणि अभ्यास सामग्रीचा विकास आणि भाषांतर निर्माण करण्याची आवश्यकता आहे. त्यास अनुसरून मंडळाने मराठी-इंग्रजी द्विभाषिक माध्यमाचा पर्याय द्वितीय व तृतीय वर्षाकरिताही उपलब्ध करून देण्यात आला आहे. तसेच त्याकरिताची शैक्षणिक सामग्रीही संबंधीत भागधारकरांना उपलब्ध करून देण्यात येत आहे.

पदविका स्तरावरील तंत्रशिक्षण अधिक दर्जेदार करण्यासाठी महाराष्ट्रातील अनुभवी व तज्ञ अध्यापकांनी व्यवहारिक मराठी भाषा व इंग्रजी भाषेतील तांत्रिक शब्दावली यांचा वापर करून मराठी - इंग्रजी भाषेचा सुवर्णमध्य साधण्याचा प्रयत्न केलेला आहे. मंडळाच्या स्तरावर गठीत सुकाणू समितीमार्फत सदर शैक्षणिक सामुग्रीचा दर्जा, तसेच इतर बाबींची तपासणी करण्यात आलेली आहे. त्यामुळे सदर शैक्षणिक सामुग्री अधिक सम्पन्न झालेली असून विद्यार्थी त्यांच्या व्यक्तिमत्त्वाचा सुसंवादी आणि सर्वांगीण विकास साधतील. परिणामतः विश्वस्तरीय मनुष्यबळाच्या गरजा पूर्ण करण्यात महाराष्ट्र राज्य अग्रेसर राहील व पर्यायाने राष्ट्रनिर्मिती करिता निश्चितच हातभार लागेल, असा मला विश्वास आहे.

अभियांत्रिकी पदविका अभ्यासक्रमातील प्रमुख विषयांची मराठी-इंग्रजी द्विभाषिक शैक्षणिक सामुग्री बनविण्यासाठी अध्यापक व सुकाणू समितीचे सदस्य यांनी दर्शविलेले समर्पण व वचनबद्धता कौतुकास पात्र आहे, या सर्वांचे मी मनः पूर्वक अभिनंदन करतो !


(प्रमोद नाईक)
संचालक

म. रा. तंत्र शिक्षण मंडळ, मुंबई.

अनुक्रमणिका

अनु	युनिटचे नाव	पृष्ठ क्रमांक
1.	प्रिन्सिपल्स ऑफ ऑब्जेक्ट ओरिएंटेड प्रोग्रामिंग (Principles of Object Oriented Programming)	1
2.	फंक्शन आणि कन्स्ट्रक्टर (Functions and Constructors)	17
3.	इंहेरिटेन्सने क्लासेस एक्सटेण्ड्स करणे (Extending classes using Inheritance)	27
4.	पॉइंटर्स आणि पॉलीमॉर्फिझम (Pointers and Polymorphism)	37
5.	फाइल ऑपरेशन (File Operation)	54

युनिट - 1

प्रिन्सिपल्स ऑफ ऑब्जेक्ट ओरिएंटेड प्रोग्रामिंग (Principles of Object Oriented Programming)

विषय निष्पत्ती (Course Outcome):

CO 1- क्लास आणि ऑब्जेक्ट चा वापर करून C ++प्रोग्राम लिहिणे.

घटक निष्पत्ती (Theory Learning Outcome):

1. POP आणि OOP या प्रोग्रामिंग लॅंग्वेज च्या फरक करणे.
2. ऑब्जेक्ट ओरिएंटेड प्रोग्रामिंग लॅंग्वेज चे वेगवेगळे फीचर स्पष्ट करणे.
3. अर्थमॉटिक एक्सप्रेसन सॉल करण्यासाठी प्रोग्राम लिहिणे.
4. वेगवेगळ्या ऑपरेटरचा वापर करून C ++प्रोग्राम लिहिणे.
5. क्लास आणि ऑब्जेक्ट चा वापर करून C ++प्रोग्रॅम लिहिणे.

• C ++चा इतिहास

C++ एक प्रोग्रामिंग भाषा आहे ज्याचा उपयोग सॉफ्टवेअर विकसित करण्यासाठी केला जातो. आपल्याला आवडलेल्या प्रोग्रामिंग भाषा म्हणजे C आणि C++ या दोन्ही भाषांचे संयोजन करण्यात आलेले आहे. या दोन्ही भाषांचा उपयोग कॉम्प्युटर प्रोग्रामिंग, एप्लिकेशन्स, सॉफ्टवेअर विकसित करण्यासाठी, आणि सिस्टम सॉफ्टवेअर विकसित करण्यासाठी केला जातो. C++ ने 1983 मध्ये बीजन स्ट्रौस्ट्रुप यांच्या नेतृत्वाखाली विकसित केला. त्यांच्या विचारानुसार, C++ या भाषेची विक्रमी वृद्धी, विशिष्टता, आणि विस्तार एक नवीन प्रोग्रामिंग परिपेक्षीय केली. C++ ने C भाषेच्या आधारावर निर्माण केले आणि त्याच्या संरचनेची सुधारणा केली.

C++ चा उपयोग विविध क्षेत्रांमध्ये केला जातो, प्रमुखतः सॉफ्टवेअर, खासगी संवाद आणि डेटाबेस विकसकांनी, खासगी खासगी खोड, गेम डेव्हलपर्स आणि सिस्टम प्रोग्रामिंग विकसकांनी. C++ ची विशेषता असे आहे की, ती एका महत्वाच्या भाषापैकी एक आहे ज्याला सर्व प्रकारचे प्रोग्राम लिहू शकता. त्यामुळे, C++ एका अत्यंत लोकप्रिय प्रोग्रामिंग भाषेच्या रूपात विकसित झाली आहे.

C++ चा विकास प्रवास:

1. सुरुवात (1979 - 1983)

- Bjarne Stroustrup यांनी AT&T Bell Labs मध्ये काम करताना C भाषेच्या मर्यादा ओळखल्या.
- त्यांनी C भाषेत Object-Oriented Programming (OOP) संकल्पना समाविष्ट करण्याचा विचार केला.
- त्यांनी C मध्ये क्लासेस (Classes), इनहेरिटन्स (Inheritance) आणि पॉलीमॉर्फिझम (Polymorphism) यांसारख्या संकल्पना जोडल्या.
- हे भाषिक विस्तार "C with Classes" म्हणून ओळखले जात होते.

1.1 प्रोसिजर ओरिएंटेड प्रोग्रामिंग लॅंग्वेज वर्सेस ओरिएंटेड प्रोग्रामिंग (Procedure Oriented Programming (POP) versus Object Oriented Programming (OOP))

खालील टेबलमध्ये प्रोसिजर ओरिएंटेड प्रोग्रामिंग (POP) आणि ऑब्जेक्ट ओरिएंटेड प्रोग्रामिंग (OOP) यांच्या विशेषतेचा तुलना केलेली आहे:

खालील टेबलमध्ये, POP आणि OOP यांच्या प्रमुख वैशिष्ट्ये सारख्या तुलनात केली गेली आहेत.

टेबल 1.1 Comparison of POP and OOP

प्रकार	POP	OOP
संरचना	लीनिअर	अंगता, हायरार्की
कोड संगतता	कमी	उच्च
डेटा आणि क्रियांशांचा संरचन	विभागीत	एकत्रित
धोरण	कार्यक्रम	वस्तू आणि वस्तू निर्माण
उपयोग	कार्यक्रमी	अनुप्रयोगात्मक
विशेषता	कोड साधारण, साफ, पुनरावलोकन-स्वतंत्र	कोड साधारण, स्वच्छ, पुनरावलोकन-स्वतंत्र
भाषांचा उदाहरण	C, Pascal	C++, Java, Python, C#
विकासाची क्षमता	अधिक	अधिक

1.2 फीचर्स ऑफ ऑब्जेक्ट ओरिएंटेड प्रोग्रामिंग (Features of Object Oriented Programming):

C++ भाषा ही एक शक्तिशाली प्रोग्रामिंग भाषा आहे ज्याचा वापर विविध प्रकारच्या सॉफ्टवेअर डेव्हलपमेंटमध्ये केला जातो.

• ऑब्जेक्ट-ओरिएंटेड प्रोग्रामिंग (OOP):

C++ ही OOP ची भाषा आहे ज्यात क्लासेस आणि ऑब्जेक्ट्स वापरून कोडची रचना केली जाते. यात इनहेरिटन्स, पॉलीमॉर्फिझम, एनकॅप्सुलेशन आणि ऍबस्ट्रॅक्शन हे OOP चे प्रमुख तत्त्वे आहेत.

1. क्लासेस आणि ऑब्जेक्ट्स (Classes and Objects):

C++ मध्ये आपण क्लासेस तयार करू शकतो ज्यात डेटा मेंबर्स आणि मेंबर फंक्शन्स असतात. ऑब्जेक्ट हे त्या क्लासचे इंस्टन्स असतात.

2. इनहेरिटन्स (Inheritance):

C++ मध्ये इनहेरिटन्सचे तत्त्व वापरून एक क्लास दुसऱ्या क्लासचे गुणधर्म (प्रॉपर्टीज) आणि वर्तन (बिहेवियर) प्राप्त करू शकतो. यामुळे कोड पुनर्वापर (रियुज) सोपा होतो.

3. पॉलीमॉर्फिझम (Polymorphism):

यामुळे विविध क्लासेसचे ऑब्जेक्ट्स एकाच इंटरफेसद्वारे वापरता येतात. दोन प्रकारचे पॉलीमॉर्फिझम असतात: कंपाईल-टाइम (फंक्शन ओवरलोडिंग) आणि रन-टाइम (व्हर्च्युअल फंक्शन).

4. एनकॅप्सुलेशन (Encapsulation):

डेटा हायडिंग आणि सुरक्षितता सुनिश्चित करण्यासाठी डेटा मेंबर्स आणि फंक्शन्सना एकत्र बांधून ठेवणे.

5. स्टॅटिक (Static) टाइप चेकिंग:

C++ मध्ये, डेटा टाइप्स कंपाईल-टाइमला तपासले जातात, जेणेकरून रन-टाइम एरर्स कमी होतील.

6. लो लेव्हल मेमरी मॅनेजमेंट (Low level Memory Management):

C++ मध्ये, पॉइंटर्सच्या मदतीने मेमरी डायनॅमिकली अॅलॉकेट आणि डिलॉकेट करता येते, ज्यामुळे उच्च कार्यक्षमतेचे प्रोग्राम तयार करता येतात.

7. मल्टीपल इनहेरिटन्स (Multiple Inheritance):

C++ मध्ये एका क्लासला एकापेक्षा जास्त बेस क्लासेसपासून इनहेरिट करता येते, जे C++ ला अधिक लवचिक बनवते.

8. मशीन इंडिपेंडेंट(Machine Independent):

C++ ही भाषा विविध प्रकारच्या हार्डवेअर आणि ऑपरेटिंग सिस्टीमवर चालू शकते. हे काही C++ भाषेचे वैशिष्ट्ये आहेत .

1.2.1 एक्झामपल्स ऑफ ऑब्जेक्ट ओरिएंटेड प्रोग्रामिंग लॉंग्वेज (Examples of Object Oriented languages)

ऑब्जेक्ट ओरिएंटेड प्रोग्रामिंग (OOP) भाषांचा वापर करून सॉफ्टवेअर डेव्हलपमेंट करणे अधिक सोपे आणि कार्यक्षम होते. येथे काही प्रमुख OOP भाषांचे उदाहरणे दिली आहेत, जे मराठीत समजावून सांगितले आहेत:

C++ ही एक व्यापकपणे वापरली जाणारी OOP भाषा आहे. यात क्लासेस, ऑब्जेक्ट्स, इनहेरिटन्स, पॉलीमॉर्फिझम, एनकॅप्सुलेशन आणि ऍबस्ट्रॅक्शन सारखी वैशिष्ट्ये आहेत. C++ ची लोह-लेव्हल मेमरी मॅनेजमेंट क्षमता देखील तिला कार्यक्षम बनवते. हे OOP भाषांचे उदाहरणे आहेत, ज्यांचा वापर करून विविध प्रकारचे सॉफ्टवेअर डेव्हलपमेंट प्रकल्प साकारले जातात.

1.2.2 एप्लीकेशन ऑफ ओओपी (Applications of OOP)-ऑब्जेक्ट ओरिएंटेड प्रोग्रामिंग (OOP) ही प्रोग्रामिंग पॅराडाइम आहे ज्यात डेटा आणि फंक्शन्स एकत्रितपणे ऑब्जेक्ट्समध्ये बांधले जातात. यामुळे सॉफ्टवेअर डेव्हलपमेंट सोपे आणि कार्यक्षम होते. OOP चे विविध अनुप्रयोग (Applications) खालीलप्रमाणे आहेत:

1. सॉफ्टवेअर डेव्हलपमेंट(Software Development):

OOP चा मुख्य अनुप्रयोग म्हणजे सॉफ्टवेअर डेव्हलपमेंट. यात मोठ्या प्रमाणावर वापरले जाते कारण यात कोड पुनर्वापर (रीयूज), मॉड्युलॅरिटी, आणि कोड मॅटेनन्स सोपी होते. उदाहरणार्थ, बँकिंग सॉफ्टवेअर, ऍप्लिकेशन सॉफ्टवेअर, एंटरप्राइज सॉफ्टवेअर.

2. गेम डेव्हलपमेंट (Game development):

गेम डेव्हलपमेंटमध्ये OOP मोठ्या प्रमाणावर वापरली जाते. गेम्समध्ये विविध ऑब्जेक्ट्स जसे की कॅरेक्टर्स, एन्व्हायरमेंट, अॅनिमेशन इत्यादींचे मॉडेलिंग करणे सोपे होते. उदाहरणार्थ, Unity3D आणि Unreal Engine सारखे गेमिंग इंजिन्स.

3. वेब डेव्हलपमेंट (Web development):

OOP आधारित वेब फ्रेमवर्कचा वापर करून डायनॅमिक आणि इंटरॅक्टिव्ह वेब ऍप्लिकेशन्स विकसित करता येतात. उदाहरणार्थ, Django (Python), Ruby on Rails (Ruby), ASP.NET (C#).

4. मोबाइल ऍप्लिकेशन डेव्हलपमेंट (Mobile application development):

iOS आणि Android ऍप्लिकेशन्स डेव्हलप करण्यासाठी **OOP** वापरली जाते. Swift आणि Objective-C वापरून iOS ऍप्स, Kotlin आणि Java वापरून Android ऍप्स विकसित करता येतात.

5. ग्राफिक्स आणि यूजर इंटरफेस (UI) डिझाइन:

GUI बेस्ड ऍप्लिकेशन्स विकसित करण्यासाठी OOP वापरली जाते. उदाहरणार्थ, JavaFX (Java), WPF (C#), Qt (C++).

6. डेटाबेस मॅनेजमेंट सिस्टीम (DBMS):

OOP आधारित डेटाबेस मॅनेजमेंट सिस्टीममध्ये डेटा ऑब्जेक्ट्सच्या स्वरूपात साठवला जातो. उदाहरणार्थ, ObjectDB (Java), db4o (C#).

7. सिस्टम सॉफ्टवेअर (System Software):

ऑपरेटिंग सिस्टीम, ड्रायव्हर्स, आणि एम्बेडेड सिस्टीम विकसित करण्यासाठी **OOP** वापरली जाते. उदाहरणार्थ, Windows (C++), macOS (Objective-C).

8. आर्टिफिशियल इंटेलिजेंस (AI) आणि मशीन लर्निंग (ML):

OOP वापरून AI आणि ML अल्गोरिदम्स आणि ऍप्लिकेशन्स विकसित करता येतात. उदाहरणार्थ, TensorFlow आणि PyTorch (Python).

9. रोबोटिक्स (Robotics):

रोबोटिक्स मध्ये OOP वापरून विविध प्रकारचे रोबोट्स डिझाइन आणि प्रोग्राम केले जातात. यासाठी C++ आणि Python चा मोठ्या प्रमाणावर वापर केला जातो.

OOP च्या वापरामुळे सॉफ्टवेअर डेव्हलपमेंट प्रक्रिया अधिक सुव्यवस्थित, सुरक्षित, आणि कार्यक्षम होते. यामुळे सॉफ्टवेअरची गुणवत्ता सुधारते आणि त्याचे मॅटेनन्स सोपे होते.

1.3 डेटा टाइप्स (Data types)

C++ मध्ये विविध प्रकारचे डेटा टाइप्स उपलब्ध आहेत जे प्रोग्राममध्ये विविध प्रकारचे मूल्ये साठवण्यासाठी वापरले जातात. या डेटा टाइप्सच्या मदतीने आपण आपला कोड अधिक स्पष्ट आणि कार्यक्षम बनवू शकतो. C++ मधील डेटा टाइप्स मुख्यत्वे चार प्रकारात विभागले गेले आहेत) बेसिक (Primitive) डेटा टाइप्स, डेरिव्हड डेटा .

1. Primitive (Basic) प्रिमिटिव्ह डेटा टाइप्स:

- **इंटीजर Integer Types:** int, short, long, long long, unsigned
- **फ्लोटिंग पॉइंट Floating Point Types:** float, double
- **कॅरेक्टर Character Type:** char
- **बुलियन Boolean Type:** bool

2. Derived डेटा टाइप्स:

- **अरे Array:** समान प्रकाराचे एकत्रित मूल्य साठवण्यासाठी.
- **पॉइंटर Pointer:** मेमरी ॲड्रेस साठवण्यासाठी.
- **रेफरन्स Reference:** व्हेरिएबलचा दुसरा नाव साठवण्यासाठी.
- **फंक्शन Function:** फंक्शन्सच्या परिभाषा आणि घोषणांसाठी.

3. User-Defined डेटा टाइप्स:

- **स्ट्रक्चर Structure:** विविध प्रकारांचे मूल्य एका गटात साठवण्यासाठी.
- **युनियन Union:** विविध प्रकारांचे मूल्य एका गटात साठवण्यासाठी, परंतु एका वेळेस एकच मूल्य साठवण्यासाठी.
- **इनम Enum:** संख्यात्मक स्थिरांकांची एक श्रेणी साठवण्यासाठी.
- **क्लास Class:** डेटा आणि फंक्शन्सना एकत्रितपणे साठवण्यासाठी.

4. व्हॉइड Void डेटा टाइप:

व्हॉइड void: कोणताही डेटा साठवत नाही, मुख्यत्वे फंक्शन परिभाषा आणि पॉइंटरसाठी वापरला जातो. हे C++ मधील काही प्रमुख डेटा टाइप्स आहेत, जे प्रोग्रामिंग करताना विविध प्रकारचे डेटा साठवण्यासाठी आणि प्रक्रिया करण्यासाठी वापरले जातात.

1.3.1 टाईप कॉम्पटॅबिलिटी (Type compatibility)

C++ मध्ये डेटा टाइप कॉम्पॅटिबिलिटी म्हणजे विविध डेटा टाइप्स एकमेकांशी कसे सुसंगत (compatible) आहेत हे समजणे. या संदर्भात, दोन महत्वाचे मुद्दे विचारात घ्यावे लागतात: इम्प्लिसिट (implicit) आणि एक्सप्लिसिट (explicit) कनव्हर्षन (type conversion). खालीलप्रमाणे हे अधिक स्पष्ट केले आहे:

1. इम्प्लिसिट कनव्हर्षन (Implicit Conversion):

इम्प्लिसिट कनव्हर्षन (किंवा कोर्सिव्ह कनव्हर्षन) हे स्वयंचलितपणे (automatically) होते, जिथे एक डेटा टाइप दुसऱ्या प्रकारात कधीही बदलला जातो जेव्हा C++ कंपाइलरला आवश्यक वाटते. उदाहरणार्थ:

- **इंटीजर ते फ्लोटिंग पॉइंट कनव्हर्षन (Integer to float conversion):**

- int a = 10;
- float b = a; // 'a' स्वयंचलितपणे float प्रकारात रूपांतरित होते.

- **शॉर्ट ते इंटीजर कनवर्षण (short to integer conversion):**

- short s = 100;
- int i = s; // 's' स्वयंचलितपणे int प्रकारात रूपांतरित होते.

2. एक्सप्लिसिट कनवर्षण (Explicit Conversion):

एक्सप्लिसिट कनवर्षण (किंवा कास्टिंग) हे त्या ठिकाणी वापरकर्त्याने निर्दिष्ट (specify) करावे लागते, जिथे एक डेटा टाइप दुसऱ्या प्रकारात बदलावा लागतो. यासाठी कास्टिंग ऑपरेटर वापरले जातात.

1.3.2 डिक्लरेशन ऑफ व्हेरिएबल (Declaration of variable)

C++ मध्ये व्हेरिएबल्स डिक्लरेशन (घोषणा) म्हणजे मेमरीमध्ये एक विशिष्ट प्रकाराचे मूल्य साठवण्यासाठी एक जागा आरक्षित करणे आणि त्या जागेला एक नाव (व्हेरिएबल) देणे. येथे व्हेरिएबल्स डिक्लरेशनच्या विविध पद्धती आणि त्यांचे उदाहरणे दिली आहेत.

1. बेसिक (Basic) व्हेरिएबल डिक्लरेशन

अ. इंटीजर (Integer)

int age; // इंटीजर प्रकाराचा व्हेरिएबल घोषित केला

ब. फ्लोटिंग पॉइंट (Floating Point)

float salary; // फ्लोट प्रकाराचा व्हेरिएबल घोषित केला

double distance; // डबल प्रकाराचा व्हेरिएबल घोषित केला

क. कॅरेक्टर (Character)

char grade; // कॅरेक्टर प्रकाराचा व्हेरिएबल घोषित केला

1.3.3 डायनामिक इनिशियलायझेशन ऑफ व्हेरिएबल (Dynamic initialization of variable)

C++ मध्ये डायनॅमिक इनिशियलायझेशन म्हणजे व्हेरिएबल्सची इनिशियलायझेशन रन-टाइममध्ये केली जाते, जेव्हा प्रोग्राम चालू असतो. डायनॅमिक इनिशियलायझेशनसाठी साधारणतः कन्स्ट्रक्टर, फंक्शन्स, आणि कधीकधी स्टँडर्ड टेम्प्लेट लायब्ररी (STL) वापरली जाते. डायनॅमिक इनिशियलायझेशन सामान्यतः **new** ऑपरेटर आणि पॉइंटरचा वापर करून केली जाते.

उदाहरण:

```
#include <iostream>
using namespace std;
int main() {
    int* ptr = new int;           // एक integer प्रकारासाठी मेमरी allocate करते
    *ptr = 5;                     // डायनॅमिकली इनिशियलायझ करते
    cout << "Value: " << *ptr << endl;
    delete ptr;                  // मेमरी डिलोकेट करते
    return 0;
}
```

Output:

```
cout << "Value: " << *ptr << endl;
```

Prints the value stored at the memory location pointed to by ptr, which is 5.

कन्स्ट्रक्टर वापरून डायनॅमिक इनिशियलायझेशन:

कन्स्ट्रक्टर वापरून डायनॅमिक इनिशियलायझेशन करणे विशेषतः ऑब्जेक्ट्ससाठी उपयोगी आहे.

उदाहरण:

```
#include <iostream>
```

```
using namespace std;
class Rectangle {
    int width, height;
public:
    Rectangle(int w, int h) {
        width = w;
        height = h;
    }
    int area() {
        return width * height;
    }
};
int main() {
    Rectangle* rect = new Rectangle(10, 5); // डायनॅमिकली Rectangle ऑब्जेक्ट इनिशियलाइज करते
    cout << "Area: " << rect->area() << endl;
    delete rect; // मेमरी डिलोकेट करते
    return 0;
}
```

Output

Area: 50

1.3.4 रेफरन्स वेरिएबल (Reference variable)

C++ मध्ये, रिफरन्स वेरिएबल (Reference Variable) हा एका वेरिएबलचा दुसरा नाव असतो. रिफरन्स वेरिएबलला सुरुवातीच्या वेरिएबलला समान मेमरी ऍड्रेस असतो, ज्यामुळे ते दोन्ही एकाच डेटा स्थानाकडे निर्देश करतात. रिफरन्स वेरिएबल्स डिक्लेर करण्यासाठी आणि त्यांना इनिशियलाइज करण्यासाठी खालील सिंटॅक्स वापरला जातो:

```
type &reference_variable = existing_variable;
```

1.3.5 टाईप कास्टिंग (Type casting)

C++ मध्ये टाईप कास्टिंग म्हणजे एक डेटा प्रकाराचा दुसऱ्या डेटा प्रकारामध्ये रूपांतरण करण्याची प्रक्रिया. C++ मध्ये टाईप कास्टिंग करण्याचे काही प्रमुख प्रकार आहेत:

1. सी स्टायल कास्ट (C-style Casts)
2. फंक्शन स्टायल कास्ट (Function-style Casts)
3. स्टॅटिक कास्ट (Static Cast)
4. डायनॅमिक कास्ट (Dynamic Cast)
5. कॉन्स्ट कास्ट (Const Cast)
6. रीइंटरप्रेट कास्ट (Reinterpret Cast)

1. सी स्टायल कास्ट (C-style Casts)

C-style casting हा सर्वात साधा आणि पारंपारिक स्वरूपाचा कास्टिंग आहे. यात C च्या **syntax** चा वापर होतो आणि विविध प्रकारचे कास्ट ऑपरेशन्स केले जाऊ शकतात.

```
int a = 10;
```

```
double b = (double)a; // C-style cast
```

2. फंक्शन स्टाईल कास्ट (Function-style Casts)

Function-style casting मध्ये **function** कॉल सारखा **syntax** वापरला जातो आणि हा **C-style casting** सारखाच असतो.

```
int a = 10;
```

```
double b = double(a); // Function-style cast
```

3. स्टॅटिक कास्ट (Static Cast)

Static_cast compile-time टाइप कास्टिंगसाठी वापरले जाते. हे **C-style** कास्टिंगपेक्षा सुरक्षित आहे कारण यामध्ये काही प्रमाणात टाइप चेकिंग होते.

```
int a = 10;
```

```
double b = static_cast<double>(a); // Static cast
```

4. डायनॅमिक कास्ट (Dynamic Cast)

Ddynamic_cast सुरक्षित डाऊनकास्टिंगसाठी वापरले जाते, विशेषतः polymorphic प्रकारांसाठी (म्हणजे virtual functions असलेल्या classes याला runtime type information (RTTI) ची आवश्यकता असते आणि runtime वर टाइप चेक करते).

1.4 स्पेशल ऑपरेटर इन सी ++ (Special Operators in C++)

C++ मध्ये काही विशेष ऑपरेटर आहेत जे मूलभूत अंकगणित आणि तर्कशास्त्रीय ऑपरेशन्स व्यतिरिक्त अतिरिक्त कार्यक्षमता प्रदान करतात. खाली **C++** मधील काही प्रमुख विशेष ऑपरेटर दिले आहेत:

1. स्कोप रेझोल्यूशन ऑपरेटर (::) (**Scope Resolution Operator**)
2. मॅम्बर ऍक्सेस ऑपरेटर (. आणि ->) (**Member Access Operator**)
3. टर्नरी किंवा कंडिशनल ऑपरेटर (?) (**Turnary or Conditional Operator**)
4. **sizeof** ऑपरेटर
5. पॉइंटर ऑपरेटर (आणि &)* (**Pointer Operator**)
6. टाइप कास्ट ऑपरेटर (static_cast, dynamic_cast, const_cast, reinterpret_cast)
7. कॉमा ऑपरेटर (,)
8. नवीन (**new**) आणि डिलीट (**delete**) ऑपरेटर
9. typedef ऑपरेटर
10. बिटवाइज ऑपरेटर (**Bitwise Operator**)

1.4.1 स्कोप रेझोल्यूशन ऑपरेटर (::) (Scope resolution operator)

स्कोप रेझोल्यूशन ऑपरेटर एखाद्या फंक्शन किंवा व्हेरिएबलचा स्कोप ठरवण्यासाठी वापरला जातो, विशेषतः नेमस्पेस, क्लासेस आणि ग्लोबल व्हेरिएबल्समध्ये.

```
#include <iostream>
```

```
int var = 10; // ग्लोबल व्हेरिएबल
```

```
class MyClass {
```

```
public:
```

```
    static int var;
```

```
};
```

```
int MyClass::var = 20; // क्लास व्हेरिएबल
```

```
int main() {
```

```
    int var = 30; // लोकल व्हेरिएबल
```

```

std::cout << " var: " << var << std::endl;
std::cout << " var: " << MyClass::var << std::endl;
std::cout << " var: " << ::var << std::endl;
return 0;
}

```

Output

```

var: 30
var: 20
var: 10

```

1.4.2 मेमरी मॅनेजमेंट ऑपरेटर (Memory management operators)

C ++(C++) मध्ये मेमरी व्यवस्थापन ऑपरेटर वापरून डायनॅमिक मेमरी अलोकेशन आणि डीलोकेशन केली जाते. हे ऑपरेटर खालीलप्रमाणे आहेत:

- **new ऑपरेटर:**

मेमरी अलोकेशन करण्यासाठी new ऑपरेटर वापरला जातो. हा ऑपरेटर ऑब्जेक्ट्स किंवा व्हेरिएबल्स साठी डायनॅमिक मेमरी ब्लॉक निर्माण करतो.

Syntax: pointer_variable = new data_type;

उदाहरण:

```
int* ptr = new int; // एक इंटिजर साठी मेमरी अलोकेशन
```

- **delete ऑपरेटर:**

मेमरी डीलोकेशन करण्यासाठी delete ऑपरेटर वापरला जातो. डायनॅमिकली अलोकेटेड मेमरी ब्लॉक डीलोकेट करण्यासाठी हा ऑपरेटर वापरला जातो.

Syntax: delete pointer_variable;

उदाहरण:

```
delete ptr; // अलोकेटेड मेमरी फ्री करणे.
```

- **new[] ऑपरेटर:**

एकाच वेळी अनेक ऑब्जेक्ट्स साठी मेमरी अलोकेशन करण्यासाठी new[] ऑपरेटर वापरला जातो.

Syntax: pointer_variable = new data_type[size];

उदाहरण:

```
int* arr = new int[10]; // 10 इंटिजर साठी मेमरी अलोकेशन
```

- **delete[] ऑपरेटर:**

new[] ऑपरेटर ने अलोकेट केलेल्या मेमरी ब्लॉकसाठी मेमरी डीलोकेशन करण्यासाठी delete[] ऑपरेटर वापरला जातो.

Syntax: delete[] pointer_variable;

उदाहरण:

```
delete[] arr; // अलोकेटेड मेमरी फ्री करणे
```

1.4.3 मॅन्युपिटलेर (Manipulators)

C ++मध्ये मॅनिप्युलेटर्स हे इनपुट आणि आउटपुट स्ट्रीम्सच्या स्वरूपाचे स्वरूप बदलण्यासाठी वापरले जातात. ते आयओस्ट्रीम लायब्ररीमध्ये उपलब्ध आहेत. काही प्रमुख मॅनिप्युलेटर्स खालीलप्रमाणे आहेत:

1. endl

endl मॅनिप्युलेटर वापरून एक नवीन ओळ (न्यू लाइन) तयार केली जाते आणि आउटपुट बफर फ्लश केला जातो.

उदाहरण: `cout << "Hello" << endl << "World";`

2. setw

setw मॅनिप्युलेटर वापरून फील्डच्या रुंदीचे मूल्य सेट केले जाते यासाठी **#include <iomanip>** हेडर फाइल लागते.

3. setfill

setfill मॅनिप्युलेटर वापरून रिकाम्या जागा भरल्या जातात. यासाठी **#include <iomanip>** हेडर फाइल लागते.

4. setprecision

setprecision मॅनिप्युलेटर वापरून फ्लोटिंग-पॉइंट नंबरची अचूकता (decimal places) सेट केली जाते. यासाठी **#include <iomanip>** हेडर फाइल लागते.

5. hex, oct, dec

hex मॅनिप्युलेटर वापरून नंबर हेक्साडेसिमल (hexadecimal) स्वरूपात प्रिंट केला जातो.

oct मॅनिप्युलेटर वापरून नंबर ऑक्टल (octal) स्वरूपात प्रिंट केला जातो.

dec मॅनिप्युलेटर वापरून नंबर डेसिमल (decimal) स्वरूपात प्रिंट केला जातो.

6. boolalpha आणि noboolalpha

boolalpha मॅनिप्युलेटर वापरून बूलियन (boolean) मूल्ये true आणि false म्हणून प्रिंट केली जातात.

noboolalpha मॅनिप्युलेटर वापरून बूलियन मूल्ये 1 आणि 0 म्हणून प्रिंट केली जातात.

1.5 स्ट्रक्चर्स ऑफ सी ++ प्रोग्राम (Structure of C++ program):

C ++ प्रोग्रामचे संरचना समजून घेणे महत्वाचे आहे कारण हे एक सशक्त आणि लवचिक प्रोग्रामिंग भाषा आहे. C ++ प्रोग्राम साधारणतः खालील घटकांमध्ये विभागला जातो:

1. लायब्ररींचा समावेश (Header Files Inclusion)
2. Namespace डिक्लेरेशन
3. फंक्शन प्रोटोटाइप्स (Function Prototypes)
4. मुख्य फंक्शन (Main Function)
5. फंक्शन्सची व्याख्या (Function Definitions)

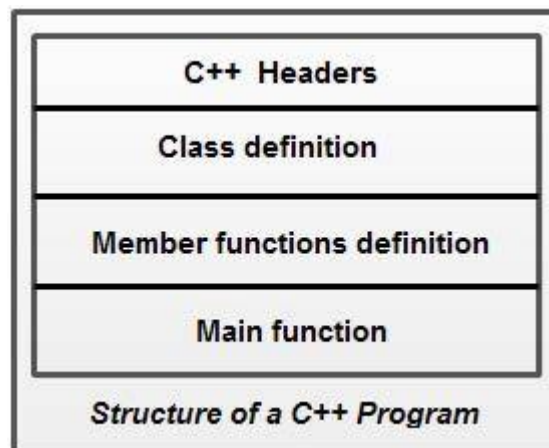


Fig. 1.1: स्ट्रक्चर्स ऑफ C ++ प्रोग्राम

1. लायब्ररीचा समावेश (Header Files Inclusion)

C++मध्ये विविध लायब्ररी फाइल्सचा समावेश करणे आवश्यक आहे जेणेकरून विविध इनबिल्ट फंक्शन्स वापरता येतील. **#include** डिरेक्टिव्ह वापरून हे केले जाते.

2. Namespace डिक्लेरेशन

C++मध्ये **namespace** वापरून वेगवेगळ्या फंक्शन्सचे योग्य वर्गीकरण केले जाते. सर्वसाधारणपणे **std** (स्टँडर्ड) **namespace** वापरला जातो.

3. फंक्शन प्रोटोटाइप्स (Function Prototypes)

फंक्शन्सचे प्रोटोटाइप्स प्रोग्रामच्या सुरुवातीला डिक्लेर केले जातात जेणेकरून मुख्य फंक्शनमध्ये त्यांचा वापर केला जाऊ शकतो.

4. मुख्य फंक्शन (Main Function)

प्रत्येक C ++ प्रोग्राममध्ये **main** नावाचे एक मुख्य फंक्शन असते, जे प्रोग्रामच्या अंमलबजावणीची सुरुवात करते.

5. फंक्शन्सची व्याख्या (Function Definitions)

प्रोटोटाइप्स म्हणून डिक्लेर केलेल्या फंक्शन्सची व्याख्या प्रोग्रामच्या शेवटी केली जाते. या प्रकारे, C ++ प्रोग्रामची संरचना लक्षात घेऊन विविध घटकांमध्ये विभागणी केली जाते, ज्यामुळे प्रोग्राम अधिक सुव्यवस्थित आणि वाचनीय होतो.

1.5.1 बेसिक इनपुट आउटपुट ऑपरेटर अँड फंक्शन इन सी ++ (Basic Input /Output operators and functions in C++)

C ++मध्ये इनपुट आणि आउटपुट ऑपरेटर आणि फंक्शन्स वापरून डेटा इनपुट आणि आउटपुट कसा करायचा हे जाणून घेणे महत्वाचे आहे. **iostream** हेडर फाइल यासाठी वापरली जाते .

इनपुट ऑपरेटर (>>)

- इनपुट ऑपरेटरचा वापर करून डेटा इनपुट केला जातो.
- उदाहरण:

```
int num;
cin >> num;
```

आउटपुट ऑपरेटर (<<)

- आउटपुट ऑपरेटरचा वापर करून डेटा स्क्रीनवर प्रिंट केला जातो.
- उदाहरण:

```
int num = 10;
cout << num;
```

1.5.2 सिम्पल C ++ प्रोग्राम (Simple C++ Program)

एक साधा C ++ प्रोग्राम जो "Hello, World!" प्रिंट करतो आणि त्यानंतर वापरकर्त्याकडून वय आणि नाव विचारतो, नंतर त्यांना प्रिंट करतो, खाली दिलेला आहे:

```
#include <iostream>
#include <string> // स्ट्रिंगसाठी आवश्यक
using namespace std;
int main() {
    // स्क्रीनवर "Hello, World!" प्रिंट करणे
    cout << "Hello, World!" << endl;
    // वय विचारणे आणि इनपुट घेणे
```

```

    int age;
    cout << "Enter your age: ";
    cin >> age;
    // इनपुट बफर साफ करणे
    cin.ignore(); // याची आवश्यकता आहे कारण cin इनपुट नंतर newline characters कन्सम्ट करत नाही
    // नाव विचारणे आणि इनपुट घेणे
    string name;
    cout << "Enter your name: ";
    getline(cin, name);
    // वय आणि नाव प्रिंट करणे
    cout << "Your name is: " << name << " and your age is: " << age << endl;
    return 0;
}

```

Output:

```

Hello, World!
Enter your age: 25
Enter your name: John Doe
Your name is: John Doe and your age is: 25

```

1.6 क्लास अँड ऑब्जेक्ट (Class & Object)

क्लास: क्लास हे डेटा और फंक्शन्सच्या एकत्रित संचाचे प्रतिनिधित्व करते.

ऑब्जेक्ट: क्लासचा एक इंस्टन्स, जो क्लासमध्ये डिक्लेअर केलेल्या घटकांचा वापर करू शकतो.

1.6.1 एक्सेस स्पेसिफायर्स (Access specifiers)

C++मध्ये, एक्सेस स्पेसिफायर्स (**Access Specifiers**) हे डेटा मेंबर्स आणि मेंबर फंक्शन्सच्या एक्सेसिबिलिटी (**accessibility**) नियंत्रित करण्यासाठी वापरले जातात. हे मुख्यतः तीन प्रकारचे असतात:

1. पब्लिक (public)
2. प्रायव्हेट (private)
3. प्रोटेक्टेड (protected)

1. पब्लिक (public)

- **public** स्पेसिफायर अंतर्गत डिक्लेअर केलेले मेंबर्स आणि फंक्शन्स सर्वत्र एक्सेस केले जाऊ शकतात.

2. प्रायव्हेट (private)

- **private** स्पेसिफायर अंतर्गत डिक्लेअर केलेले मेंबर्स आणि फंक्शन्स फक्त त्या क्लासच्या आतूनच एक्सेस केले जाऊ शकतात.

3. प्रोटेक्टेड (protected)

- **protected** स्पेसिफायर अंतर्गत डिक्लेअर केलेले मेंबर्स आणि फंक्शन्स त्या क्लासमध्ये आणि त्याच्या डेरिव्हड (**derived**) क्लासमध्ये एक्सेस केले जाऊ शकतात.

1.6.2 डिफाईनिंग मेंबर फंक्शन इनसाईड क्लास अँड आऊटसाईड क्लास (Defining member functions: Inside class and Outside class)

C++मध्ये, मेंबर फंक्शन्स दोन प्रकारे डिफाइन करता येतात: क्लासच्या आत (Inside class) आणि क्लासच्या बाहेर (Outside class). खाली दोन्ही प्रकारांचे स्पष्टीकरण आणि उदाहरणे दिली आहेत.

1. क्लासच्या आत मेंबर फंक्शन्स डिफाइन करणे (Inside class definition)

मेंबर फंक्शन्स थेट क्लासच्या डिक्लेरेशनमध्ये डिफाइन केली जातात.

उदाहरण:

```
#include <iostream>
#include <string>
using namespace std;
class Student {
private:
    string name;
    int age;
public:
    // Constructor
    Student(string n, int a) {
        name = n;
        age = a;
    }
    // Setter for name (Inside class definition)
    void setName(string n) {
        name = n;
    }
    // Getter for name (Inside class definition)
    string getName() {
        return name;
    }
    // Setter for age (Inside class definition)
    void setAge(int a) {
        age = a;
    }
    // Getter for age (Inside class definition)
    int getAge() {
        return age;
    }
    // Display function (Inside class definition)
    void displayInfo() {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};

int main() {
    // Student क्लास चा ऑब्जेक्ट तयार करणे
    Student student1("John", 20);
    // सदस्य फंक्शन्स वापरून डेटा ऍक्सेस करणे
    student1.displayInfo();
}
```

```

        // डेटा अपडेट करणे
        student1.setName("Jane");
        student1.setAge(22);
        // अपडेट केलेला डेटा प्रदर्शित करणे
        student1.displayInfo();
        return 0;
    }

```

Output:

Name: John, Age: 20

Name: Jane, Age: 22

2. क्लासच्या बाहेर मेंबर फंक्शन्स डिफाइन करणे (Outside class definition)

मेंबर फंक्शन्स क्लासच्या बाहेर डिफाइन केली जातात. यासाठी फंक्शनचे डिक्लेरेशन क्लासच्या आत आणि डिफिनिशन क्लासच्या बाहेर दिले जाते.

उदाहरण:

```

#include <iostream>
#include <string>
using namespace std;
class Student {
private:
    string name;
    int age;
public:
    // Constructor (Inside class definition)
    Student(string n, int a);
    // Setter for name (Function prototype)
    void setName(string n);
    // Getter for name (Function prototype)
    string getName();
    // Setter for age (Function prototype)
    void setAge(int a);
    // Getter for age (Function prototype)
    int getAge();
    // Display function (Function prototype)
    void displayInfo();
};
// Constructor definition (Outside class definition)
Student::Student(string n, int a) {
    name = n;
    age = a;
}
// Setter for name (Outside class definition)
void Student::setName(string n) {
    name = n;
}

```

```

// Getter for name (Outside class definition)
string Student::getName() {
    return name;
}
// Setter for age (Outside class definition)
void Student::setAge(int a) {
    age = a;
}
// Getter for age (Outside class definition)
int Student::getAge() {
    return age;
}
// Display function (Outside class definition)
void Student::displayInfo() {
    cout << "Name: " << name << ", Age: " << age << endl;
}
int main() {
    // Student क्लास चा ऑब्जेक्ट तयार करणे
    Student student1("John", 20);
    // सदस्य फंक्शन्स वापरून डेटा ऍक्सेस करणे
    student1.displayInfo();
    // डेटा अपडेट करणे
    student1.setName("Jane");
    student1.setAge(22);
    // अपडेट केलेला डेटा प्रदर्शित करणे
    student1.displayInfo();
    return 0;
}

```

Output:

Name: John, Age: 20 Name: Jane, Age: 22

1.6.5 क्रिएटिंग ऑब्जेक्ट (Creating objects)

ऑब्जेक्ट तयार करण्याचे सिंटॅक्स (Syntax)

ClassName objectName; // Creating an object of ClassName

1.6.6 मेमरी अलोकेशन फॉर ऑब्जेक्ट (Memory allocations for objects)

C++ मध्ये ऑब्जेक्ट्ससाठी मेमरी अलोकेशन दोन प्रकारे केली जाऊ शकते: स्टॅटिक मेमरी अलोकेशन आणि डायनॅमिक मेमरी अलोकेशन. C++ मध्ये ऑब्जेक्ट्ससाठी मेमरी अलोकेशन दोन पद्धतींनी केली जाऊ शकते: स्टॅटिक आणि डायनॅमिक. स्टॅटिक मेमरी अलोकेशनमध्ये मेमरी स्वयंचलितपणे व्यवस्थापित केली जाते, तर डायनॅमिक मेमरी अलोकेशनमध्ये मेमरी मॅन्युअली व्यवस्थापित करावी लागते. दोन्ही पद्धतींचा वापर त्यांच्या गरजेनुसार आणि प्रोग्रामच्या आकारानुसार केला जातो.

1. स्टॅटिक मेमरी अलोकेशन (Static Memory Allocation)

स्टॅटिक मेमरी अलोकेशन मध्ये मेमरी अलोकेशन रनटाइमच्या वेळी होते आणि ऑब्जेक्ट्स स्टॅकवर निर्माण होतात. यासाठी कोणतेही विशेष कीवर्ड किंवा फंक्शन्स वापरण्याची गरज नसते.

उदाहरण:

```
#include <iostream>
```

```
using namespace std;
class Student {
public:
    string name;
    int age;
    void display() {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};
int main() {
    Student student1; // स्टैटिक अलोकेशन
    student1.name = "John";
    student1.age = 20;
    student1.display(); // Name: John, Age: 20
    return 0;
}
```

output

Name: John, Age: 20

2. डायनॅमिक मेमरी अलोकेशन (Dynamic Memory Allocation)

डायनॅमिक मेमरी अलोकेशन रनटाइमच्या वेळी `new` कीवर्ड वापरून केली जाते आणि ऑब्जेक्ट्स हीपवर निर्माण होतात. यासाठी मेमरी मॅनेजमेंट मॅन्युअली करावी लागते आणि `delete` कीवर्ड वापरून मेमरी फ्री करावी लागते.

उदाहरण:

```
#include <iostream>
using namespace std;
class Student {
public:
    string name;
    int age;
    void display() {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};
int main() {
    // Dynamic Allocation
    Student* student = new Student;
    student->name = "Jane";
    student->age = 22;
    student->display(); // Name: Jane, Age: 22
    // Free Memory
    delete student;
    return 0;
}
```

Output

Name: Jane, Age: 22

संदर्भ:

1. <https://www.w3schools.com/cpp/>.
2. <https://www.javatpoint.com/cpp-tutorial> .
3. <https://www.programiz.com/cpp-programming>.
4. <https://www.programiz.com/cpp-programming/online-compiler/>.
5. https://www.onlinegdb.com/online_cplusplus_compiler.

युनिट- 2

फंक्शन आणि कन्स्ट्रक्टर (Functions and Constructors)

विषय निष्पत्ती (Course outcomes): कन्स्ट्रक्टर वापरून C++ प्रोग्राम विकसित करा.

घटक निष्पत्ती (Theory Learning outcomes):

1. इनलाइन फंक्शन वापरून प्रोग्राम विकसित करा.
2. दिलेल्या समस्येचे निराकरण करण्यासाठी friend function विकसित करा.
3. ऑब्जेक्ट्सच्या अॅरेचा वापर करून C++ प्रोग्राम्स लिहा.
4. कन्स्ट्रक्टर वापरून ऑब्जेक्ट initialize करण्यासाठी C++ प्रोग्राम लिहा.
5. डिस्ट्रक्टर वापरून ऑब्जेक्ट डिलीट करण्यासाठी C++ प्रोग्राम लिहा.

2.1.1 इनलाइन फंक्शन (Inline function):

इनलाइन फंक्शन असे आहे ज्यासाठी कंपाइलर मेमरीमध्ये सूचनांचा वेगळा संच तयार करण्याऐवजी फंक्शन डेफिनिशनमधील कोड थेट कॉलिंग फंक्शनच्या कोडमध्ये कॉपी करतो. इनलाइन फंक्शन हे फंक्शन कॉल ओव्हरहेड कमी करण्यासाठी कॉल पॉइंटवर विस्तारित करण्याचा प्रयत्न करणारे फंक्शन आहे.

उदाहरण:

```
#include <iostream>
inline int add(int a, int b)
{
    return a + b;
}
int main()
{
    std::cout << "Sum: " << add(5, 3) << std::endl;
    return 0;
}
```

आउटपुट (Output):

Sum: 8

2.1.2 स्टॅटिक डेटा मेम्बर (Static Data Member)

स्टॅटिक डेटा मेम्बर हे स्टॅटिक कीवर्ड वापरून घोषित केले जातात. स्टॅटिक डेटा मेम्बरचे काही विशिष्ट वैशिष्ट्ये आहेत जी खालीलप्रमाणे आहेत. स्टॅटिक डेटा मेम्बरची फक्त एक copy संपूर्ण class साठी तयार केली जाते. आणि त्या class चे कितीही ऑब्जेक्ट्स तयार केल्या तरी सर्व ऑब्जेक्ट्सद्वारे सामायिक(share) केली जाते .

उदाहरण:

```
#include <iostream>
class Example {
    static int count;
public:
    Example() { count++; }
    static int getCount() { return count; }
};
int Example::count = 0;
int main() {
    Example obj1, obj2, obj3;
```

```
std::cout << "Number of objects: " << Example::getCount() << std::endl;
return 0;
}
```

आउटपुट (Output):

Number of objects: 3

2.1.3 स्टॅटिक मेम्बर फंक्शन (Static Member Function):

C++ मध्ये स्टॅटिक मेम्बर फंक्शन क्लासमधील स्टॅटिक मेम्बर फंक्शन हे फंक्शन आहे जे स्टॅटिक म्हणून घोषित केले जाते कारण ज्या फंक्शनला खाली परिभाषित केल्याप्रमाणे काही गुणधर्म प्राप्त होतात:

- स्टॅटिक मेम्बर फंक्शन हे क्लासच्या कोणत्याही ऑब्जेक्टपासून स्वतंत्र असते.
- क्लासचे कोणतेही ऑब्जेक्ट अस्तित्वात नसले तरीही स्टॅटिक मेम्बर फंक्शन कॉल केले जाऊ शकते.
- स्कोप रिझोल्यूशन ऑपरेटरद्वारे क्लासचे नाव वापरून स्टॅटिक मेम्बर फंक्शन देखील ऍक्सेस केले जाऊ शकते.
- स्टॅटिक मेम्बर फंक्शन स्टॅटिक डेटा मेम्बर आणि स्टॅटिक मेम्बर फंक्शन्स क्लासच्या आत किंवा बाहेर ऍक्सेस करू शकते.
- स्टॅटिक मेम्बर फंक्शन्सना क्लासमध्ये स्कोप असतो आणि सध्याच्या ऑब्जेक्ट पॉइंटरमध्ये प्रवेश करू शकत नाही.
- क्लासचे किती ऑब्जेक्ट्स तयार केले आहेत हे निर्धारित करण्यासाठी तुम्ही स्टॅटिक मेम्बर फंक्शन देखील वापरू शकता

उदाहरण:

```
#include <iostream>
class Example {
    static int count;
public:
    Example() { count++; }
    static int getCount() { return count; }
};
int Example::count = 0;
int main() {
    Example obj1, obj2;
    std::cout << "Number of objects: " << Example::getCount() << std::endl;
    return 0;
}
```

आउटपुट (Output):

Number of objects: 2

2.1.4 फ्रेंड फंक्शन (Friend Function):

जर C++ मध्ये फंक्शनला फ्रेंड फंक्शन म्हणून परिभाषित केले तर , फंक्शन वापरून क्लासचा प्रोटेक्टड आणि प्राव्हेट डेटा ऍक्सेस करता येतो. फ्रेंड कीवर्ड वापरल्यामुळे कंपाइलरला कळते की दिलेले फंक्शन एक फ्रेंड फंक्शन आहे.

उदाहरण:

```
#include <iostream>
class ClassB;
class ClassA {
    int dataA;
```

```

public:
    ClassA(int val) : dataA(val) {}
    friend void showData(ClassA&, ClassB&);
};
class ClassB
{
    int dataB;
public:
    ClassB(int val) : dataB(val) {}
    friend void showData(ClassA&, ClassB&);
};
void showData(ClassA& objA, ClassB& objB) {
    std::cout << "ClassA data: " << objA.dataA << std::endl;
    std::cout << "ClassB data: " << objB.dataB << std::endl;
}
int main()
{
    ClassA objA(10);
    ClassB objB(20);
    showData(objA, objB);
    return 0;
}

```

आउटपुट (Output):

ClassA data: 10

ClassB data: 20

2.2.1 अरे ऑफ ऑब्जेक्ट (Array of Objects):

C++ मध्ये, ऑब्जेक्ट्सचा अरे तयार करणे म्हणजे क्लास परिभाषित करणे आणि त्यानंतर अरे तयार करणे ज्यामध्ये प्रत्येक घटक त्या क्लासचे उदाहरण आहे.

उदाहरण:

```

#include <iostream>
#include <string>
// Define a class called Person
class Person {
public:
    string name;
    int age;
    // Constructor to initialize the Person object
    Person(string n, int a) {
        name = n;
        age = a;
    }
    // Method to display Person's details
    void display() {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
}

```

```
};
int main() {
    // Create an array of Person objects
    Person people[] = {
        Person("Alice", 30),
        Person("Bob", 25),
        Person("Charlie", 35)
    };
    // Size of the array
    int size = sizeof(people) / sizeof(people[0]);
    // Access and manipulate elements in the array
    for (int i = 0; i < size; i++) {
        people[i].display();
    }
    return 0;
}
```

आउटपुट (Output):

Name: Alice, Age: 30

Name: Bob, Age: 25

Name: Charlie, Age: 35

2.2.2 ऑब्जेक्ट अंश फंक्शन अर्गुमेंट (Object as Function Argument):

C++ मध्ये, तुम्ही फंक्शन्समध्ये अर्गुमेंट म्हणून ऑब्जेक्ट्स पास करू शकता. हे अनेक प्रकारे केले जाऊ शकते: ऑब्जेक्ट्स व्हॅल्यू नुसार पास करू शकता किंवा रेफरन्स नुसार पास करू शकता.

• ऑब्जेक्ट्स व्हॅल्यू नुसार पास करणे (Pass by Value)

जेव्हा तुम्ही ऑब्जेक्ट्स व्हॅल्यू नुसार पास करता तेव्हा त्या ऑब्जेक्ट्स प्रत तयार केली जाते. ऑब्जेक्ट मोठा असल्यास किंवा कॉपी कन्स्ट्रक्टरकडे लक्षणीय ओव्हरहेड असल्यास हे कमी कार्यक्षम असू शकते.

उदाहरण:

```
#include <iostream>
class Example {
    int value;
public:
    Example(int v) : value(v) {}
    void display() { std::cout << value << std::endl; }
};
void show(Example obj) {
    obj.display();
}
int main() {
    Example ex(5);
    show(ex);
    return 0;
}
```

आउटपुट (Output): 5

• ऑब्जेक्ट्स रेफरन्स नुसार पास करणे (Pass by Reference)

ऑब्जेक्ट्स रेफरन्स नुसार पास करणे याला प्राधान्य दिले जाते कारण ते कॉपी करणे टाळते आणि संफरन्स करण्याची आवश्यकता नसते. ही पद्धत कार्यक्षम आणि सुरक्षित दोन्ही आहे.

उदाहरण:

```
#include <iostream>
class Example {
    int value;
public:
    Example(int v) : value(v) {}
    void display() { std::cout << value << std::endl; }
};
void show(const Example& obj) {
    obj.display();
}
int main() {
    Example ex(5);
    show(ex);
    return 0;
}
```

आउटपुट (Output):5

2.3.1 कन्सेप्ट ऑफ कन्स्ट्रक्टर (Concepts of Constructors):

C++ मधील कन्स्ट्रक्टर हे क्लासचे एक विशेष मेंबर फंक्शन आहे जे त्या क्लासतील ऑब्जेक्ट इन्स्टंट केल्यावर आपोआप कॉल केले जाते. कन्स्ट्रक्टरचा प्राथमिक उद्देश ऑब्जेक्टच्या मेंबर ना योग्य डीफॉल्ट किंवा वापरकर्त्याने प्रदान केलेल्या व्हॅल्यू वर प्रारंभ करणे आहे. कन्स्ट्रक्टरचे नाव क्लाससारखेच असते आणि त्यांचा रिटर्न प्रकार नसतो.

2.3.2 कन्स्ट्रक्टरचे प्रकार (Types of Constructors):

कन्स्ट्रक्टरचे प्रकार खालील प्रमाणे आहेत.

- **डिफॉल्ट कन्स्ट्रक्टर (Default constructor):** एक कन्स्ट्रक्टर जो कोणताही पॅरामीटर घेत नाही.

उदाहरण:

```
#include <iostream>
class Person {
public:
    string name;
    int age;
    // Default constructor
    Person() {
        name = "Unknown";
        age = 0;
        cout << "Default constructor called" << endl;
    }
    void display() const {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};
```

```
int main() {
    Person person1; // Default constructor is called
    person1.display();
    return 0;
}
```

आउटपुट (Output):

Default constructor called

Name: Unknown, Age: 0

- **पॅरामीटराइज्ड कन्स्ट्रक्टर (Parameterized Constructor):** एक कन्स्ट्रक्टर जो एक किंवा अधिक पॅरामीटर घेतो.

उदाहरण:

```
#include <iostream>
class Person {
public:
    string name;
    int age;
    // Parameterized constructor
    Person(string n, int a) {
        name = n;
        age = a;
        cout << "Parameterized constructor called" << endl;
    }
    void display() const {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};
int main() {
    Person person2("Alice", 30); // Parameterized constructor is called
    person2.display();
    return 0;
}
```

आउटपुट (Output):

Parameterized constructor called

Name: Alice, Age: 30

- **कॉपी कन्स्ट्रक्टर (Copy Constructor):** एक कन्स्ट्रक्टर जो विद्यमान ऑब्जेक्टची प्रत म्हणून नवीन ऑब्जेक्ट तयार करतो.

उदाहरण:

```
#include <iostream>
class Person {
public:
    string name;
    int age;
    // Parameterized constructor
    Person(string n, int a) : name(n), age(a) {}
    // Copy constructor
```

```

    Person(const Person &p) {
        name = p.name;
        age = p.age;
        cout << "Copy constructor called" << endl;
    }
    void display() const {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};
int main() {
    Person person2("Alice", 30); // Parameterized constructor is called
    Person person3 = person2;    // Copy constructor is called
    person3.display();
    return 0;
}

```

आउटपुट (Output):

Copy constructor called

Name: Alice, Age: 30

2.4.1 कन्स्ट्रक्टर ओवरलोडिंग (Constructor Overloading):

C++ मध्ये कन्स्ट्रक्टर ओवरलोडिंग तुम्हाला क्लासमध्ये अनेक कन्स्ट्रक्टर प्रत्येक वेगवेगळ्या पॅरामीटर लिस्ट सह तयार करण्याची अनुमती देते, हे तुम्हाला प्रदान केलेल्या वितर्कावर अवलंबून वेगवेगळ्या प्रकारे ऑब्जेक्ट्स तयार करण्यास सक्षम करते.

उदाहरण:

```

#include <iostream>
#include <string>
class Person {
public:
    string name;
    int age;
    // Default constructor
    Person() {
        name = "Unknown";
        age = 0;
        cout << "Default constructor called" << endl;
    }
    // Parameterized constructor
    Person(string n, int a) {
        name = n;
        age = a;
        cout << "Parameterized constructor called" << endl;
    }
    // Another parameterized constructor with only name
    Person(string n) {
        name = n;
        age = 0;
    }
}

```

```

        cout << "Parameterized constructor with name only called" << endl;
    }
    // Method to display Person's details
    void display() const {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};
int main() {
    // Create objects using different constructors
    Person person1;           // Default constructor
    Person person2("Alice", 30); // Parameterized constructor
    Person person3("Bob");      // Parameterized constructor with name only
    // Display details of each person
    person1.display();
    person2.display();
    person3.display();
    return 0;
}

```

आउटपुट (Output):

Default constructor called

Parameterized constructor called

Parameterized constructor with name only called

Name: Unknown, Age: 0

Name: Alice, Age: 30

Name: Bob, Age: 0

2.4.2 कन्स्ट्रक्टर विथ डीफॉल्ट अर्ग्युमेंट (Constructors with Default arguments):

कन्स्ट्रक्टर पैरामीटरसमध्ये डीफॉल्ट वॅल्यू असू शकतात.

उदाहरण:

```

#include <iostream>
class Example {
    int value;
public:
    Example(int v = 0) : value(v) {
        std::cout << "Constructor called with value " << value << std::endl;
    }
    void display() { std::cout << value << std::endl; }
};
int main() {
    Example obj1;
    Example obj2(10);
    return 0;
}

```

Constructor called with value 0

Constructor called with value 10

आउटपुट (Output):

Constructor called with value 0

Constructor called with value 10

2.5 डिस्ट्रक्टर (Destructors):

C++ मधील डिस्ट्रक्टर हे एक विशेष मेंबर फंक्शन आहे जे ऑब्जेक्ट डिस्ट्रॉय झाल्यावर आपोआप कॉल केले जाते. डिस्ट्रक्टरचा प्राथमिक उद्देश म्हणजे मेमरी किंवा फाइल हँडल यांसारखी ऑब्जेक्टने त्याच्या जीवनकाळात मिळवलेली संसाधने सोडणे. डिस्ट्रक्टरचे नाव क्लाससारखेच असते, ज्याचा उपसर्ग टिल्ड (~) असतो आणि ते कोणतेही वितर्क घेत नाहीत किंवा व्हॅल्यू रिटर्न करत नाहीत.

उदाहरण:

```
#include <iostream>
using namespace std;
class Person {
public:
    string name;
    int age;
    // Constructor
    Person(string n, int a) : name(n), age(a) {
        cout << "Constructor called for " << name << endl;
    }
    // Destructor
    ~Person() {
        cout << "Destructor called for " << name << endl;
    }
    // Method to display Person's details
    void display() const {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};
int main() {
    // Creating an object
    Person person1("Alice", 30);
    person1.display();
    {
        // Creating another object inside a block scope
        Person person2("Bob", 25);
        person2.display();
    } // person2 goes out of scope here and destructor is called
    return 0;
} // person1 goes out of scope here and destructor is called
```

आउटपुट (Output):

Constructor called for Alice

Name: Alice, Age: 30

Constructor called for Bob

Name: Bob, Age: 25

Destructor called for Bob

Destructor called for Alice

संदर्भ:

1. <https://www.w3schools.com/cpp/>.
2. <https://www.javatpoint.com/cpp-tutorial> .
3. <https://www.programiz.com/cpp-programming>.
4. <https://www.programiz.com/cpp-programming/online-compiler/>.
5. https://www.onlinegdb.com/online_cplusplus_compiler.

युनिट- 3

इंहेरिटेंसने क्लासेस एक्सटेन्ड करणे (Extending classes using Inheritance)

विषय निष्पत्ती (Course outcomes): C++ मध्ये इंहेरिटेंस इम्पलिमेंट करा.

घटक निष्पत्ती (Theory Learning outcomes):

1. दिलेल्या इंहेरिटेंस चा प्रकार त्याच्या वैशिष्ट्यांवर आधारित स्पष्ट करा.
2. C++ प्रोग्रॅममध्ये दिलेला इंहेरिटेंस प्रकार लागू करा.
3. वर्च्युअल बेस क्लास वापरून C++ प्रोग्राम लिहा.
4. दिलेल्या डिराईव्हड (Derived) क्लास मध्ये कन्स्ट्रक्टर वापरा.

3.1.1 इंहेरिटेंस (Introduction to Inheritance):

इन्हेरिटेंस ही ऑब्जेक्ट-ओरिएंटेड प्रोग्रामिंग (OOP) मधील एक मूलभूत संकल्पना आहे जी नवीन क्लास ला (ज्याला सब क्लास किंवा डिराईव्हड क्लास म्हणतात) विद्यमान क्लास चे गुणधर्म आणि वर्तन (पद्धती) प्राप्त करण्यास अनुमती देते (ज्याला सुपरक्लास किंवा बेस क्लास म्हणतात). हे कोडचा पुनर्वापर, तार्किक रचना आणि सोप्या, अधिक व्यवस्थापित करण्यायोग्य घटकांसह अधिक जटिल प्रणाली तयार करण्याच्या क्षमतेस प्रोत्साहन देते.

• इन्हेरिटेंस चे फायदे (Benefits of Inheritance)

1. **कोड पुनर्वापरयोग्यता:** सामान्य कार्यक्षमता एकदा सुपरक्लासमध्ये लिहिली जाऊ शकते आणि उपवर्गाद्वारे पुन्हा वापरली जाऊ शकते.
2. **तार्किक संरचना:** विविध वर्गांमधील स्पष्ट संबंधांसह तार्किकरित्या कोड आयोजित करण्यात मदत करते.
3. **विस्तारक्षमता:** विद्यमान कोडमध्ये कमीत कमी बदलांसह नवीन कार्यक्षमता जोडली जाऊ शकते.

3.1.2 डिराईव्हड क्लास परिभाषित करणे (Defining a Derived Class):

डिराईव्हड क्लास, ज्याला सबक्लास किंवा चाइल्ड क्लास असेही म्हणतात, हा एक क्लास आहे जो दुसऱ्या क्लासवर आधारित तयार केला जातो, ज्याला बेस क्लास, सुपरक्लास किंवा पॅरेंट क्लास म्हणून ओळखले जाते. डिराईव्हड क्लासला बेस क्लासचे गुणधर्म आणि पद्धतींचा वारसा मिळतो, ज्यामुळे त्याला बेस क्लासची कार्यक्षमता पुन्हा वापरता येते आणि ती वाढवता येते. डिराईव्हड क्लास त्याच्या स्वतःच्या गुणधर्म आणि पद्धती देखील सादर करू शकतो किंवा बेस क्लासमधून वारशाने मिळालेल्या ओव्हरराइड करू शकतो.

Syntax :

```
class BaseClass {
    // Base class members
};
class DerivedClass : access_specifier BaseClass {
    // Derived class members
};
```

उदाहरण:

```
#include <iostream>
// Base class
class Base {
public:
    void display() {
        cout << "Base display" << endl;
    }
protected:
```

```

    int protectedVar;
private:
    int privateVar;
};
// Publicly derived class
class PublicDerived : public Base {
public:
    void show() {
        display();          // Valid, public in Base
        protectedVar = 10;  // Valid, protected in Base
        // privateVar = 20; // Invalid, private in Base
    }
};
// Privately derived class
class PrivateDerived : private Base {
public:
    void show() {
        display();          // Valid, public in Base becomes private here
        protectedVar = 10;  // Valid, protected in Base becomes private here
        // privateVar = 20; // Invalid, private in Base
    }
};
int main() {
    PublicDerived pub;
    pub.display();          // Valid, inherited as public
    // pub.protectedVar = 10; // Invalid, protected in Base
    // pub.privateVar = 20;  // Invalid, private in Base
    PrivateDerived priv;
    // priv.display();       // Invalid, inherited as private
    return 0;
}

```

आउटपुट (Output):

Base display

3.1.3 व्हिजिबिलिटी मोड आणि प्रभाव (Visibility Modes and Effects):

C++ मध्ये, व्हिजिबिलिटी मोड्स (अॅक्सेस स्पेसिफायर म्हणूनही ओळखले जातात) क्लास चे सदस्य (विशेषता आणि पद्धती) कसे ऍक्सेस करता येतील हे ठरवतात. जेव्हा डिराईव्हड क्लास (derived class) बेस क्लास पासून इंहेरिट होतो, तेव्हा व्हिजिबिलिटी मोड डिराईव्हड क्लासतील बेस क्लास सदस्यांच्या प्रवेशयोग्यतेवर परिणाम करतो. C++ मध्ये तीन व्हिजिबिलिटी मोड आहेत.

पब्लिक (Public), प्रोटेक्टेड (Protected) आणि प्राव्हेट (Private).

व्हिजिबिलिटी मोड (Visibility Mode)

• पब्लिक इंहेरिटेन्स (Public Inheritance)

1. बेस क्लासचे मॅम्बर डिराईव्हड क्लासमध्ये त्यांचे ऍक्सेस स्पेसिफायर राखून ठेवतात.
2. बेस क्लासचे पब्लिक मॅम्बर डिराईव्हड क्लासचे पब्लिक मॅम्बर बनतात.
3. बेस क्लासचे प्रोटेक्टेड मॅम्बर डिराईव्हड क्लासचे प्रोटेक्टेड मॅम्बर बनतात.

4. बेस क्लासचे प्राव्हेट मेंबर थेट डिराईव्हडं क्लासमध्ये प्रवेश करण्यायोग्य राहतात.
- **प्रोटेक्टेड इंहेरिटेन्स (Protected Inheritance)**
 1. बेस क्लासचे पब्लिक मेंबर डिराईव्हडं क्लासचे प्रोटेक्टेड मेंबर बनतात.
 2. बेस क्लासचे प्रोटेक्टेड मेंबर डिराईव्हडं क्लासचे प्रोटेक्टेड मेंबर राहतात.
 3. बेस क्लासचे प्राव्हेट मेंबर थेट डिराईव्हडं क्लासमध्ये प्रवेश करण्यायोग्य राहतात.
 - **प्राव्हेट इंहेरिटेन्स (Private Inheritance)**
 1. बेस क्लासचे पब्लिक मेंबर डिराईव्हडं क्लासचे प्राव्हेट मेंबर बनतात.
 2. बेस क्लासचे प्रोटेक्टेड मेंबर डिराईव्हडं क्लासचे प्राव्हेट मेंबर बनतात.
 3. बेस क्लासचे प्राव्हेट मेंबर थेट डिराईव्हडं क्लासमध्ये प्रवेश करण्यायोग्य राहतात.

उदाहरण:

```
#include <iostream>
class Base
{
protected:
    int protectedVar = 10;
public:
    int publicVar = 20;
};
class PublicDerived : public Base
{
public:
    void display() {
        std::cout << "Public Var: " << publicVar << std::endl;
        std::cout << "Protected Var: " << protectedVar << std::endl;
    }
};
class ProtectedDerived : protected Base {
public:
    void display() {
        std::cout << "Protected Var: " << protectedVar << std::endl;
        std::cout << "Public Var: " << publicVar << std::endl;
    }
};
class PrivateDerived : private Base
{
public:
    void display() {
        std::cout << "Protected Var: " << protectedVar << std::endl;
        std::cout << "Public Var: " << publicVar << std::endl;
    }
};
int main() {
    PublicDerived publicObj;
    publicObj.display();
    ProtectedDerived protectedObj;
```

```
protectedObj.display();
PrivateDerived privateObj;
privateObj.display();
return 0;
}
```

आउटपुट (Output):

```
Public Var: 20
Protected Var: 10
Protected Var: 10
Public Var: 20
Protected Var: 10
Public Var: 20
```

3. 2. 1 इंहेरिटेन्सचे प्रकार (Types of Inheritance):

इंहेरिटेन्सचे खालील प्रकार आहेत

- **सिंगल इंहेरिटेन्स (Single Inheritance)**

एक क्लास बेसक्लास पासून इंहेरिट होतो त्याला सिंगल इंहेरिटेन्स म्हणतात.

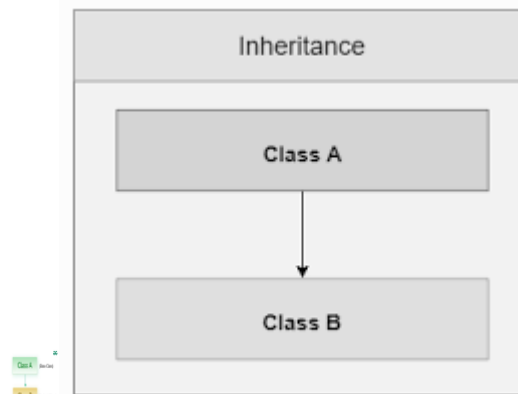


Fig. 3.1: सिंगल इंहेरिटेन्स(Single Inheritance)

उदाहरण:

```
#include <iostream>
class Animal
{
public:
    void eat()
    {
        cout << "Eating" << endl;
    }
};
class Dog : public Animal
{
public:
    void bark() {
        cout << "Barking" << endl;
    }
};
int main() {
```

```

Dog d;
d.eat(); // Inherited from Animal
d.bark(); // Defined in Dog
return 0;
}

```

आउटपुट (Output):

Eating

Barking

• मल्टीलेवल इंहेरिटेन्स (Multilevel Inheritance)

एक क्लास दुसऱ्या डिराईव्हड (derived) क्लास पासून इंहेरिट होतो त्याला मल्टीलेवल इंहेरिटेन्स म्हणतात.

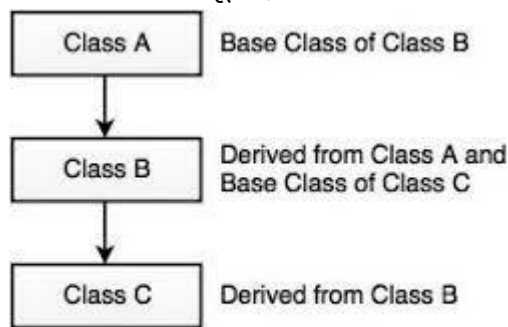


Fig. Multilevel Inheritance

Fig. 3.2: मल्टीलेवल इंहेरिटेन्स (Multilevel Inheritance)

उदाहरण:

```

#include <iostream>
class Animal
{
public:
    void eat() {
        cout << "Eating" << endl;
    }
};
class Mammal : public Animal {
public:
    void walk() {
        cout << "Walking" << endl;
    }
};
class Dog : public Mammal {
public:
    void bark() {
        cout << "Barking" << endl;
    }
};
int main() {
    Dog d;
    d.eat(); // Inherited from Animal
}

```

```

d.walk(); // Inherited from Mammal
d.bark(); // Defined in Dog
return 0;
}

```

आउटपुट (Output):

Eating
Walking
Barking

• मल्टीपल इंहेरिटेन्स (Multiple Inheritance)

एक क्लास एकापेक्षा जास्त बेसक्लास पासून इंहेरिट होतो त्याला मल्टीपल इंहेरिटेन्स म्हणतात.

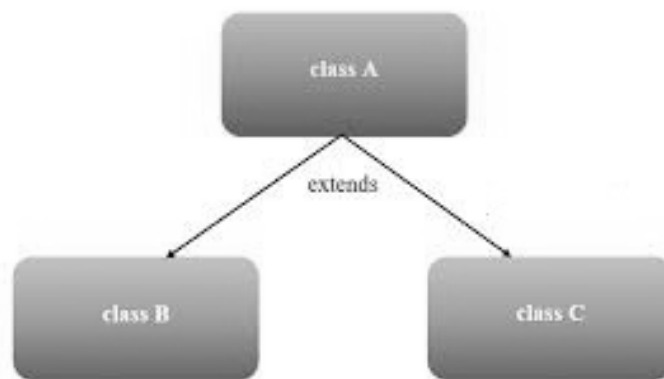


Fig. 3.3: मल्टीपल इंहेरिटेन्स (Multiple Inheritance)

उदाहरण:

```

#include <iostream>
class Animal {
public:
    void eat() {
        cout << "Eating" << endl;
    }
};
class Mammal {
public:
    void walk() {
        cout << "Walking" << endl;
    }
};
class Dog : public Animal, public Mammal {
public:
    void bark() {
        cout << "Barking" << endl;
    }
};
int main() {
    Dog d;
    d.eat(); // Inherited from Animal
    d.walk(); // Inherited from Mammal
}

```

```

d.bark(); // Defined in Dog
return 0;
}

```

आउटपुट (Output):

Eating
Walking
Barking

- **हायरारचीकल इंहेरिटेंस(Hierarchical Inheritance)**

मल्टिपल क्लासेस एकाच बेसक्लास पासून इंहेरिट होतात त्याला हायरारचीकल इंहेरिटेंस म्हणतात.

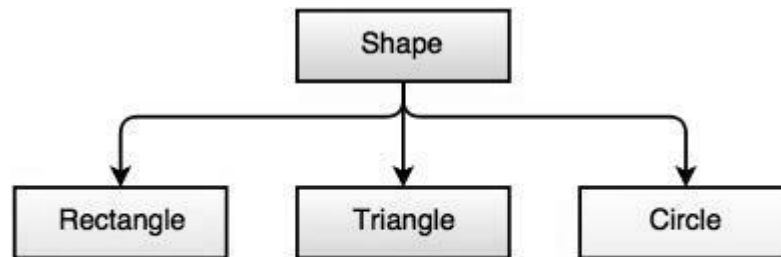


Fig. Hierarchical Inheritance with Class Shape

Fig. 3.4 हायरारचीकल इंहेरिटेंस (Hierarchical Inheritance)

उदाहरण:

```

#include <iostream>
class Animal {
public:
    void eat() {
        cout << "Eating" << endl;
    }
};
class Dog : public Animal {
public:
    void bark() {
        cout << "Barking" << endl;
    }
};
class Cat : public Animal {
public:
    void meow() {
        cout << "Meowing" << endl;
    }
};
int main() {
    Dog dog;
    Cat cat;
    dog.eat(); // Inherited from Animal
    dog.bark(); // Defined in Dog
    cat.eat(); // Inherited from Animal
    cat.meow(); // Defined in Cat
}

```

```
    return 0;
```

```
}
```

आउटपुट (Output):

Eating

Barking

Eating

Meowing

- **हाइब्रिड इंहेरिटेन्स (Hybrid Inheritance)**

दोन किंवा अधिक प्रकारच्या इंहेरिटेन्सचे संयोजन म्हणजे हाइब्रिड इंहेरिटेन्स.

उदाहरण:

```
#include <iostream>
```

```
class Animal {
```

```
public:
```

```
    void eat() {
```

```
        cout << "Eating" << endl;
```

```
    }
```

```
};
```

```
class Mammal : public Animal {
```

```
public:
```

```
    void walk() {
```

```
        cout << "Walking" << endl;
```

```
    }
```

```
};
```

```
class Bird : public Animal {
```

```
public:
```

```
    void fly() {
```

```
        cout << "Flying" << endl;
```

```
    }
```

```
};
```

```
class Bat : public Mammal, public Bird {
```

```
public:
```

```
    void sound() {
```

```
        cout << "Screeching" << endl;
```

```
    }
```

```
};
```

```
int main() {
```

```
    Bat b;
```

```
    b.eat(); // Inherited from Animal
```

```
    b.walk(); // Inherited from Mammal
```

```
    b.fly(); // Inherited from Bird
```

```
    b.sound(); // Defined in Bat
```

```
    return 0;
```

```
}
```

आउटपुट (Output):

Eating

Walking

Flying

Screeching

3.3.1 व्हर्चुअल बेस क्लास (Virtual base class):

व्हर्चुअल बेस क्लासचा वापर मल्टिपल इनहेरिटन्समधील डायमंड समस्या सोडवण्यासाठी केला जातो. जेव्हा एक क्लास दोन क्लास पासून इन्हेरिट होतो जे एकाच बेस क्लास पासून इनहेरिट झालेले आहेत, तेव्हा हे संदिग्धता (ambiguity) निर्माण करते कारण डिराईव्हड क्लास बेस क्लासच्या दोन घटनांना इन्हेरिट करतो. व्हर्चुअल इनहेरिटन्स वापरल्याने बेस क्लासचा फक्त एकच प्रसंग इन्हेरिट झाल्याची खात्री करून या समस्येचे निराकरण करण्यात मदत होते.

उदाहरण:

```
#include <iostream>
class Base {
public:
    void show() {
        cout << "Base class" << endl;
    }
};
class Derived1 : virtual public Base {};
class Derived2 : virtual public Base {};
class FinalDerived : public Derived1, public Derived2 {};
int main() {
    FinalDerived fd;
    fd.show(); // No ambiguity
    return 0;
}
```

आउटपुट (Output):

Base class

3.3.2 अब्स्ट्रॅक्ट क्लास (Abstract class)

C++ मधील अब्स्ट्रॅक्ट क्लास हा एक क्लास आहे ज्याचा इन्स्टंट केला जाऊ शकत नाही आणि सामान्यतः बेस क्लास म्हणून वापरला जातो. यात किमान एक पिव्युर व्हर्चुअल फंक्शन आहे.

उदाहरण:

```
#include <iostream>
class AbstractBase {
public:
    virtual void show() = 0; // Pure virtual function
};
class Derived : public AbstractBase {
public:
    void show() {
        cout << "Derived class implementation" << endl;
    }
};
int main() {
    Derived d;
```

```

    d.show(); // Calls the overridden function in Derived class
    return 0;
}

```

आउटपुट (Output):

Derived class implementation

3. 3. 3 डिराईव्हड वर्गातील कन्स्ट्रक्टर (Constructor in derived class)

जेव्हा डिराईव्हड (derived) क्लासचा इन्स्टंट तयार केला जातो, तेव्हा बेस क्लासचा कन्स्ट्रक्टरला प्रथम कॉल होतो, त्यानंतर डिराईव्हड क्लासचा कन्स्ट्रक्टर कॉल होतो. डिराईव्हड क्लास कन्स्ट्रक्टरमध्ये इनिशिएलायझर लिस्ट वापरून बेस क्लास कन्स्ट्रक्टरला स्पष्टपणे कॉल केले जाऊ शकते.

उदाहरण:

```

#include <iostream>
class Base {
public:
    Base() {
        cout << "Base class constructor" << endl;
    }
};
class Derived : public Base {
public:
    Derived() {
        cout << "Derived class constructor" << endl;
    }
};
int main() {
    Derived d; // Instantiates Derived class
    return 0;
}

```

आउटपुट (Output):

Base class constructor

Derived class constructor

संदर्भ:

1. <https://www.w3schools.com/cpp/>
2. <https://www.javatpoint.com/cpp-tutorial>
3. <https://www.programiz.com/cpp-programming>
4. <https://www.programiz.com/cpp-programming/online-compiler/>
5. https://www.onlinegdb.com/online_cplusplus_compiler

युनिट-4

पॉइंटर्स आणि पॉलीमॉर्फिझम (Pointers and Polymorphism)

विषय निष्पत्ती (Course outcomes): C++ मध्ये पॉलीमॉर्फिझमची अंमलबजावणी करा.

घटक निष्पत्ती (Theory Learning outcomes):

1. पॉइंटरचा वापर करून दिलेले अंकगणितीय ऑपरेशन्स करण्यासाठी C++ प्रोग्राम तयार करा.
2. दिलेली समस्या सोडवण्यासाठी ऑब्जेक्टसाठी पॉइंटर वापरा.
3. दिलेली समस्या सोडवण्यासाठी कंपाइल टाइम पॉलीमॉर्फिझम वापरा.
4. दिलेली समस्या सोडवण्यासाठी रनटाइम पॉलीमॉर्फिझम वापरा.

4.1 पॉइंटरची संकल्पना (Concept of Pointer)

पॉइंटर (पॉइंटर व्हेरिएबल्स) हे विशेष व्हेरिएबल्स आहेत जे मूल्यांऐवजी अॅड्रेसेस (addresses) संग्रहित करण्यासाठी वापरले जातात. आम्ही आमच्या प्रोग्राममध्ये घोषित केलेल्या प्रत्येक व्हेरिएबलचे मेमरीमध्ये संबंधित स्थान असते, ज्याला आम्ही व्हेरिएबलचा मेमरी अॅड्रेस (address) म्हणतो. C++ मध्ये, पॉइंटर हे व्हेरिएबल आहे जे दुसऱ्या व्हेरिएबलचा मेमरी अॅड्रेस ठेवते. पॉइंटर शक्तिशाली आणि लवचिक असतात, C++ भाषेतील पॉइंटर एक व्हेरिएबल आहे, त्याला लोकेटर (Locator) किंवा इंडिकेटर (Indicator) असेही म्हणतात जे मूल्याच्या अॅड्रेसकडे निर्देश करतात. अॅड्रेसचे चिन्ह पॉइंटरद्वारे दर्शविले जाते. डायनॅमिक डेटा स्ट्रक्चर्स तयार आणि सुधारित करण्याव्यतिरिक्त, ते प्रोग्राम्सना कॉल-बाय-रेफरन्सचे अनुकरण करण्याची परवानगी देतात. पॉइंटरच्या मुख्य अनुप्रयोगांपैकी एक म्हणजे अॅरे किंवा इतर डेटा स्ट्रक्चर्सच्या घटकांद्वारे पुनरावृत्ती करणे. पॉइंटर व्हेरिएबल जे व्हेरिएबलच्या समान डेटा प्रकाराला संदर्भित करते ज्या व्हेरिएबलशी तुम्ही व्यवहार करत आहात त्या व्हेरिएबलचा अॅड्रेस त्यावर सेट केलेला असतो (जसे की int किंवा स्ट्रिंग).

C++ मधील पॉइंटर्सचे मुख्य पैलू येथे आहेत:

4.1.1 घोषणा आणि आरंभ (Declaration and Initialization)-

पॉइंटर घोषित करण्यासाठी, तुम्ही तारका asterisk(*) चिन्ह वापरता. उदाहरणार्थ:

Syntax (मांडणी)

datatype *Pointer Name;

int *ptr; // ptr can point to an address which holds int data

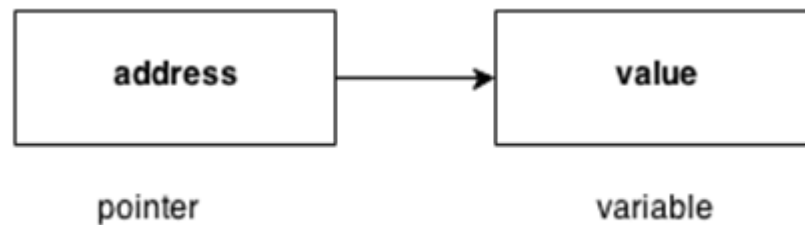


Fig. 4.1 P ointer अॅड्रेस संदर्भ

• पॉइंटर कसा वापरायचा?(How to use a pointer?)

1. पॉइंटर व्हेरिएबल स्थापित करा.
2. व्हेरिएबलच्या पत्त्यावर पॉइंटर नियुक्त करण्यासाठी युनरी ऑपरेटर (&) ची नियुक्ती करणे, जे व्हेरिएबलचा पत्ता देते.

- युनरी ऑपरेटर (*) वापरून, जे त्याच्या युक्तिवादाद्वारे प्रदान केलेल्या पत्त्यावर व्हेरिएबलचे मूल्य देते, एखाद्या पत्त्यामध्ये संग्रहित मूल्यामध्ये प्रवेश करू शकतो. माहिती किती बाइट्समध्ये ठेवली आहे हे डेटा प्रकाराला माहित असल्याने, आम्ही ती एका संदर्भासह संबद्ध करतो. जेव्हा आपण पॉइंटर वाढवतो तेव्हा डेटा प्रकाराचा आकार ज्यामध्ये पॉइंटर पॉइंट जोडला जातो.

- **पॉइंटरचा फायदा (Advantage of pointer)-**

- पॉइंटर कोड कमी करतो आणि कार्यप्रदर्शन सुधारतो, त्याचा वापर स्ट्रिंग्स(Strings), ट्री (Tree)इ. पुनर्प्राप्त करण्यासाठी केला जातो आणि (Array) आरे, स्ट्रक्चर्स (Structure) आणि फंक्शन्ससह (Functions) वापरला जातो.
- आपण पॉइंटर वापरून फंक्शनमधून अनेक व्हॅल्यूज परत करू शकतो.
- हे तुम्हाला संगणकाच्या मेमरीमधील कोणत्याही मेमरी स्थानामध्ये प्रवेश करण्यास सक्षम करते.

- **Symbols used in pointer (पॉइंटरमध्ये वापरलेली चिन्हे)**

Symbol	Name	Description
& (ampersand sign)	Address operator	Determine the address of a variable.
* (asterisk sign)	Indirection operator	Access the value of an address.

Declaring a pointer (सूचक घोषित करणे)

The pointer in C++ language can be declared using * (asterisk symbol).

int *a; //pointer to int

char *c; //pointer to char

Pointer Example (पॉइंटर उदाहरण)

अॅड्रेस (address) आणि मूल्य (value) छापण्यासाठी पॉइंटर्स वापरण्याचे साधे उदाहरण पाहू.

```
#include <iostream>
using namespace std;
int main()
{
int number=30;
int * p;
p=&number;//stores the address of number variable
cout<<"Address of number variable is:"<<&number<<endl;
cout<<"Address of p variable is:"<<p<<endl;
cout<<"Value of p variable is:"<<*p<<endl;
return 0;
}
```

Output:

Address of number variable is:0x7ffccc8724c4

Address of p variable is:0x7ffccc8724c4

Value of p variable is:30

• पॉइंटर अंकगणित (Pointers Arithmetic)

C++ मधील रिथमेटिक ऑपरेटर ऑपरेंड्सवर अंकगणित किंवा गणितीय ऑपरेशन्स करण्यासाठी वापरले जातात. उदाहरणार्थ, '+' (Addition) हा बेरीजसाठी वापरला जातो, '-' (Subtraction) वजाबाकीसाठी वापरला जातो, '*' (Multiplication) गुणाकारासाठी वापरला जातो, इ. सोप्या भाषेत, अंकगणित ऑपरेटर व्हेरिएबल्स आणि डेटावर अंकगणित ऑपरेशन्स करण्यासाठी वापरले जातात; ते ऑपरेटर आणि ऑपरेंडमधील समान संबंधांचे पालन करतात. C प्रोग्रामिंगमध्ये पॉइंटर अंकगणित म्हणजे पॉइंटर्सवर गणितीय ऑपरेशन्स (जसे की बेरीज, वजाबाकी, वाढवणे, कमी करणे) करणे. पॉइंटर्स मेमरी ऍड्रेस (स्मरणिका पत्ता) दर्शवतात, त्यामुळे त्यांच्यावर गणितीय क्रिया केल्याने ऍड्रेस बदलतो.

पॉइंटर ऑपरेशन्स (Pointer Operations)

C मध्ये पॉइंटरवर खालील गणितीय ऑपरेशन्स करू शकतो:

Addition (+) – पॉइंटरला काही संख्या (n) जोडल्यास तो पुढच्या n एलिमेंट्सकडे जातो.

Subtraction (-) – पॉइंटरमधून काही संख्या वजा केल्यास तो मागे n एलिमेंट्सकडे जातो.

Increment (++) – पॉइंटर पुढच्या मेमरी लोकेशनवर जातो.

Decrement (--) – पॉइंटर मागील मेमरी लोकेशनवर जातो.

Subtraction of two pointers – दोन पॉइंटर्सच्या मध्ये किती एलिमेंट्स आहेत हे दर्शवते.

Example:

```
#include <stdio.h>
int main() {
    int arr[] = {10, 20, 30, 40};
    int *ptr = arr; // ptr पहिल्या घटकावर निर्देश करते
    printf("Before: %d\n", *ptr); // 10
    ptr = ptr + 1; // पुढील घटकाकडे जा
    printf("After: %d\n", *ptr); // 20
    return 0;
}
```

Table 4.1-Pointer Operators

Operator	Name of the Operators	Operation	Implementation
+	Addition	Used in calculating the Addition of two operands	x+y
-	Subtraction	Used in calculating Subtraction of two operands	x-y
*	Multiplication	Used in calculating Multiplication of two operands	x*y
/	Division	Used in calculating Division of two operands	x/y
%	Modulus	Used in calculating Remainder after calculation of two operands	x%y

4.2. ऑब्जेक्टसाठी पॉइंटर (Pointer to object)

ऑब्जेक्टसाठी पॉइंटर म्हणजे काय? (What is a Pointer to an object?)

C++ मधील ऑब्जेक्टचा पॉइंटर एक हेरिटेबल आहे ज्यामध्ये ऑब्जेक्टचा मेमरी अड्रेस असतो. पॉइंटर मेमरी आयटमवर अप्रत्यक्ष प्रवेश आणि नियंत्रण प्रदान करतात. जेव्हा आम्हाला ऑब्जेक्टसाठी डायनॅमिकरित्या मेमरी वाटप करणे, लिंक केलेल्या याद्या किंवा ट्री तयार करणे किंवा डुप्लिकेट न बनवता पद्धतींचा संदर्भ देऊन ऑब्जेक्ट पास करणे आवश्यक असते तेव्हा ते विशेषतः उपयुक्त असतात.

ऑब्जेक्ट पॉइंटरचे वर्तन हेरिटेबल पॉइंटरसारखेच असते. परंतु या प्रकरणात, हेरिटेबलसऐवजी ऑब्जेक्टचा अड्रेस ठेवला जातो. जेव्हा क्लास ऑब्जेक्ट मुख्य फंक्शनमध्ये तयार होतो, तेव्हा पॉइंटर हेरिटेबल हेरिटेबल प्रमाणेच घोषित केले जाते. ऑब्जेक्टसाठी पॉइंटर जनरेट करताना पॉइंटरसाठी डेटा प्रकार वापरण्याची शिफारस केलेली नाही. त्याऐवजी, आपण ऑब्जेक्ट पॉइंटरच्या वर्गाचे नाव वापरणे आवश्यक आहे. मुख्य फंक्शनमध्ये पॉइंटर वापरून क्लास सदस्य फंक्शन कॉल करण्यासाठी -> चिन्ह वापरणे आवश्यक आहे.

मांडणी (Syntax):

यात खालील वाक्यरचना आहे **(Declaring a Pointer to an Object):**

ऑब्जेक्टसाठी पॉइंटर pointer घोषित करणे: आम्ही ऑब्जेक्टच्या क्लास (object's class) नाव वापरून आम्ही ऑब्जेक्टला पॉइंटर घोषित करतो, त्यानंतर तारा asterisk(*) आणि पॉइंटरचे नाव.

```
className *pointer name; // Declaration of a pointer to an object
```

• ऑब्जेक्ट्स आणि पॉइंटर्स तयार करणे (Creating Objects and Pointers):

```
class name object name;
```

```
class name * Pointer_name;
```

```
Pointer_name=& object_name
```

Ex. Student S;

```
Student *p;
```

```
p=&s;
```

एखाद्या ऑब्जेक्टच्या सार्वजनिक सदस्यामध्ये प्रवेश करण्यासाठी आपण ऑब्जेक्ट पॉइंटर देखील वापरू शकतो.

आपण डॉट (.) ऑपरेटर वापरून आणि एरोarrow (->) ऑपरेटर वापरून दोन प्रकारे विद्यार्थ्याच्या सदस्य कार्यात प्रवेश करू शकतो.

Ex. Write a program to declare a class BOOK having data members book_id, book_name, book_price. Accept and display the information for one object using pointer to that object.

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

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

```
class book
```

```
{
```

```
private:
```

```
int id;
```

```
char name[10];
```

```
float price;
```

```
public:
```

```
void accept()
```

```
{
```

```
cout<<"Enter Book Id-";
```

```
cin>> id;
```

```
cout<<"Enter Book Name-";
```

```

cin>> name;
cout<<"Enter Book Price-";
cin>> price;
}
void display()
{
cout<<"\n BOOK ID="<<id;
cout<<"\n BOOK NAME="<<name;

cout<< BOOK PRICE="<<price;
}
Void main()
{
book b;
book *p; // pointer declaration
clrscr();
p=&b; //pointer Initialisation
p->accept();
p->display();
getch();
}

```

Output:

```

Enter Book Id – 101
Enter Book Name – OOP
Enter Book Price – 250
BOOK ID = 101
BOOK NAME = OOP
BOOK PRICE = 250

```

- **this pointer (थिस पॉइंटर)**

1. हा पॉइंटर सध्याच्या ऑब्जेक्टचा पॉइंटर आहे ज्याचा सदस्य (डेटा सदस्य किंवा फंक्शन सदस्य) सध्या वापरला जात आहे.
2. हे एक पूर्वनिर्धारित predefined पॉइंटर व्हेरिएबल आहे.
3. प्रत्येक वर्गात जेव्हा सदस्य फंक्शन कॉल केले जाते.हे पॉइंटर आपोआप पास होते.त्या फंक्शनसाठी एक अस्पष्ट युक्तिवाद.
4. सध्याच्या ऑब्जेक्टवर पॉइंटर ऍक्सेस करण्यासाठी this हा कीवर्ड वापरला जातो.
5. या ऑपरेटरसह वर्तमान ऑब्जेक्टच्या सदस्यामध्ये प्रवेश करण्यासाठी आपल्याला Arrow (->) ऑपरेटर वापरावे लागेल.

Example (उदाहरण)

```

#include<iostream.h>
#include<conio.h>
class test
{
private:
char name[10];

```

```

public:
    void getdata()
    {
        cout<<"\n enter the name="";
        cin>> name;
    }
    void putdata()
    {
        cout<<"\n name="<< this->name;
    }
};

void main()
{
    test t1;
    clrscr();
    t1.getdata();
    t1.putdata();
    getch();
}

```

Output:

enter the name= Amrut

name= Amrut

• पॉइंटर टू डिराईव्हड क्लास- (Pointer to derived class)

पॉइंटर टू ऑब्जेक्ट ऑफ बेस क्लास हे डिराईव्हड क्लासच्या (Derived class) ऑब्जेक्टसाठी कंपॅटिबल पॉइंटर टाईप आहेत. म्हणून सिंगल पॉइंटर व्हेरिएबल वेगवेगळ्या क्लासच्या ऑब्जेक्टला पॉइंट करण्यासाठी पॉइंट बनवता येते.

Example (उदाहरण)

```

#include<iostream.h>
#include<conio.h>
class test
{
public:
    void show()
    {
        cout<<"\n This is a base class="";
    }
};

class derived: public test
{
public:
    void show()
    {
        cout<<"This is Derived class."";
    }
};

```

```

void main()
{
    base *A,a;
    derived *B , b;
    clrscr();
    A=&b;
    A->show();
    getch();
}

```

Output:

This is Derived class.

4.3. पॉलिमॉर्फिझम (Introduction of Polymorphism)

पॉलिमॉर्फिझम हे ऑब्जेक्ट-ओरिएंटेड प्रोग्रामिंग (OOP) च्या मुख्य अवधारणांपैकी एक महत्वाचे सिद्धांत आहे. तो साधारण सूटच्या एका जातीच्या ऑब्जेक्टसमध्ये समाविष्ट केल्यास त्यांना विविध श्रेणीच्या ऑब्जेक्ट्स म्हणून वापरू शकता. त्यामुळे सॉफ्टवेअर डिझाईनमध्ये लवचिकता आणि विस्तारशीलतेला समर्थन मिळतो. पॉलिमॉर्फिझम म्हणजे "अनेक रूप", आणि ते तद्वत संघटन करतो की आपल्याला वर्गामधील संबंधित असलेले अनेक वर्ग आहेत. आपल्याला आशय आहे; उत्तराधिकारीत्व आपल्याला इतर वर्गापासून गुणस्थान आणि विधींची संदर्भित वाचवणे देत. पॉलिमॉर्फिझम त्या विध्यांचा वापर करतो की विविध कामे करण्यासाठी त्या विधींचा वापर करतो "पॉलीमॉर्फिझम" हे शब्द अनेक रूपांचा असा अर्थ असतो. सोप्या शब्दांत, आपल्याला पॉलीमॉर्फिझम हे एका संदेशाच्या अनेक रूपांमध्ये प्रदर्शित करण्याची क्षमता म्हणता येऊ शकते. एक वास्तविक उदाहरण पॉलीमॉर्फिझमचा आहे एका व्यक्तीचा, जोच्या एका वेळी विविध गुणधर्म असू शकतात. एक माणूने एका वेळी पिता, एक पती आणि एक कर्मचारी असल्याचं दर्शवू शकतो. म्हणजे एका व्यक्तीने विविध परिस्थितीत विविध वर्तन दर्शवतो. हे पॉलीमॉर्फिझम म्हणतात. पॉलीमॉर्फिझम हे ऑब्जेक्ट-ओरिएंटेड प्रोग्रामिंगच्या महत्वाच्या गुणांपैकी एक मानले जाते.

पॉलिमॉर्फिझमचे प्रकार (Types of Polymorphism):

1. कंपाइल टाइम पॉलिमॉर्फिझम (Compile Time Polymorphism)
2. रन टाइम पॉलिमॉर्फिझम (Run Time Polymorphism)

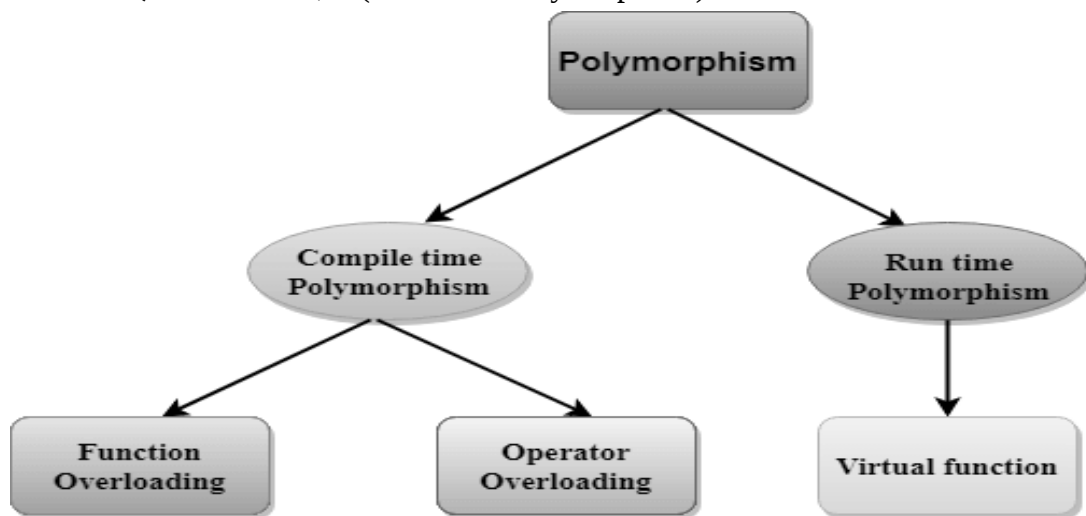


Fig. 4.2: पॉलिमॉर्फिझमचा प्रकार

4.4.1. स्टॅटिक बाइंडिंग/ कंपाइल टाइम (Static Binding/ Compile Time):

'मेथड ओव्हरलोडिंग' म्हणूनही ओळखले जाते, ज्यामुळे एका श्रेणीत एका नावाच्या मल्टिपल मेथड्स असू शकतात, परंतु त्यांच्या पॅरामीटर्स वेगवेगळ्या असल्यानुसार कंपायलरने कोणता मेथड कॉल करावा हे ठरवतो.

फंक्शन ओव्हरलोडिंग (Function Overloading)

कोणत्याही फंक्शनच्या नावाच्या प्रकारात विविध पॅरामीटर्स असताना, त्या फंक्शनला ओव्हरलोडेड म्हणतात, म्हणून हे फंक्शन ओव्हरलोडिंग म्हणतात. फंक्शनला पॅरामीटर्सच्या संख्येत बदल किंवा पॅरामीटर्सच्या प्रकारात बदल करून, हे ओव्हरलोडिंग केले जाऊ शकते. सोप्या शब्दात, हे वस्तूच्या अभिकल्पनेची एक सुविधा आहे ज्याने एका फंक्शनच्या नावाखालील विविध कार्ये दिली जातात. फंक्शन ओव्हरलोडिंगच्या काही नियम आहेत ज्यांना फंक्शनला ओव्हरलोड केल्यानंतर पालन करण्याची गरज आहे.

```
#include <iostream.h>
// Function to add two integers
int add(int a, int b) {
    return a + b;
}
// Function to add two floating point numbers
float add(float a, float b) {
    return a + b;
}
void main() {
    int x = 5, y = 10;
    float p = 5.5, q = 10.5;
    cout << "\nSum of integers: " << add(x, y);
    cout << "\nSum of floats: " << add(p, q);
}
```

Output:

Sum of integers: 15

Sum of floats: 16.0

वरील उदाहरणामध्ये, दोन्ही 'add' फंक्शन्सचे नाव समान आहे परंतु भिन्न पॅरामीटर प्रकार आहेत. पास झालेल्या वितर्काच्या आधारे कोणते फंक्शन कॉल करायचे हे कंपाइलर ठरवतो.

• कन्स्ट्रक्टर ओव्हरलोडिंगची पुनरावृत्ती (Revision of Constructor Overloading):

C++ मध्ये, प्रत्येकाकडे आर्ग्युमेंटची वेगळी यादी असेल तोपर्यंत, आमच्याकडे एकाच नावाच्या वर्गात एकापेक्षा जास्त कन्स्ट्रक्टर असू शकतात. ही संकल्पना कन्स्ट्रक्टर ओव्हरलोडिंग म्हणून ओळखली जाते आणि फंक्शन ओव्हरलोडिंग सारखीच असते.

ओव्हरलोड केलेल्या कन्स्ट्रक्टरचे नाव समान असते (वर्गाचे नेमके नाव) आणि संख्या आणि वितर्काच्या प्रकारानुसार भिन्न.

- पास केलेल्या वितर्काची संख्या आणि प्रकार यावर अवलंबून कन्स्ट्रक्टरला कॉल केला जातो.
- ऑब्जेक्ट तयार करताना, कंपाइलरला कळवण्यासाठी वितर्क पास केले पाहिजेत, कोणत्या कन्स्ट्रक्टरला कॉल करणे आवश्यक आहे.

कन्स्ट्रक्टर ओव्हरलोडिंग प्रदर्शित करण्यासाठी खाली एक C++ प्रोग्राम आहे.

```
#include <iostream.h>
class Box {
private:
    double length;
```

```

double width;
double height;
public:
    // Default constructor
    Box() {
        length = 0;
        width = 0;
        height = 0;
    }
    // Parameterized constructor
    Box(double l, double w, double h) {
        length = l;
        width = w;
        height = h;
    }
    void display() {
        cout << "\nLength: " << length << ", Width: " << width << ", Height: " << height;
    }
};

void main() {
    Box box1; // Calls default constructor
    Box box2(3.0, 4.0, 5.0); // Calls parameterized constructor
    box1.display();
    box2.display();
}

```

Output:

Length:0 , Width: 0, Height: 0

Length:3.0 , Width: 4.0, Height: 5.0

वरील उदाहरणामध्ये, 'बॉक्स' वर्गामध्ये दोन कन्स्ट्रक्टर आहेत: एक डीफॉल्ट कन्स्ट्रक्टर आणि पॅरामीटराइज्ड कन्स्ट्रक्टर.

• ऑपरेटर ओव्हरलोडिंग (Operator Overloading)

ऑपरेटर ओव्हरलोडिंगसाठी नियम (Rule of Operator overloading)

C++ मध्ये ऑपरेटरना डेटा प्रकारासाठी विशेष अर्थ प्रदान करण्याची क्षमता आहे, ही क्षमता ऑपरेटर ओव्हरलोडिंग म्हणून ओळखली जाते. उदाहरणार्थ, दोन स्ट्रिंग्स जोडण्यासाठी स्ट्रिंग क्लाससाठी अॅडिशन ऑपरेटर (+) चा वापर करू शकतो. आम्हाला माहित आहे की या ऑपरेटरचे कार्य दोन ऑपरेंड जोडणे आहे. म्हणून एकच ऑपरेटर '+', पूर्णांक ऑपरेंडमध्ये ठेवल्यावर, त्यांना जोडतो आणि जेव्हा स्ट्रिंग ऑपरेंडमध्ये ठेवतो, तेव्हा त्यांना जोडतो.

ऑपरेटर ओव्हरलोडिंगचे प्रदर्शन करण्यासाठी खाली C++ प्रोग्राम आहे:

• नियम (Rule)

फंक्शनला ओव्हरलोड करताना, आपल्याला वापरायचा 'operator' कीवर्ड वापरावा, ज्याच्यानंतर क्लासचं नाव येईल आणि 'operator' संकेतांचा वापर करावा, जसे की '+'.

• मेम्बर आणि नॉन मेम्बर फंक्शन (Member and Non Member Function)

ऑपरेटर्सला दोन पदांच्या दृष्टीने दोन रीतीने ओव्हरलोड करण्याची संधी आहे: पहिली हे मेम्बर फंक्शनस्वरूपात करणे, दुसरी हे नॉन मेम्बर फंक्शनस्वरूपात. सदस्य फंक्शनस्वरूपात ओव्हरलोड केलेल्या फंक्शनवर कॉल

केलेला ऑब्जेक्ट उपस्थित करतो आणि असदस्य फंक्शनस्वरूपात, डाव्या ऑपरंडसाठी कोणत्याही प्रकारचा उपयोग करू शकता.

- **आवश्यक ऑपरंडसची संख्या (Numbers of Operands)**

अधिकांश ऑपरेटर्स एक किंवा दोन ऑपरंडसह ओव्हरलोड केले जाऊ शकतात. उदाहरणार्थ, यूनरी ऑपरेटर्ससारख्या ++ म्हणजे एक किंवा बायनरी ऑपरेटर्ससारख्या + म्हणजे दोन ऑपरंडसह.

- **प्रेसिडेन्स आणि असोसिएटिव्हिटी (Precedence and associativity)**

जेव्हा ऑपरेटर्स ओव्हरलोड करता तेव्हा ऑपरेटरची प्रेसिडेन्स आणि असोसिएटिव्हिटी बदलू शकत नाही. उदाहरणार्थ, (*) ची प्रेसिडेन्स किंवा (/) ची असोसिएटिव्हिटी बदलता येणार नाही.

- **रिटर्न (Return) टाइप**

जेव्हा आपण ऑपरेटर्स ओव्हरलोड करता तेव्हा, आपल्या आवश्यकतेनुसार ओव्हरलोड केलेल्या ऑपरेटर फंक्शनची परत टाइप परिभाषित करणे आवश्यक आहे.

- **फ्रेंड (Friend) फंक्शन**

जर ऑपरेटर्स ओव्हरलोड केले जातात आणि त्यांच्या ऑपरेटर्सला क्लासच्या खासगी सदस्य व्यवस्थांचा उपयोग करावा लागतो तर त्यांना मित्र फंक्शनस्वरूपात घोषित करणे आवश्यक आहे.

- **विशेष सिंटॅक्स: (Syntax)**

() किंवा [], -> यासारख्या ऑपरेटर्ससाठी जेव्हा ओव्हरलोड केले जाते तेव्हा, ओव्हरलोड केलेल्या फंक्शनच्या विशेष सिंटॅक्सचा वापर करावा लागतो. उदाहरणार्थ, [] ऑपरेटरचे ओव्हरलोड करण्यासाठी आणि ऑब्जेक्ट्सला कॉलेबलसरखे बनविण्यासाठी, आपल्याला operator() नावाच्या फंक्शनची परिभाषा करावी लागेल.

- **अनरी आणि बायनरी ऑपरेटरचे ओव्हरलोडिंग (Unary and binary Operator overloading)**

अनरी ऑपरेटर	+, -, ++, --
बायनरी ऑपरेटर	+, -, *, /, %

अनरी ऑपरेटर एकाच ऑपरेंडवर काम करतात, तर बायनरी ऑपरेटर दोन ऑपरेंडवर काम करतात.

उदाहरण: ओव्हरलोडिंग युनरी ऑपरेटर "-":

```
#include <iostream.h>
class Point {
private:
    int x, y;
public:
    // Default constructor
    Point() {
        x=0;
        y=0;
    }
    // Parameterized constructor
    Point(int xVal, int yVal){
        x=xVal;
        y=yVal;
    }
    // Overload unary minus (-) operator
    Point operator-() const {
```

```

        return Point(-x, -y);
    }
    // Display function
    void display() const {
        cout << "\nPoint(" << x << ", " << y << ")";
    }
};

void main() {
    Point p1(10, 20);
    Point p2 = -p1; // Calls overloaded unary minus operator
    cout << "Original Point: ";
    p1.display();
    cout << "Negated Point: ";
    p2.display();
}

```

Output:

Original Point:

Point(10 , 20)

Negated Point:

Point(10 , 20)

उदाहरण: ओवरलोडिंग बाइनरी ऑपरेटर "+" :

```

#include <iostream.h>
#include <string.h>
class String {
private:
    char* str;
public:
    // Constructor
    String(const char* s = "") {
        str = new char[strlen(s) + 1];
        strcpy(str, s);
    }
    // Overload + operator for concatenating strings
    String operator+(const String& other) const {
        char* newStr = new char[strlen(str) + strlen(other.str) + 1];
        strcpy(newStr, str);
        strcat(newStr, other.str);
        String temp(newStr);
        delete[] newStr;
        return temp;
    }
    // Display function
    void display() const {
        cout << "\n"<<str;
    }
}

```

```
};
void main() {
    String s1("Hello, ");
    String s2("World!");
    String s3 = s1 + s2; // Calls overloaded + operator
    cout << "String 1: ";
    s1.display();
    cout << "String 2: ";
    s2.display();
    cout << "Concatenated String: ";
    s3.display();
}
```

Output:

String 1:Hello,

String 2:World!

String 3: Hello,World!

4.5 रन टाइम पॉलिमॉर्फिझम (Run Time Polymorphism):

या प्रकारचे बहुरूपता फंक्शन ओव्हरराइडिंगद्वारे प्राप्त होते . लेट बाइंडिंग आणि डायनॅमिक पॉलिमॉर्फिझम ही रनटाइम पॉलिमॉर्फिझमची इतर नावे आहेत. रनटाइम पॉलिमॉर्फिझममध्ये रनटाइमच्या वेळी फंक्शन कॉलचे निराकरण केले जाते . याउलट, कंपाइल टाइम पॉलिमॉर्फिझमसह, कंपाइलर रनटाइमच्या वेळी वजा करून ऑब्जेक्टला कोणते फंक्शन कॉल बांधायचे हे ठरवतो.

● फंक्शन ओव्हरराइडिंग (Function Overloading)

फंक्शन ओव्हरराइडिंग तेव्हा होते जेव्हा डिराईव्हड क्लासची बेस क्लासच्या मेंबर फंक्शन्सपैकी एकाची व्याख्या असते. ते बेस फंक्शन ओव्हरराइड केले आहे असे म्हटले जाते.

डेटा सदस्यांसह रनटाइम पॉलिमॉर्फिझम-

C++ मधील डेटा मेंबरद्वारे रनटाइम पॉलिमॉर्फिझम साध्य करता येत नाही. आपण पॅरेंट क्लासच्या रेफरन्स व्हेरिएबलद्वारे फील्ड ऍक्सेस करत आहोत असे उदाहरण पाहू जे डिराईव्हड क्लासच्या उदाहरणाचा संदर्भ देते.

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

```
// base class declaration.
```

```
class Animal {
```

```
public:
```

```
    string color = "Black";
```

```
};
```

```
// inheriting Animal class.
```

```
class Dog : public Animal {
```

```
public:
```

```
    string color = "Grey";
```

```
};
```

```
// Driver code
```

```
void main()
```

```
{
```

```
    Animal d = Dog(); // accessing the field by reference
```

```
                        // variable which refers to derived
```

```
    cout << d.color;
```

}

Output:

Black

● **व्हर्चुअल फंक्शन (Virtual Function):**

व्हर्चुअल फंक्शन हे सदस्य फंक्शन आहे जे बेस क्लासमध्ये व्हर्चुअल कीवर्ड वापरून घोषित केले जाते आणि डिराईव्हड वर्गामध्ये पुन्हा परिभाषित (ओव्हरराइड) केले जाते.

व्हर्चुअल फंक्शन्सबद्दल काही महत्वाचे मुद्दे:

- व्हर्चुअल फंक्शन्स डायनॅमिक स्वरूपाची असतात.
- ते बेस क्लासमध्ये "व्हर्चुअल" हा कीवर्ड टाकून परिभाषित केले जातात आणि नेहमी बेस क्लाससह घोषित केले जातात आणि चाइल्ड क्लासमध्ये ओव्हरराइड केले जातात.
- रनटाइम दरम्यान व्हर्चुअल फंक्शन कॉल केले जाते.

व्हर्चुअल फंक्शन दाखवण्यासाठी खाली C++ प्रोग्राम आहे:

```
#include <iostream.h>
// Declaring a Base class
class Base {
public:
    // virtual function
    virtual void display()
    {
        cout << "Called virtual Base Class function"<< "\n\n";
    }
    void print()
    {
        cout << "Called Base print function"<< "\n\n";
    }
};
// Declaring a Child Class
class Child : public Base {
public:
    void display()
    {
        cout << "Called Child Display Function"<< "\n\n";
    }
    void print()
    {
        cout << "Called Child print Function"<< "\n\n";
    }
};
// Driver code
void main()
{
    // Create a reference of class Base
    Base* base;
    Child child1;
    base = &child1;
```

```

    // This will call the virtual function
    base-> base::display();
    // this will call the non-virtual function
    base->print();
}

```

Output:

Called virtual Base Class function

Called Base print function

● प्युअर व्हर्च्युअल फंक्शन (Pure Virtual Function) :

C++ मधील शुद्ध व्हर्च्युअल फंक्शन (किंवा अॅबस्ट्रॅक्ट फंक्शन) हे व्हर्च्युअल फंक्शन आहे ज्यासाठी आपण अंमलबजावणी करू शकतो, परंतु आपण ते फंक्शन डिराईव्हड वर्गात ओव्हरराइड केले पाहिजे, अन्यथा, डिराईव्हड वर्ग देखील एक अमूर्त वर्ग होईल. घोषणेमध्ये 0 नियुक्त करून शुद्ध आभासी कार्य घोषित केले जाते.

शुद्ध व्हर्च्युअल फंक्शन्सचे उदाहरण :

```

// An abstract class
class Test {
    // Data members of class
public:
    // Pure Virtual Function
    virtual void show() = 0;
    /* Other members */
};

```

पूर्ण उदाहरण

प्युअर व्हर्च्युअल फंक्शन अॅबस्ट्रॅक्ट क्लासमधून घेतलेल्या वर्गाद्वारे लागू केले जाते.

```

#include <iostream.h>
class Base {
    // private member variable
    int x;
public:
    // pure virtual function
    virtual void fun() = 0;
    // getter function to access x
    int getX() { return x; }
};
// This class inherits from Base and implements fun()
class Derived : public Base {
    // private member variable
    int y;
public:
    // implementation of the pure virtual function

    void fun()
    {
        cout << "fun() called";
    }
};

```

```
void main()
{
    // creating an object of Derived class
    Derived d;
    // calling the fun() function of Derived class
    d.fun();
}
```

Output:

fun() called

संदर्भ:

1. <https://www.w3schools.com/cpp/>
2. <https://www.javatpoint.com/cpp-tutorial>
3. <https://www.programiz.com/cpp-programming>
4. <https://www.programiz.com/cpp-programming/online-compiler/>
5. https://www.onlinegdb.com/online_c++_compiler

युनिट - 5

फाइल ऑपरेशन (File Operation)

विषय निष्पत्ती (Course Outcome): फाइल ऑपरेशन्स करण्यासाठी C++ प्रोग्राम विकसित करणे. (Develop C++ Program to perform file operations.)

घटक निष्पत्ती (Theory Learning Outcomes):

1. दिलेल्या फाइलवर ऑपरेशन करण्यासाठी संबंधित वर्ग(class) ओळखणे.
2. भिन्न फाइल मोडचे वर्णन करणे.
3. फाइलमधून वाचन/लेखन (Read/Write) ऑपरेशन करण्यासाठी C++ प्रोग्राम विकसित करणे.

5.1 C++ स्ट्रीम वर्ग (Stream Class)-C++ स्ट्रीम क्लासेस, फाईल स्ट्रीम ऑपरेशन्ससाठी क्लासेस. (C++ Stream Classes, Classes for file stream operations.)

C + + स्ट्रीम हा प्रोग्राममध्ये किंवा बाहेर डेटाचा प्रवाह आहे जसे की डेटा मोजण्यासाठी किंवा रीडमधून परत येतो.C++ प्रवाहासह ऑपरेट करण्यासाठी स्टँडर्ड iOS प्रवाह लायब्ररी प्रदान करते.iOS स्ट्रीम ही एक ऑब्जेक्ट ओरिएंटेड लायब्ररी आहे जी प्रवाह वापरून इनपुट आउटपुट कार्यक्षमता प्रदान करते. C++ मध्ये फाइल्सशी संबंधित विविध प्रवाह परिभाषित करण्यासाठी आणि इनपुट-आउटपुट ऑपरेशन्स करण्यासाठी अनेक प्रवाह क्लास आहेत. हे सर्व क्लास iostream.h फाइलमध्ये परिभाषित केले आहेत. खाली दिलेली Fig. 5.1 या क्लासची पदानुक्रम दर्शवते.

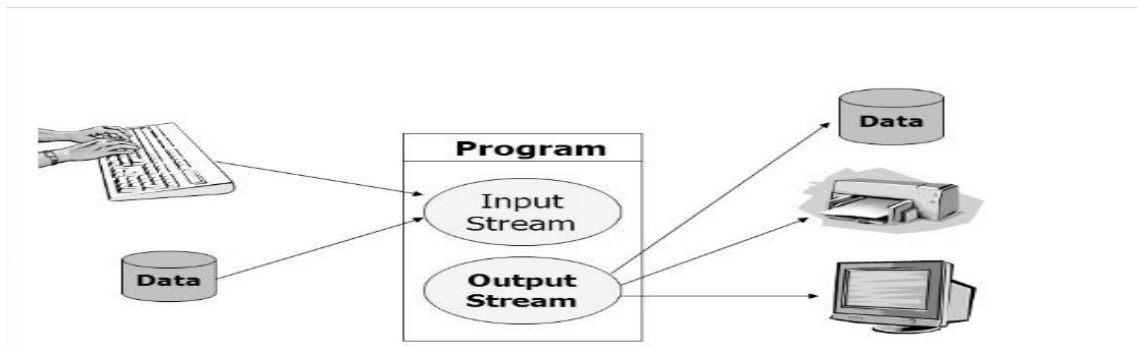
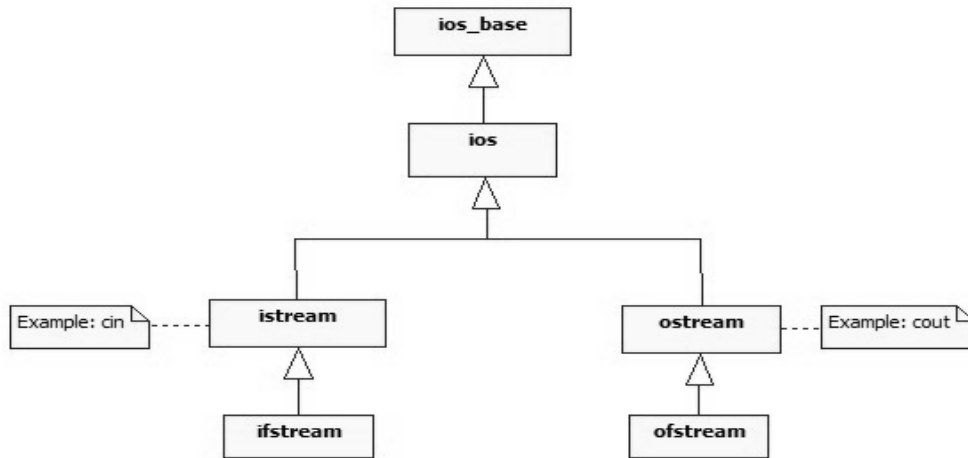


Fig. 5.1: Stream Class Hierarchy

1. **ios** क्लास हा प्रवाह क्लास पदानुक्रमातील सर्वोच्च वर्ग आहे. `istream`, `ostream` आणि `streambuf` वर्गासाठी हा बेस क्लास आहे.

2. **istream आणि ostream iostream** क्लाससाठी बेस क्लास देतात. क्लास istream इनपुटसाठी आणि ओस्ट्रीम आउटपुटसाठी वापरला जातो.
3. **क्लास ios** अप्रत्यक्षपणे आयस्ट्रीम आणि ओस्ट्रीम वापरून आयओस्ट्रीम क्लासला वारसा मिळाला आहे. आयओएस क्लासच्या डेटा आणि सदस्य फंक्शन्सची डुप्लिसीटी टाळण्यासाठी, आयस्ट्रीम आणि ओस्ट्रीममध्ये इनहेरिट करताना व्हर्च्युअल बेस क्लास म्हणून घोषित केले जाते.

या प्रवाह वर्गाद्वारे प्रदान केलेल्या सुविधा-

- **ios वर्ग:** ios क्लास इतर सर्व प्रवाह वर्गांना सर्व इनपुट आणि आउटपुट सुविधा प्रदान करण्यासाठी जबाबदार आहे.
- **istream वर्ग:** हा क्लास इनपुट प्रवाह हाताळण्यासाठी जबाबदार आहे. हे अक्षर, स्ट्रिंग आणि गेट, गेटलाइन, रीड, इग्नोर, पुटबॅक इत्यादीसारख्या वस्तू हाताळण्यासाठी फंक्शनची संख्या प्रदान करते.

Example:

```
#include <iostream>
using namespace std;
int main()
{
    char x;
    // used to scan a single char
    cin.get(x);

    cout << x;
}
```

Output:

If the user inputs the character A, the console output will be:

A

If the user inputs the character b, the console output will be:

b

- **ostream क्लास:** हा क्लास आउटपुट प्रवाह हाताळण्यासाठी जबाबदार आहे. हे अक्षरे, स्ट्रिंग्स आणि ऑब्जेक्ट्स जसे की लेखन, पुट इत्यादी हाताळण्यासाठी फंक्शनची संख्या प्रदान करते.

Example:

```
#include <iostream>
using namespace std;
int main()
{
    char x;
    // used to scan a single char
    cin.get(x);
    // used to put a single char onto the screen.
    cout.put(x);
}
```

Output:

If the user inputs the character A, the console output will be:

A

If the user inputs the character b, the console output will be:

b

- **iostream:** हा क्लास इनपुट आणि आउटपुट दोन्ही प्रवाह हाताळण्यासाठी जबाबदार आहे कारण आयस्ट्रीम क्लास आणि ओस्ट्रीम क्लास दोन्ही त्यात वारशाने मिळतात. गेट, गेटलाइन, रीड, इग्नोर, पुटबॅक, पुट, राइट इत्यादी वर्ण, स्ट्रिंग आणि ऑब्जेक्ट्स हाताळण्यासाठी हे istream क्लास आणि ostream क्लास दोन्हीचे कार्य प्रदान करते.

उदाहरण:

```
#include <iostream>
using namespace std;
int main()
{
    // this function display
    // ncount character from array
    cout.write("welcome in file handling", 5);
}
```

the output of the program will be:

welcome in file handling

Streams in C++ :- आम्ही एक्झिक्युटिंग प्रोग्रामला इनपुट देतो आणि एक्झिक्यूशन प्रोग्राम आउटपुट परत देतो. एक्झिक्युटिंग प्रोग्रॅमला इनपुट म्हणून दिलेल्या बाइट्सचा क्रम आणि एक्झिक्युटिंग प्रोग्राममधून आउटपुट म्हणून येणाऱ्या बाइट्सच्या क्रमाला प्रवाह म्हणतात. दुसऱ्या शब्दांत, प्रवाह हे अनुक्रमे डेटाच्या प्रवाहाशिवाय दुसरे काहीही नसतात. एक्झिक्युटिंग प्रोग्राम आणि कीबोर्ड आणि मॉनिटर सारख्या उपकरणांमधील इनपुट आणि आउटपुट ऑपरेशनला "कन्सोल I/O ऑपरेशन" म्हणून ओळखले जाते. एक्झिक्युटिंग प्रोग्राम आणि फाइल्समधील इनपुट आणि आउटपुट ऑपरेशनला "डिस्क I/O ऑपरेशन" म्हणून ओळखले जाते.

Classes for File stream operations :

C++ च्या I/O प्रणालीमध्ये क्लासचा संच असतो जो फाइल हाताळण्याच्या पद्धती परिभाषित करतो. यामध्ये ifstream, ofstream आणि fstream वर्गांचा समावेश आहे. हे क्लास fstream आणि संबंधित istream वर्गातून घेतले आहेत. डिस्क फाइल्स व्यवस्थापित करण्यासाठी डिझाइन केलेले हे वर्ग fstream मध्ये घोषित केले जातात आणि म्हणून आम्ही फाइल्स वापरणाऱ्या कोणत्याही प्रोग्राममध्ये ही फाइल समाविष्ट केली पाहिजे.

1. ios:- ios stands for input output stream. हा क्लास या क्लास पदानुक्रमातील इतर वर्गांसाठी आधार वर्ग आहे. या क्लास मध्ये आवश्यक सुविधा आहेत ज्यांचा वापर इतर सर्व डिराईव्हड क्लास इनपुट आणि आउटपुट ऑपरेशन्ससाठी करतात.

2. istream:- istream stands for input stream. हा क्लास 'ios' या क्लासपासून घेतला आहे. हा क्लास इनपुट प्रवाह हाताळतो. एक्सट्रॅक्शन ऑपरेटर(>>) फाइल्सपासून प्रोग्रॅमच्या अंमलबजावणीपर्यंत इनपुट प्रवाह हाताळण्यासाठी या क्लासत ओव्हरलोड केलेले आहे. हा क्लास इनपुट फंक्शन्स घोषित करतो जसे की get(), getline() आणि read().

3. ostream:- ostream stands for output stream. हा वर्ग 'ios' या क्लास पासून घेतला आहे.

हा क्लास आउटपुट प्रवाह हाताळतो. प्रोग्राम एक्झिक्यूशनमधून फाइल्सवर आउटपुट स्ट्रीम हाताळण्यासाठी या क्लासमध्ये इन्सर्शन ऑपरेटर(<<) ओव्हरलोड केला जातो. हा क्लास put() आणि write() सारखी आउटपुट फंक्शन्स घोषित करतो.

4. streambuf:- या क्लासत एक पॉइंटर आहे जो बफरकडे निर्देश करतो जो इनपुट आणि आउटपुट प्रवाह व्यवस्थापित करण्यासाठी वापरला जातो.

5. fstreambase:- हा क्लास फाईल स्ट्रीमसाठी सामान्य ऑपरेशन्स प्रदान करतो. एफस्ट्रीम, इफस्ट्रीम आणि ऑफस्ट्रीम क्लाससाठी आधार म्हणून काम करते. या क्लासत open() आणि close() फंक्शन आहे.

6. ifstream:- This class provides input operations.

त्यात डीफॉल्ट इनपुट मोडसह open() फंक्शन आहे. ifstream मधून get(), getline(), read(), seekg() आणि tellg() फंक्शन्स इनहेरिट करते.

7. ofstream:- This class provides output operations.

त्यात डीफॉल्ट आउटपुट मोडसह open() फंक्शन आहे. ofstream मधून put(), write(), seekp() आणि tellp() फंक्शन्स इनहेरिट करते.

8. fstream:-

हा क्लास एकाचवेळी इनपुट आणि आउटपुट ऑपरेशन्ससाठी समर्थन प्रदान करतो. ifstream द्वारे ifstream आणि ofstream वर्गातील सर्व फंक्शन्स इनहेरिट करते.

9. filebuf:-

वाचन आणि लिहिण्यासाठी फाइल बफर सेट करणे हा त्याचा उद्देश आहे. फाइलची लांबी निश्चित करण्यासाठी आम्ही फाइल बफर मेंबर फंक्शन देखील वापरू शकतो. C++ मध्ये, fstream headerfile मध्ये उपलब्ध असलेल्या fstream, ifstream, ofstream या तीन वर्गांचा वापर करून फाइल्स मुख्यतः हाताळल्या जातात.

ofstream: फायलीवर लिहिण्यासाठी प्रवाह क्लास

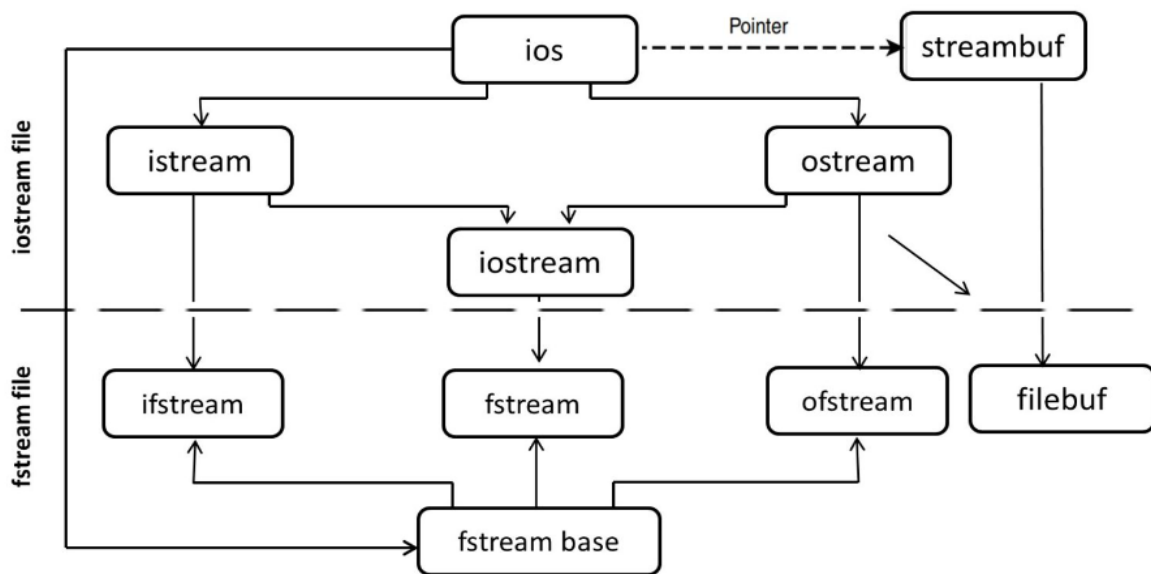
ifstream: फायलीमधून वाचण्यासाठी प्रवाह क्लास

fstream: फायलीमधून/पर्यंत वाचणे आणि लिहिणे या दोन्हीसाठी क्लास प्रवाहित करा.

फाइल स्ट्रीम वर्ग (File Stream Classes):-

1. ifstream - हा वर्ग (Class) इनपुट फाइल्स हाताळण्यासाठी वापरतो.
2. ofstream - हा वर्ग (Class) आउटपुट फाइल्स हाताळण्यासाठी वापरतो.
3. Fstream - हा वर्ग (Class) इनपुट आणि आउटपुट फाइल्स हाताळण्यासाठी वापरला जातो.

फाइल प्रवाह वर्ग पदानुक्रम (File stream class hierarchy)



File Stream Classes Hierarchy

Fig. 5.2 File stream class hierarchy

5.2 c++ मधील फाइलचा शेवट शोधणे , फाइल मोड शोधणे (Detection of End of File, File modes)

5.2.1 c++ मधील फाइलचा शेवट शोधणे (Detection of End of File)

C++ मध्ये फाइलचा शेवट ओळखण्यासाठी विविध पद्धती आहेत. येथे काही पद्धती दिल्या आहेत ज्याद्वारे आपण C++ मध्ये फाइलचा शेवट (EOF) ओळखू शकता. यासाठी आपण ifstream (input file stream) चा वापर करू

शकता.फाईलमध्ये किती डेटा साठवला जातो हे मात्र अनेकदा माहीत नसते. तर, आपण आपला वेळ मजकूर फाईलमधील डेटा हाताने मोजण्यात घालवतो, की संगणकाला डेटाच्या प्रमाणात व्यवहार करू देतो? अर्थात, आम्ही संगणकाला मोजणी करू देतो.C++ एक विशेष फंक्शन प्रदान करते, eof(), जे इनपुट फाईल स्ट्रीममधून वाचण्यासाठी अधिक डेटा नसताना शून्य (म्हणजे TRUE) आणि अन्यथा शून्य (म्हणजे FALSE) देते.

एन्ड-ऑफ-फाईल वापरण्याचे नियम (eof()):

1. इनपुट फाईल स्ट्रीममधून वाचलेल्या डेटावर प्रक्रिया करण्यापूर्वी नेहमी फाईलच्या शेवटच्या स्थितीची चाचणी घ्या.

- i. लूप सुरू करण्यापूर्वी प्राइमिंग इनपुट स्टेटमेंट वापरा
- ii. लूप बॉडीच्या तळाशी इनपुट स्टेटमेंट पुन्हा करा

2. इनपुट फाईल स्ट्रीममधून डेटा मिळविण्यासाठी थोडा वेळ लूप वापरा. for लूप तेव्हाच इष्ट आहे जेव्हा तुम्हाला फाईलमधील डेटा आयटमची अचूक संख्या माहित असते, जी आम्हाला माहित नसते.

1.EOF ओळखण्यासाठी eof() पद्धती

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
    ifstream file("example.txt");
    if (!file.is_open()) {
        cout << "फाईल उघडली नाही." << endl;
        return 1;
    }
    string line;
    while (!file.eof()) {
        getline(file, line);
        cout << line << endl;
    }
    file.close();
    return 0;
}
```

Example:

If example.txt contains the following text:

Hello, world!

This is an example file.

The output will be:

Hello, world!

This is an example file.

2. EOF ओळखण्यासाठी getline() पद्धती

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
    ifstream file("example.txt");
    if (!file.is_open()) {
        cout << "फाईल उघडली नाही." << endl;
```

```

    return 1;
}
string line;
while (getline(file, line)) {
    cout << line << endl;
}
file.close();
return 0;
}

```

Example 1:

If example.txt contains the following text:

Hello, world!

This is an example file.

Output:

Hello, world!

This is an example file.

3.EOF ओळखण्यासाठी operator >> पद्धती

```

#include <iostream>
#include <fstream>
using namespace std;
int main() {
    ifstream file("example.txt");
    if (!file.is_open()) {
        cout << "फाईल उघडली नाही." << endl;
        return 1;
    }
    string word;
    while (file >> word) {
        cout << word << " ";
    }
    file.close();
    return 0;
}

```

Example 1: If example.txt contains the text:

Hello world! This is an example.

Output:

Hello world! This is an example.

Each word is printed followed by a space, including punctuation marks.

Example 2: If example.txt contains the text:

The quick brown fox jumps over the lazy dog.

Output:

sql

Copy code

The quick brown fox jumps over the lazy dog.

Output:

The quick brown fox jumps over the lazy dog.

Each word is printed followed by a space.

Example 3: If example.txt contains:

One

Two

Three

Output:

One Two Three

Each word is printed on the same line with spaces in between.

Example 4: If example.txt is empty:

<no output>

Since there are no words to read, the program will not produce any output.

Example 5: If example.txt does not exist or cannot be opened:

फाईल उघडली नाही.

वरील उदाहरणांमध्ये, आपण फाईलचा शेवट ओळखण्यासाठी वेगवेगळ्या पद्धती वापरल्या आहेत. या सर्व पद्धती आपल्याला फाईलमधील डेटा योग्यरित्या वाचण्यास मदत करतील. EOF ओळखण्यासाठी eof(), getline() आणि operator >> या पद्धती अत्यंत उपयुक्त आहेत.

5.2.2 फाईल मोड (File modes)

C++ मध्ये फाईल मोड्स विविध प्रकारे वापरता येतात, जेणेकरून फाईल ओपन करताना विशिष्ट क्रिया किंवा अटी लागू करता येतात. इथे काही सामान्य फाईल मोड्स दिले आहेत:

1. ios::in - वाचनासाठी फाईल उघडते.

```
ifstream file("example.txt", ios::in);
```

2. ios::out - लिहिण्यासाठी फाईल उघडते. फाईल अस्तित्वात नसेल तर नवीन फाईल तयार करते. अस्तित्वात असल्यास, फाईलच्या सामग्रीवर पुनर्लेखन करते.

```
ofstream file("example.txt", ios::out);
```

3. ios::app - लिहिण्यासाठी फाईल उघडते, परंतु नवीन डेटा शेवटी जोडते (append).

```
ofstream file("example.txt", ios::app);
```

4. ios::ate - फाईल उघडते आणि फाईलच्या शेवटी पोहोचते. वाचन आणि लिखाणासाठी वापरता येते.

```
ofstream file("example.txt", ios::ate);
```

5. ios::trunc - फाईल उघडते आणि तिची सामग्री काढून टाकते. सामान्यतः ios::out सोबत वापरली जाते.

```
ofstream file("example.txt", ios::out | ios::trunc);
```

6. ios::binary - फाईलला बायनरी मोडमध्ये उघडते. यामुळे डेटा बायनरी स्वरूपात लिहिला किंवा वाचला जातो.

```
ofstream file("example.bin", ios::out | ios::binary);
```

• एकत्रित मोड्स वापरणे

फाईल मोड्स एकत्रित करून फाईल ऑपरेशन सुसंगत बनवता येते. उदाहरणार्थ:

```
ofstream file("example.txt", ios::out | ios::app);
```

वरील कोडमध्ये फाईल लिहिण्यासाठी उघडली जाते आणि नवीन डेटा शेवटी जोडला जातो.

उदाहरण: (Reading and Writing in file)

```
#include <iostream>
```

```
#include <fstream>
```

```
using namespace std;
```

```
void writeFile(const string& filename) {
```

```
    ofstream outFile(filename, ios::out); // Open the file for writing
```

```
    if (!outFile) {
```

```

    cerr << "Failed to open the file for writing: " << filename << endl;
    return;
}
outFile << "This is an example." << endl;
outFile << "Learning file read and write in C++." << endl;
outFile.close(); // Close the file
cout << "Successfully wrote to the file: " << filename << endl;
}

void readFile(const string& filename) {
    ifstream inFile(filename, ios::in); // Open the file for reading
    if (!inFile) {
        cerr << "Failed to open the file for reading: " << filename << endl;
        return;
    }
    string line;
    while (getline(inFile, line)) { // Read the file line by line
        cout << line << endl;
    }
    inFile.close(); // Close the file
}

int main() {
    string filename = "example.txt";
    writeFile(filename); // Call the writeFile function to write to the file
    readFile(filename); // Call the readFile function to read from the file
    return 0;
}

```

Expected Output:

When you run this program, it should output the following:

Successfully wrote to the file: example.txt

This is an example.

Learning file read and write in C++.

या उदाहरणांमध्ये आपण विविध फाईल मोड्सचा वापर कसा करावा हे पाहिले. फाईल मोड्सचा योग्य वापर केल्याने फाईल ऑपरेशन्स अधिक प्रभावी आणि सुरक्षित बनतात.

5.3 C++ मध्ये फाइल्स उघडणे (Opening files in C++)

फाईलमध्ये डेटा वाचण्यासाठी किंवा प्रविष्ट करण्यासाठी, आम्हाला प्रथम ती उघडण्याची आवश्यकता आहे. हे वाचनासाठी 'ifstream' आणि फाइल लिहिण्यासाठी किंवा जोडण्यासाठी 'fstream' किंवा 'ofstream' च्या मदतीने करता येते. या तिन्ही ऑब्जेक्ट्समध्ये ओपन() फंक्शन प्री-बिल्ट आहे.

मांडणी(Syntax)**1.open(FileName , Mode);**

येथे:फाईलचे नाव - हे फाइलचे नाव दर्शवते जी उघडायची आहे.

मोड - फाईल उघडण्यासाठी भिन्न मोड आहे आणि ते या लेखात स्पष्ट केले आहे.

फाइल मोड पॅरामीटर (File mode parameter)

iso:: फाईल वाचन मोडमध्ये उघडली

ios::out फाईल लेखन मोडमध्ये उघडली
 ios::app फाईल संलग्न मोडमध्ये उघडली
 ios::ate फाईल अपेंड मोडमध्ये उघडली परंतु फाईलच्या शेवटी वाचा आणि लिहा.
 ios::baenरी फाइल बायनरी मोडमध्ये उघडली
 ios::trunc फाईल ट्रंकेट मोडमध्ये उघडली
 ios::nocreate फाईल अस्तित्वात असेल तरच उघडते
 ios::noreplace फाईल अस्तित्वात नसल्यासच उघडते

Program for Opening File:

```
#include<iostream>
#include<fstream>
using namespace std;
int main(){
    fstream FileName;
    FileName.open("FileName", ios::out);
    if (!FileName){
        cout<<"Error while creating the file";
    }
    else{
        cout<<"File created successfully";
        FileName.close();
    }
    return 0;
}
```

वरील कोडचे स्पष्टीकरण

येथे आमच्याकडे आयओस्ट्रीम लायब्ररी आहे, जी इनपुट/आउटपुट प्रवाहासाठी जबाबदार आहे. आमच्याकडे एक fstream लायब्ररी देखील आहे, जी फाइल्स हाताळण्यासाठी जबाबदार आहे. एफस्ट्रीम क्लासचा एक ऑब्जेक्ट तयार करणे आणि त्याला 'फाइलनेम' असे नाव दिले.

वर तयार केलेल्या ऑब्जेक्टवर, नवीन फाइल तयार करण्यासाठी आपल्याला ओपन() फंक्शन लागू करावे लागेल आणि मोड 'आउट' वर सेट केला जाईल ज्यामुळे आम्हाला फाइलमध्ये लिहिता येईल. फाइल निर्माण तपासण्यासाठी आम्ही 'if' स्टेटमेंट वापरतो. फाइल अस्तित्वात नसल्यास कन्सोलवर संदेश मुद्रित करते. फाइल अस्तित्वात असल्यास/तयार केली असल्यास कन्सोलवर संदेश प्रिंट करते. फाईल बंद करण्यासाठी आपण ऑब्जेक्टवर close() फंक्शन वापरतो.

Output

File created successfully

5.3.1 फाइलमध्ये लिहिणे (Writing to file)

आत्तापर्यंत आपण C++ वापरून फाईल कशी तयार करायची ते शिकलो. आता, आपण आधी तयार केलेल्या फाइलमध्ये डेटा कसा लिहायचा ते शिकू. फाइलमध्ये डेटा लिहिण्यासाठी आणि तसे करण्यासाठी आम्ही fstream किंवा ofstream ऑब्जेक्ट वापरू; आम्ही डबल-कोट्समध्ये संलग्न मजकूरासह स्ट्रीम इन्सर्शन ऑपरेटर (<<) वापरू.

open() फंक्शनच्या मदतीने, आपण 'FileName' नावाची नवीन फाईल तयार करू आणि नंतर आपण मोडला 'ios::out' वर सेट करू कारण आपल्याला फाइलमध्ये डेटा लिहायचा आहे.

Syntax:

```
FileName<<"Insert the text here";
```

Program for Writing to File:

```
#include<iostream>
#include<fstream>
using namespace std;
int main() {
    fstream FileName;
    FileName.open("FileName.txt", ios::out);
    if (!FileName) {
        cout<<" Error while creating the file ";
    }
    else {
        cout<<"File created and data got written to file";
        FileName<<"This is a blog posted on <a href='\"https://www.mygreatlearning.com\" data-
internallinksmanager029f6b8e52c='\"11\" title='\"Great Learning Homepage\" target='\"_blank\"
rel='\"noopener\">Great Learning</a>";
        FileName.close();
    }
    return 0;
}
```

वरील कोडचे स्पष्टीकरण

येथे आमच्याकडे आयओस्ट्रीम लायब्ररी आहे, जी इनपुट/आउटपुट प्रवाहासाठी जबाबदार आहे. आमच्याकडे एक fstream लायब्ररी देखील आहे, जी फाइल्स हाताळण्यासाठी जबाबदार आहे. एफस्ट्रीम क्लासचा एक ऑब्जेक्ट तयार करणे आणि त्याला 'फाइलनेम' असे नाव दिले. वर तयार केलेल्या ऑब्जेक्टवर, नवीन फाइल तयार करण्यासाठी आपल्याला ओपन() फंक्शन लागू करावे लागेल आणि मोड 'आउट' वर सेट केला जाईल ज्यामुळे आम्हाला फाइलमध्ये लिहिता येईल. फाइल निर्माण तपासण्यासाठी आम्ही 'if' स्टेटमेंट वापरतो. फाइल अस्तित्वात नसल्यास कन्सोलवर संदेश मुद्रित करते. फाइल अस्तित्वात असल्यास/तयार केली असल्यास कन्सोलवर संदेश प्रिंट करते. तयार केलेल्या फाइलमध्ये डेटा लिहित आहे. फाइल बंद करण्यासाठी आपण ऑब्जेक्टवर close() फंक्शन वापरतो.

Output

File created and data got written to file

5.3.2 C++ मधील फाइलमधून वाचन (Reading from file in c++)

फाइलमधून डेटा मिळवणे ही एक अत्यावश्यक गोष्ट आहे कारण डेटा मिळाल्याशिवाय, आम्ही कोणतेही कार्य करू शकत नाही. परंतु काळजी करू नका, C++ हा पर्याय देखील प्रदान करतो. आम्ही वापरकर्त्याकडून डेटा मिळविण्यासाठी CIN सह फाइलमधील डेटाचे वाचन करू शकतो, परंतु नंतर आम्ही वापरकर्त्याच्या स्टॅंडर्ड कन्सोलमधून इनपुट घेण्यासाठी CIN वापरतो. येथे आपण fstream किंवा ifstream वापरू.

Syntax:

FileName>>Variable;

Program for Reading from File:

```
#include<iostream>
#include <fstream>
using namespace std;
int main()
{
    fstream FileName;
```

```

FileName.open("FileName.txt", ios::in);
if (!FileName) {
    cout<<"File doesn't exist.";
}
else {
    char x;
    while (1) {
        FileName>>x;
        if(FileName.eof())
            break;
        cout<<x;
    }
}
FileName.close();
return 0;
}

```

वरील कोडचे स्पष्टीकरण

येथे आमच्याकडे आयओस्ट्रीम लायब्ररी आहे, जी इनपुट/आउटपुट प्रवाहासाठी जबाबदार आहे. आमच्याकडे एक fstream लायब्ररी देखील आहे जी फाइल्स हाताळण्यासाठी जबाबदार आहे. एफस्ट्रीम क्लासचा एक ऑब्जेक्ट तयार करणे आणि त्याला 'फाइलनेम' असे नाव दिले. वर तयार केलेल्या ऑब्जेक्टवर, नवीन फाइल तयार करण्यासाठी आपल्याला ओपन() फंक्शन लागू करावे लागेल आणि मोड 'इन' वर सेट केला जाईल ज्यामुळे आम्हाला फाइलमधून वाचता येईल. फाइल निर्माण तपासण्यासाठी आम्ही 'if' स्टेटमेंट वापरतो. फाइल अस्तित्वात नसल्यास कन्सोलवर संदेश मुद्रित करते.

x नावाने वर्ण(char) व्हेरिएबल तयार करणे. while लूपच्या मदतीने फाइलचे पुनरावृत्ती. व्हेरिएबल x वर फाइल डेटा मिळवणे. येथे आपण eof() सह if कंडिशन वापरत आहोत जे फाइलच्या शेवटी कंपाइलरला फाइलच्या शेवटपर्यंत वाचण्यास सांगते. फाइल शेवटपर्यंत पोहोचल्यावर वाचन थांबवण्यासाठी आम्ही 'ब्रेक' स्टेटमेंट वापरतो. व्हेरिएबल x मध्ये उपलब्ध असलेली सामग्री प्रिंट करण्यासाठी प्रिंट स्टेटमेंट. फाइल बंद करण्यासाठी आपण ऑब्जेक्टवर close() फंक्शन वापरतो

Output

Hello World, Thank You for Visiting Great Learning.

5.3.3 C++ मध्ये फाइल बंद करणे (Closing a file in c++)

फाइल बंद करणे हा एक चांगला सराव आहे आणि फाइल बंद करणे आवश्यक आहे. जेव्हा जेव्हा C++ प्रोग्राम संपतो तेव्हा तो वाटप केलेली मेमरी साफ करतो आणि फाइल बंद करतो. आपण close() फंक्शनच्या मदतीने कार्य करू शकतो.

syntax:

```
FileName.close();
```

Program to Close a File:

```

#include <iostream>
#include <fstream>
using namespace std;
int main() {
    fstream FileName;
    FileName.open("FileName.txt", ios::in);

```

```

if (!FileName) {
    cout<<"File doesn't exist";
}
else {
    cout<<"File opened successfully";
}
}
FileName.close();
return 0;
}

```

वरील कोडचे स्पष्टीकरण

येथे आमच्याकडे आयओस्ट्रीम लायब्ररी आहे, जी इनपुट/आउटपुट प्रवाहासाठी जबाबदार आहे. आमच्याकडे एक `fstream` लायब्ररी देखील आहे, जी फाइल्स हाताळण्यासाठी जबाबदार आहे. एफस्ट्रीम क्लासचा एक ऑब्जेक्ट तयार करणे आणि त्याला 'फाइलनेम' असे नाव दिले. वर तयार केलेल्या ऑब्जेक्टवर, नवीन फाइल तयार करण्यासाठी आम्ही `open()` फंक्शन लागू करू, आणि मोड 'आउट' वर सेट केला आहे ज्यामुळे आम्हाला फाइलमध्ये लिहिता येते.

फाइल निर्माण तपासण्यासाठी आम्ही 'if' स्टेटमेंट वापरतो.

फाइल अस्तित्वात नसल्यास कन्सोलवर संदेश मुद्रित करते.

फाइल उघडली किंवा उघडली तर संदेश कन्सोलवर मुद्रित करते.

फाईल बंद करण्यासाठी आपण ऑब्जेक्टवर `close()` फंक्शन वापरतो.

5.3.4 फाइल हाताळणीसाठी वापरलेले सामान्य फंक्शन

`open()`: त्याचा वापर फाईल तयार करण्यासाठी केला जातो आणि विद्यमान फाईल उघडण्यासाठी देखील वापरला जातो.

`read()`: त्याचा वापर फाईलमधील डेटा वाचण्यासाठी केला जातो.

`write()`: फाईलमध्ये नवीन डेटा लिहिण्यासाठी वापरला जातो.

`close()`: फाईल बंद करण्यासाठी वापरले जाते.

`get()`: फाईलमधील एक अक्षर वाचण्यासाठी

`put()`-फाइलमध्ये एकच अक्षर लिहिण्यासाठी

C++ मध्ये फाईल उघडण्यासाठी आणि त्यावर विविध ऑपरेशन्स करण्यासाठी `ifstream`, `ofstream`, आणि `fstream` वापरता येतात. हे तीन प्रकारचे स्ट्रीम्स फाईल उघडण्यासाठी आणि त्यावर वाचन, लिखाण किंवा दोन्ही प्रकारचे ऑपरेशन्स करण्यासाठी वापरले जातात

ifstream:

वाचनासाठी फाईल उघडते.

`getline(inFile, line)` वापरून फाईलमधील डेटा ओळी-ओळीने वाचतो.

ofstream:

लिखाणासाठी फाईल उघडते.

`outFile << "हे एक उदाहरण आहे." << endl;` वापरून फाईलमध्ये डेटा लिहितो.

fstream:

वाचन आणि लिखाण दोन्ही करण्यासाठी फाईल उघडते.

`file.seekg(0, ios::beg);` वापरून फाईलच्या सुरुवातीला पोहोचतो आणि नंतर डेटा वाचतो.

5.3.5 फाइल उघडण्याच्या पद्धती (File Opening Methods):

1. C++ मध्ये फाइल उघडण्यासाठी open फंक्शन वापरता येते, जे फाइल स्ट्रीम ऑब्जेक्टवर कॉल केले जाते. open फंक्शन आपल्याला फाइलचे नाव आणि मोड निर्दिष्ट करून फाइल उघडण्याची परवानगी देते. हे ifstream, ofstream, आणि fstream सर्वांसाठी लागू होते.
2. C++ मध्ये फाइल उघडण्यासाठी स्ट्रीम क्लासेसच्या कन्स्ट्रक्टरचा वापर केला जाऊ शकतो. या पद्धतीने फाइल उघडण्यासाठी तुम्ही ifstream, ofstream, आणि fstream चे कन्स्ट्रक्टर वापरू शकता. कन्स्ट्रक्टरमध्ये तुम्ही थेट फाइलचे नाव आणि मोड पास करू शकता.

फाइल वाचण्यासाठी आणि लिहिण्यासाठी ओपन फंक्शन वापरणे (Using open Function to Read and Write to a File)

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
    fstream file;
    file.open("example.txt", ios::in | ios::out | ios::app); // Open the file for reading and writing
    if (!file) {
        cerr << "Failed to open the file: example.txt" << endl;
        return 1;
    }
    // Write new data to the file
    file << "This is a newly added line." << endl;
    // Go to the beginning of the file and read all data
    file.seekg(0, ios::beg);
    string line;
    while (getline(file, line)) {
        cout << line << endl;
    }
    file.close(); // Close the file
    return 0;
}
```

Expected Output:

Assuming example.txt initially contains some lines, and then after running this program, it appends "This is a newly added line." to the file. Upon reading and printing the entire file contents, the output might look like this:

This is the first line of example.txt.

This is the second line of example.txt.

This is a newly added line.

उदाहरण:

कन्स्ट्रक्टर वापरून फाइल उघडणे आणि लिहिणे (Using Constructors to Read and Write to a File)

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
    fstream file("example.txt", ios::in | ios::out | ios::app); // Open the file for reading and writing
    if (!file) {
```

```

    cerr << "Failed to open the file: example.txt" << endl;
    return 1;
}
// Write new data to the file
file << "This is a newly added line." << endl;
// Go to the beginning of the file and read all data
file.seekg(0, ios::beg);
string line;
while (getline(file, line)) {
    cout << line << endl;
}
file.close(); // Close the file
return 0;
}

```

Expected Output:

Assuming example.txt initially contains some lines, and after running this program, it appends "This is a newly added line." to the end of the file. Upon reading and printing the entire file contents, the output might look like this:

This is the first line of example.txt.

This is the second line of example.txt.

This is a newly added line.

5.3.6 फाइलमध्ये फॉर्मेट केलेले इनपुट आउटपुट फंक्शन (Formatted Input and Output Functions)

Besides using << and >> operators, you can also use other functions for formatted I/O:

- **getline:** Reads an entire line from the file.
- **put:** Writes a single character to the file.
- **get:** Reads a single character from the file.
- **write:** Writes a block of data to the file.
- **read:** Reads a block of data from the file.
- **Writing Formatted Data (ofstream):**

Opening a file for writing: ofstream outFile("output.txt");

This opens a file named "output.txt" for writing. If the file does not exist, it is created. If it does exist, its contents are truncated.

Writing formatted data: outFile << "Name: " << name << endl;

This uses the << operator to write formatted data to the file.

Closing the file: outFile.close();

This closes the file once all writing operations are done.

- **Reading Formatted Data (ifstream):**

Opening a file for reading: ifstream inFile("output.txt");

This opens a file named "output.txt" for reading.

Reading formatted data:

inFile >> ws; and getline(inFile, name); to read the name.

inFile.ignore(5); inFile >> age; to read the age after ignoring "Age: ".

inFile.ignore(8); inFile >> height; to read the height after ignoring "Height: ".

Closing the file: inFile.close();

This closes the file once all reading operations are done

Example

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
    ofstream outFile("output.txt", ios::out | ios::binary);
    if (!outFile) {
        cerr << "Failed to open the file for writing: output.txt" << endl;
        return 1;
    }
    // Writing raw binary data to the file
    const char data[] = "Binary data";
    outFile.write(data, sizeof(data));
    outFile.close(); // Close the file
    ifstream inFile("output.txt", ios::in | ios::binary);
    if (!inFile) {
        cerr << "Failed to open the file for reading: output.txt" << endl;
        return 1;
    }
    char buffer[256];
    inFile.read(buffer, sizeof(data));
    cout << "Read from file: " << buffer << endl;

    inFile.close(); // Close the file
    return 0;
}
```

Expected Output:

Given that the program writes "Binary data" to output.txt and then reads it back:

Read from file: Binary data

5.4 फाइलसचे प्रकार

फाइलचे प्रकार रँडम ऍक्सेस फाइल आणि सीक्वेन्शियल ऍक्सेस फाइल आहेत (types of file are random access file and Sequential access file)

1. रँडम ऍक्सेस फाइल (Random access file)

C++ मध्ये रँडम ऍक्सेस फाइल हँडलिंग म्हणजे तुम्ही फाइलमधील कोणत्याही ठिकाणाहून डेटा वाचू शकता किंवा लिहू शकता, कोणत्याही विशिष्ट क्रमाने न जाता. seekg आणि seekp फंक्शन्स वापरून हे साध्य केले जाते. seekg वाचनासाठी (ifstream) आणि seekp लिखाणासाठी (ofstream) वापरले जाते. fstream क्लासचा वापर करून दोन्ही कार्ये केली जाऊ शकतात.

स्पष्टीकरण**1. डेटा लिहिणे (ofstream):**

- ofstream outFile("random_access_file.txt", ios::out | ios::binary);
 - बायनरी मोडमध्ये फाइल उघडते आणि त्यामध्ये डेटा लिहिते.
- outFile << "ABCDEFGHJKLMNOPQRSTUVWXYZ";
 - फाइलमध्ये 26 अक्षरे लिहिते.
- outFile.close();

- फाईल बंद करते.

2. डेटा रँडमली वाचणे (ifstream):

- ifstream inFile("random_access_file.txt", ios::in | ios::binary);
 - बायनरी मोडमध्ये फाईल वाचनासाठी उघडते.
- inFile.seekg(10, ios::beg);
 - फाईलच्या सुरुवातीपासून 10व्या बाइटवर जाते.
- inFile.get(ch);
 - 10व्या बाइटवरील अक्षर वाचते.
- inFile.seekg(-5, ios::end);
 - फाईलच्या शेवटापासून 5व्या बाइटवर जाते.
- inFile.get(ch);
 - शेवटापासून 5व्या बाइटवरील अक्षर वाचते.
- inFile.close();
 - फाईल बंद करते.

3. विशिष्ट ठिकाणी डेटा बदलणे (fstream):

- fstream file("random_access_file.txt", ios::in | ios::out | ios::binary);
 - बायनरी मोडमध्ये वाचन आणि लिखाणासाठी फाईल उघडते.
- file.seekp(5, ios::beg);
 - फाईलच्या सुरुवातीपासून 5व्या बाइटवर जाते.
- file.put('X');
 - 5व्या बाइटवर 'X' अक्षर लिहिते.
- file.seekp(-3, ios::end);
 - फाईलच्या शेवटापासून 3व्या बाइटवर जाते.
- file.put('Y');
 - शेवटापासून 3व्या बाइटवर 'Y' अक्षर लिहिते.
- file.close();
 - फाईल बंद करते.

Example: Random Access File Handling in C++

Step 1: Writing Data to a File

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
    // Open the file for writing in binary mode
    ofstream outFile("random_access_file.txt", ios::out | ios::binary);
    if (!outFile) {
        cerr << "Failed to open the file for writing: random_access_file.txt" << endl;
        return 1;
    }
    // Writing some data to the file
    outFile << "ABCDEFGHJKLMNOPQRSTUVWXYZ"; // Write 26 characters
    outFile.close(); // Close the file
    return 0;
}
```

Step 2: Reading Data from Specific Positions

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
    // Open the file for reading in binary mode
    ifstream inFile("random_access_file.txt", ios::in | ios::binary);
    if (!inFile) {
        cerr << "Failed to open the file for reading: random_access_file.txt" << endl;
        return 1;
    }
    // Read character from a specific position
    inFile.seekg(10, ios::beg); // Go to the 10th byte from the beginning
    char ch;
    inFile.get(ch); // Read the character at that position
    cout << "Character at position 10: " << ch << endl;
    inFile.seekg(-5, ios::end); // Go to the 5th byte from the end
    inFile.get(ch); // Read the character at that position
    cout << "Character at position -5 from end: " << ch << endl;
    inFile.close(); // Close the file
    return 0;
}
```

Step 3: Modifying Data at Specific Positions

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
    // Open the file for reading and writing in binary mode
    fstream file("random_access_file.txt", ios::in | ios::out | ios::binary);
    if (!file) {
        cerr << "Failed to open the file for reading and writing: random_access_file.txt" << endl;
        return 1;
    }
    // Write a character at a specific position
    file.seekp(5, ios::beg); // Go to the 5th byte from the beginning
    file.put('X'); // Write 'X' at that position
    file.seekp(-3, ios::end); // Go to the 3rd byte from the end
    file.put('Y'); // Write 'Y' at that position
    file.close(); // Close the file
    return 0;
}
```

2. सीकवेनशियल ऍक्सेस फाईल (Sequential access file)

C++ मध्ये सीकवेनशियल ऍक्सेस फाईल हँडलिंग म्हणजे डेटा एका विशिष्ट क्रमाने वाचणे किंवा लिहिणे. साधारणतः फाईलमधील डेटा सुरुवातीपासून शेवटपर्यंत एका ओळीने किंवा ब्लॉकने वाचला किंवा लिहिला जातो. ifstream आणि ofstream क्लासेस वापरून हे साध्य केले जाते.

स्पष्टीकरण

1. डेटा लिहिणे (ofstream):

- `ofstream outFile("sequential_access_file.txt");`
 - फाईल उघडते आणि डेटा लिहिण्यासाठी सज्ज करते. जर फाईल अस्तित्वात नसेल तर ती तयार केली जाते. अस्तित्वात असेल तर तिची सामग्री मिटवली जाते.
- `outFile << "Line 1: Hello, World!" << endl;`
 - फाईलमध्ये ओळ लिहिते. प्रत्येक `<<` ऑपरेटरने डेटा फाईलमध्ये जोडला जातो.
- `outFile.close();`
 - फाईल बंद केली जाते.

2. डेटा वाचणे (ifstream):

- `ifstream inFile("sequential_access_file.txt");`
 - फाईल वाचण्यासाठी उघडते.
- `while (getline(inFile, line))`
 - फाईलमधील प्रत्येक ओळ वाचते आणि line स्ट्रिंगमध्ये साठवते. getline फंक्शन प्रत्येक ओळीनंतर newline कॅरॅक्टरवर थांबते.
- `cout << line << endl;`
 - वाचलेली ओळ कन्सोलवर प्रिंट करते.
- `inFile.close();`
 - फाईल बंद केली जाते.

सीकवेनशियल अॅक्सेस फाईल हँडलिंगचे फायदे (Benefits of Sequential access file handling)

- **सोपे आणि सोयीस्कर:** डेटा एका विशिष्ट क्रमाने वाचणे आणि लिहिणे सोपे आहे.
- **कमीत कमी ओव्हरहेड:** सॉफ्टवेअर आणि हार्डवेअर संसाधने कमीत कमी वापरतात.
- **सामान्य वापर:** कॉन्फिगरेशन फाईल्स, लॉग फाईल्स, टेक्स्ट डेटा स्टोरेज इत्यादींमध्ये याचा वापर केला जातो.

फाईलमध्ये डेटा लिहिणे (ofstream)

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
    ofstream outFile("sequential_access_file.txt");
    if (!outFile) {
        cerr << "Failed to open the file for writing: sequential_access_file.txt" << endl;
        return 1;
    }
    // फाईलमध्ये काही डेटा लिहा
    outFile << "Line 1: Hello, World!" << endl;
    outFile << "Line 2: C++ File Handling" << endl;
    outFile << "Line 3: Sequential Access" << endl;
    outFile.close(); // फाईल बंद करा
    return 0;
}
```

फाईलमधील डेटा वाचणे (ifstream)

```
#include <iostream>
```

```
#include <fstream>
using namespace std;
int main() {
    ifstream inFile("sequential_access_file.txt");
    if (!inFile) {
        cerr << "Failed to open the file for reading: sequential_access_file.txt" << endl;
        return 1;
    }
    string line;
    // फाईलमधील प्रत्येक ओळ वाचा
    while (getline(inFile, line)) {
        cout << line << endl; // ओळ प्रिंट करा
    }
    inFile.close(); // फाईल बंद करा
    return 0;
}
```

- **Important Program:**

1. **C++ program to copy the content of one file into another:**

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
    // Open the source file for reading
    ifstream sourceFile("source.txt");
    if (!sourceFile) {
        cerr << "Failed to open the source file." << endl;
        return 1;
    }
    // Open the destination file for writing
    ofstream destFile("destination.txt");
    if (!destFile) {
        cerr << "Failed to create the destination file." << endl;
        return 1;
    }
    char ch;
    // Read from the source file character by character and write to the destination file
    while (sourceFile.get(ch)) {
        destFile.put(ch);
    }
    // Close both files
    sourceFile.close();
    destFile.close();
    cout << "File copied successfully." << endl;
    return 0;
}
```

Output of the Program:

If source.txt is successfully opened and destination.txt is successfully created:

File copied successfully.

If source.txt cannot be opened:

Failed to open the source file.

If destination.txt cannot be created:

Failed to create the destination file.

Example:

Suppose source.txt contains the following text:

Hello, world!

After running the program, destination.txt will contain the same text:

Hello, world!

And the console output will be:

File copied successfully.

2 . C++ program to perform input output operations on binary files.

```
#include <iostream>
#include <fstream>
using namespace std;
// Structure to represent a student
struct Student {
    int rollNumber;
    char name[50];
    float marks;
};
int main() {
    // Writing data to a binary file
    {
        // Create an object of ofstream to write to a binary file
        ofstream outFile("students.bin", ios::binary);
        if (!outFile) {
            cerr << "Failed to open the file for writing." << endl;
            return 1;
        }
        // Array of students
        Student students[] = {
            {1, "John", 85.5},
            {2, "Alice", 90.0},
            {3, "Bob", 75.25}
        };
        // Write the array of students to the binary file
        outFile.write(reinterpret_cast<char*>(&students), sizeof(students));
        // Close the file
        outFile.close();
    }
    // Reading data from a binary file
    {
```

```
// Create an object of ifstream to read from the binary file
ifstream inFile("students.bin", ios::binary);
if (!inFile) {
    cerr << "Failed to open the file for reading." << endl;
    return 1;
}
// Array to store the read students
Student readStudents[3];
// Read data from the file into the array
inFile.read(reinterpret_cast<char*>(&readStudents), sizeof(readStudents));
// Close the file
inFile.close();
// Display the read data
cout << "Student Records:" << endl;
for (int i = 0; i < 3; ++i) {
    cout << "Roll Number: " << readStudents[i].rollNumber << endl;
    cout << "Name: " << readStudents[i].name << endl;
    cout << "Marks: " << readStudents[i].marks << endl;
    cout << endl;
}
}
return 0;
}
```

output

Student Records:

Roll Number: 1

Name: John

Marks: 85.5

Roll Number: 2

Name: Alice

Marks: 90

Roll Number: 3

Name: Bob

Marks: 75.25

3. C++ program to count the number of characters in a file:

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
    // Open the file for reading
    ifstream inFile("input.txt");
    if (!inFile) {
        cerr << "Failed to open the file." << endl;
```

```

    return 1;
}
// Variable to count the characters
int count = 0;
char ch;
// Read characters from the file and count them
while (inFile.get(ch)) {
    count++;
}
// Close the file
inFile.close();
// Display the count
cout << "Number of characters in the file: " << count << endl;
return 0;
}

```

Output Example

Suppose input.txt contains the following text:

Hello, world!

Here's how the program processes this input:

It opens input.txt and reads each character, including punctuation and spaces.

The characters are: H, e, l, l, o, ,, , w, o, r, l, d, !

The total number of characters is 13.

The console output will be:

Number of characters in the file: 13

If the file input.txt does not exist or cannot be opened, the console output will be:

Failed to open the file.

4. C++ program to find end of file.

```

#include <iostream>
#include <fstream>
int main() {
    // Open the file
    std::ifstream file("example.txt");

    // Check if the file is open
    if (!file.is_open()) {
        std::cerr << "Error opening the file.\n";
        return 1;
    }
    // Read the file until the end
    char ch;
    while (!file.eof()) {
        file.get(ch); // Read a character
        if (!file.eof()) {
            // Process the character
            std::cout << ch;
        }
    }
}

```

```
}  
// Close the file  
file.close();  
return 0;  
}
```

Example Output:

Suppose example.txt contains the following text:

Hello, world!

Here's how the program processes this input:

It opens example.txt and reads each character until the end of the file.

The characters are: H, e, l, l, o, , , w, o, r, l, d, !

These characters are printed to the console one by one.

The console output will be:

Hello, world!

If the file example.txt does not exist or cannot be opened, the console output will be:

Error opening the file.

संदर्भ:

1. <https://www.w3schools.com/cpp/>
2. <https://www.javatpoint.com/cpp-tutorial>
3. <https://www.javatpoint.com/cpp-files-and-streams>
4. <https://www.programiz.com/cpp-programming>
5. <https://www.programiz.com/cpp-programming/online-compiler/>
6. https://www.onlinegdb.com/online_cplusplus_compiler