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

अभियांत्रिकी आणि तंत्रज्ञान पदविका

शिक्षण पुस्तिका
(Learning Material)

PROGRAMMING IN C

(312303)

K-Scheme

प्रोग्रामिंग इन सी

संगणक अभियांत्रिकी गट
(के–स्कीम)

मराठी–इंग्रजी (द्विभाषिक) माध्यम
(अभियांत्रिकी व तंत्रज्ञानातील दुसरे सत्र पदविका)

शिक्षण पुस्तिका
(Learning Material)

प्रोग्रामिंग इन सी

PROGRAMMING IN C

(312303)

संगणक अभियांत्रिकी गट
(के–स्कीम)

मराठी–इंग्रजी (द्विभाषिक) माध्यम
(अभियांत्रिकी व तंत्रज्ञानातील दुसरे सत्र पदविका)

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

प्रोग्रामिंग इन सी
(312303)

मार्गदर्शक

प्रा. विजय नामदेवराव कुकरे
विभागप्रमुख, संगणक अभियांत्रिकी

प्रा. काळे गोरक्षनाथ भागवतराव
उपप्राचार्य व विभागप्रमुख, संगणक तंत्रज्ञान

लेखक

प्रा. बोऱ्हाडे एस.एस.
(अधिव्याख्याता, संगणक तंत्रज्ञान)

प्रा. वाकचौरे एस.एल.
(अधिव्याख्याता, संगणक तंत्रज्ञान)

प्रा. भाबड व्हि.एम.
(अधिव्याख्याता, संगणक तंत्रज्ञान)

प्रा. मुसळे डी.एम.
(अधिव्याख्याता, संगणक तंत्रज्ञान)

प्रा. घुगे जी.डी.
(अधिव्याख्याता, संगणक तंत्रज्ञान)

प्रा. शिंदे बी.बी.
(अधिव्याख्याता, संगणक तंत्रज्ञान)

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, विश्वसनिय 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 करीता 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 पूवक अभिनंदन करतो !

(प्रमोद नाईक)
संचालक
म. रा. तंत्र शिक्षण मंडळ, मुंबई.

अनुक्रमणिका

घटक -१

बेसिक्स ऑफ C प्रोग्रामिंग

(Basics of 'C' Programming)

गुण (१२)

विषय निष्पत्ती (Course Outcome): इनपुट /आउटपुट फंक्शन आणि अरिथेमॅटिक एक्सप्रेशन चा वापर करून C प्रोग्राम डेव्हलप करणे.

घटक निष्पत्ती (Unit Outcome):

१. दिलेल्या प्रॉब्लेम स्टेटमेंट साठी अल्गोरिदम लिहा.

२. C प्रोग्रामिंग साठी बिल्डिंग ब्लॉक अडेंटीफाय करा.

३. C प्रोग्राम डेव्हलप करण्यासाठी कॉन्स्टंट, व्हेरिएबल, डेटा टाइप या मूलभूत गोष्टींचा वापर करा.

४. प्रिंट एफ आणि स्कॅन फ या फंक्शन चा वापर करून C प्रोग्राम लिहा.

५. अर्थमॅटिक ऑपरेटर्स आणि बिटवाईज ऑपरेटर चा उपयोग करून C प्रोग्राम लिहा.

१.१ अल्गोरिदमची मूलभूत तत्त्वे(Fundamentals of algorithms):

गणना किंवा इतर समस्या सोडवण्याच्या ऑपरेशन्समध्ये पाळल्या जाणाऱ्या नियमांचा एक संच" किंवा "गणितीय समस्या सोडवण्याची प्रक्रिया मर्यादित टप्प्यांमध्ये ज्यामध्ये वारंवार पुनरावृत्ती होणारी ऑपरेशन्स समाविष्ट असतात".याला अल्गोरिदम असे म्हणतात. अल्गोरिदम विशिष्ट समस्येचे निराकरण करण्यासाठी मर्यादित चरणांच्या क्रमाचा संदर्भ देते.तुम्ही काय साध्य करू इच्छिता, त्यानुसार अल्गोरिदम सोपे आणि अवघड असू शकतात.

एक अल्गोरिदम तयार करण्यासाठी, खूप विचारांचे आणि योजनांची आवश्यकता असते. त्यासाठी काही मुख्य घटक आहेत:

- सुसंगतता (Correctness): अल्गोरिदम सुसंगत असणे म्हणजेच त्याचे काम सारखे करताना त्याचे उत्तर बरोबर असते. सुसंगततेची खात्री घेण्यासाठी विचारांचे आणि प्रमाणांचे वापर केला पाहिजे.
- इफिशियन्सी (Efficiency): अल्गोरिदम इफिशियन्सी म्हणजेच तो खूप वेगवेगळ्या प्रकारे संपन्न होते. समस्येचे विचार करण्यासाठी अल्गोरिदम किती वेगवेगळं काम करते ते दिसते.
- वापरकर्ता-मान्यता (User-Friendliness): अल्गोरिदम युझरसाठी सुविधाजनक, सोपं आणि समजूत्याचे होणं महत्त्वाचे आहे.

- क्लॅरिटी (Clarity): अल्गोरिदम लिहिण्यासाठी वापरलेला क्लॅरिटी सोपं, स्पष्ट आणि आधारभूत असली पाहिजे.

एक अल्गोरिदम तयार करण्यासाठी, आपल्याला समस्येचे सुसंगत आणि इफिशियन्सी सोडावे, पर्याय वाचलेला, विचारांचे वापर करावे विविध प्रकारच्या डेटा संग्रहणांसाठी उपयुक्त डेटा संरचनांसाठी ठरवावे आणि क्लॅरिटी सोडावे आणि सुसंगततेची खात्री घेणे आवश्यक आहे.

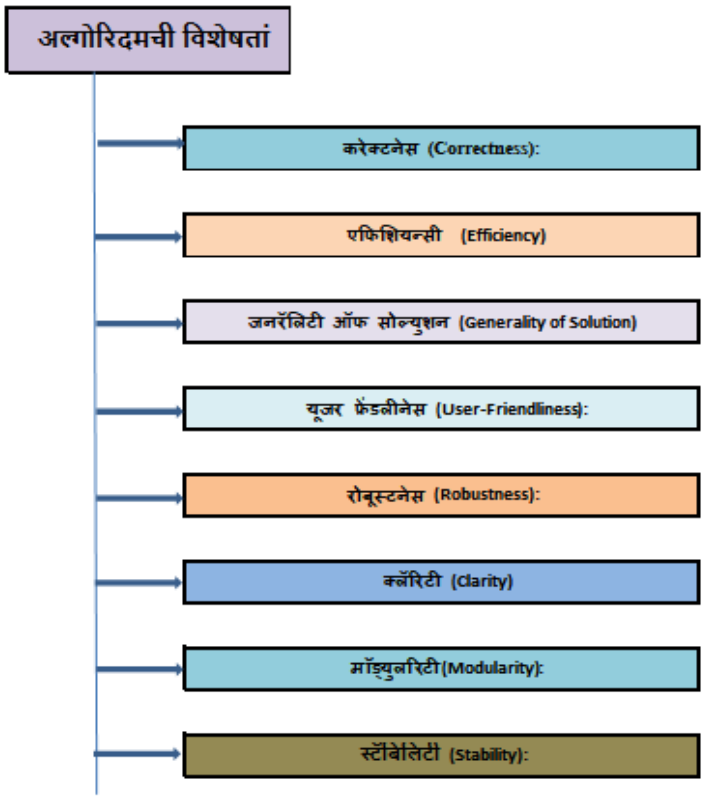
१.१.१. अल्गोरिदमची धारणा(Notion of algorithm):

८२५ AD मध्ये एका पर्शियन गणितज्ञाने "अल-जबर वाल मुकाबला" हे पुस्तक लिहिले, ज्यामध्ये विशिष्ट श्रेणीतील समस्यांचे निराकरण करण्यासाठी सामान्य पद्धती आहेत. त्यांनीच प्रथमच अशा पद्धती सादर केल्या. त्याचे नाव अबू जाफर मोहम्मद इब्न मुसा अल खोवारीस्मी होते. अल्गोरिदम हा शब्द त्याच्या नावावरून आला आहे.

अल्गोरिदम म्हणजेच कोणत्याही कार्याचे सुसंगत आणि इफिशियन्सी सारखे काम करणारे अनुक्रम. एक अल्गोरिदम तयार करण्यासाठी, त्याचे सारखे काम करणारं उत्तर दिला पाहिजे. असे करण्यासाठी, तुमच्या आपल्या मागच्या समस्येचे आणि कामाचे स्वरूप ठरवावे. त्यानुसार, तुम्हाला एक सुसंगत विधान ठरवावे आणि त्यामुळे तुमचे उत्तर सापडेल.

एक अल्गोरिदम सुसंगत, स्थानिकपूर्ण, आणि स्पष्ट असावे हे महत्त्वाचे आहे. असे केला पाहिजे की, त्याचे उपयोग कोणत्याही स्थितीसाठी करण्यात यावे शकते. एक सुसंगत आणि स्थानिकपूर्ण अल्गोरिदम किंवा फंक्शनवाही आपल्याला केवळ सहाय्यकारक ठरून देते, परंतु त्याचे वापर एक विचारांचे आणि संकल्पांचे प्रमाण होते.

अल्गोरिदमची विशेषतां (Characteristics of an Algorithm)



आकृती १ अल्गोरिदमची विशेषतां

अल्गोरिदमची विशेषतांची मुख्य प्रमुखता:

- **करेक्टनेस (Correctness):** अल्गोरिदम सुसंगत असल्याचे म्हणजेच त्याचे काम सारखे करताना त्याचे उत्तर बरोबर असते. सुसंगततेची खात्री घेण्यासाठी विचारांचे आणि प्रमाणांचे वापर केला पाहिजे.
- **इफिशियन्सी (Efficiency):** अल्गोरिदम इफिशियन्सीचे म्हणजेच तो खूप वेगवेगळ्या प्रकारे संपन्न होते. समस्येचे विचार करण्यासाठी अल्गोरिदम किती वेगवेगळं काम करते ते दिसते.
- **जनरॅलिटी ऑफ सोल्युशन (Generality of Solution):** एक अल्गोरिदम एकाच समस्येसाठी मात्र असा नसल्यास, तो विविध समस्यांसाठी सामान्य सोडावे किंवा सुलभपणे सापडवे किंवा लागणारं सोडावे शकते.
- **यूजर फ्रेंडलीनेस (User-Friendliness):** अल्गोरिदम युझरसाठी सुविधाजनक, सोपं आणि समजूत्याचे होणं महत्त्वाचे आहे.

- **रोबूस्टनेस (Robustness):** अल्गोरिदम विविध प्रकारच्या परिस्थितीसाठी समर्थ होणं महत्त्वाचे आहे. त्यामुळे, कोणत्याही परिस्थितीत असलेल्या अवांछित बदलांसाठी सज्ज राहावे शकते.
- **क्लॅरिटी (Clarity):** अल्गोरिदम सोपं, स्पष्ट आणि आधारभूत असला पाहिजे. सर्व कार्यांची वाचने वाचनं कसंच सोपी असावी.
- **मॉड्युलरिटी (Modularity):** अल्गोरिदम तात्काळीन बदलांसाठी सुविधेसाठी इफिशियन्सी विभाजलेला असला पाहिजे. त्यामुळे पुनरावलोकन किंवा सुधारणे सोपी व इतर भागांकिंवा प्रक्रियांसह केली जाऊ शकतात.
- **स्टॅबिलिटी (Stability):** अल्गोरिदम संबंधित स्थितीत स्थिर राहावे आवश्यक आहे, असे किंवा विशेष प्रकारच्या स्थितीत स्थिरता ठेवणं महत्त्वाचे आहे.

अल्गोरिदमची विशेषतांचे योग्यता असलेले अल्गोरिदम त्याचे उपयोग आपल्या समस्येचे स्थानिक विचार करण्यासाठी तयार करण्यात आणि वापरण्यात येतात.

दोन नंबरची बेरीज चा अल्गोरिदम

१.प्रारंभ: सांगण्याचे संकेत, फ्लोचार्ट सुरुवात करण्यासाठी.

२.एंटर (अंक1): पहिला अंक घेणे.

३.एंटर (अंक2): दुसरं अंक घेणे.

४.जोडणे (अंक1 + अंक2): दोन अंकांचे योजना करणे.

५.समाप्त: सांगण्याचे संकेत, प्रक्रिया समाप्त करण्यासाठी.

➢ फ्लोचार्टची विशेषतां (Characteristics of an Flowchart)

फ्लोचार्ट (Flowchart) हे एक ग्राफिकल प्रतिष्ठान आहे. ज्यात विभिन्न प्रक्रियां, निर्णये आणि क्रियां दर्शविलेल्या संकेतांचे वापर करून कामाची प्रक्रिया आणि संरचना दर्शविली जाते. फ्लोचार्ट एक निर्दिष्ट गोष्टीच्या सिद्धांतांचे अनुसरण करून किंवा व्यवस्थापनाच्या अनुसरणांसाठी डिझायन केलेली आहे. या फ्लोचार्टमध्ये वापरलेल्या संकेतांची विशिष्ट चिन्ह आहेत, ज्यांना ग्राफिकल भाषेत स्थान देण्यात आलाय.

फ्लोचार्टमध्ये वापरलेल्या प्रमुख संकेतांचे मराठीतील चिन्ह आणि त्यांचे वर्णन खालीलप्रमाणे आहे:

प्रारंभ / शुरुआत आणि समाप्त / अंत: फ्लोचार्टमध्ये कार्याचे सुरुवातीला आणि समाप्तीसाठी वापरलेला चिन्ह. आपल्या प्रोग्राम आणि प्रक्रियांच्या सुरुवात आणि समाप्तीसाठी.

प्रक्रिया / क्रिया: फ्लोचार्टमध्ये एके किंवा एकाधिक क्रिया किंवा प्रक्रियांचे सुरु ठेवणे.

निर्णय / शेरा: कोणत्याही निर्णय किंवा शेरा केल्याचे सुचले.

शाखा / शाखांतर: विशिष्ट स्थितीतील निर्णयांचे फ्लॉ केलाय.

सांगण्याचे / इनपुट आणि आउटपुट संकेत: डेटा किंवा इनपुट आणि आउटपुट दर्शविलेला चिन्ह.

यादी / अनुक्रम / पूर्ण सांगणं: संबंधित प्रक्रियांचे आणि क्रियांचे संदर्भ दर्शविलेला चिन्ह.

फ्लोचार्टमध्ये वापरलेल्या संकेतांचे योग्य अनुसरण करुन, तुम्ही प्रक्रिया किंवा प्रोग्राम डिझायन करू शकता.

सामान्य फ्लोचार्ट चिन्हे :

- **प्रक्रिया चिन्ह(Process Symbol):**

प्रक्रिया चिन्ह, ज्याला क्रिया चिन्ह देखील म्हणतात, एक आयत आहे. हे एक प्रक्रिया, क्रिया किंवा फंक्शन दर्शवते. फ्लो चार्ट सिम्बॉलॉजीमध्ये हे सर्वात जास्त वापरले जाणारे चिन्ह आहे. आयत एका मोठ्या प्रक्रियेमध्ये एकल पायरी किंवा संपूर्ण उप-प्रक्रिया दर्शवू शकते.

- **प्रारंभ आणि समाप्ती चिन्ह (Start and End Symbol):**

टर्मिनेटर चिन्ह प्रक्रिया/प्रणालीचे प्रारंभ आणि शेवटचे बिंदू चिन्हांकित करते. चिन्हासह, "प्रारंभ" किंवा "समाप्त" हा शब्द अंडाकृती आकारात नमूद केला आहे. हे वर्कफ्लोची सुरुवात किंवा अंतिम परिणाम आणि मार्गाचे संभाव्य परिणाम सूचित करते. या चिन्हाला टर्मिनेटर चिन्ह असेही संबोधले जाते.

- **निर्णय चिन्ह(Decision Symbol):**

हे डायमंड-आकाराचे फ्लोचार्ट चिन्ह उत्तर देण्याच्या प्रश्नाला सूचित करते, सहसा होय/नाही किंवा खरे/खोटे. फ्लोचार्ट सहसा उत्तर किंवा त्यानंतरच्या परिणामांवर अवलाबून वेगवेगळ्या शाखांमध्ये विभाजित होतो. निर्णय बॉक्सचा परिणाम प्रक्रियेच्या फंक्शनप्रवाहातील पुढील चरण निर्धारित करतो. डायमंडच्या वेगवेगळ्या बिंदूंमधून वेगवेगळ्या निर्णयांचे प्रतिनिधित्व करणाऱ्या रेषा निघतात.

- **फ्लोलाइन चिन्ह(Flowline Symbol):**

फ्लोलाइन प्रक्रिया कोणत्या दिशेने वाहते ते दर्शवते. हे प्रक्रियेची दिशा दाखवते आणि फ्लोचार्टमधील दोन ब्लॉक्सना जोडते. प्रक्रियेच्या प्रवाहाच्या दिशेने निर्देशित करून ते बाण म्हणून दर्शविले जाते. फ्लोलाइन्स आणि पृष्ठ कनेक्टर प्रतिनिधी आकारांमधील संबंध दर्शवतात.

- **पृष्ठावरील चिन्ह (onpage Symbol)**

पृष्ठावर वेगळे घटक जोडण्यासाठी कनेक्टर चिन्ह अवघड फ्लोचार्टमध्ये वापरले जाते.

- **ऑफ-पेज कनेक्टर/लिंक चिन्ह(onpage Connector):**

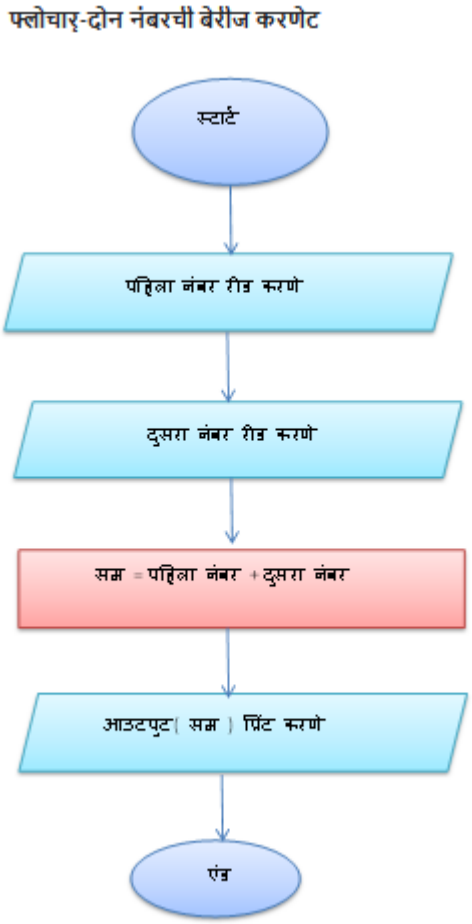
ऑफ-पेज कनेक्टर अवघड फ्लोचार्टमध्ये अनेक पृष्ठांवर वेगळे घटक जोडण्यासाठी वापरले जाते.

इनपुट/आउटपुट चिन्ह(Input Output Symbol):

समांतरभुज चिन्हाला डेटा चिन्ह असेही संबोधले जाते. इनपुट/आउटपुट चिन्ह सूचित करते की डेटा इनपुट किंवा आउटपुटसाठी उपलब्ध आहे, तसेच वापरलेल्या किंवा व्युत्पन्न केलेल्या संसाधनांचे प्रतिनिधित्व करतो. वर्कफ्लोमध्ये या टप्प्यावर माहिती आवश्यक असल्याचे सूचित करते.

पूर्वनिर्धारित प्रक्रिया चिन्ह(Predefined Process):

पूर्वनिर्धारित प्रक्रिया चिन्ह अशी प्रक्रिया किंवा फंक्शनप्रवाह दर्शवते ज्यासाठी उच्च स्तरीय तपशील आवश्यक आहे. पूर्वनिर्धारित प्रक्रियांसाठी अतिरिक्त माहिती इतरत्र प्रदान केली जाते किंवा प्रक्रिया सामान्यतः समजली जाते.



१.१.२ असाइनमेंट स्टेटमेंट आणि बेसिक कंट्रोल स्ट्रक्चर साठी वापरलेली नोटेशन(Notations used for assignment statements and basic control structures):

अल्गोरिदम लिहिताना, काही सामान्य नियमांची अनुसरण करणं महत्त्वाचे आहे त्यामुळे त्याची सुसंगतता आणि इफिशियन्सी दिला जाते.

- शीर्षक (Title): तुमच्या अल्गोरिदमचे एक शीर्षक दिला पाहिजंट. शीर्षक संक्षेपात्मक असावे आणि त्यामुळे अल्गोरिदम वाचण्यात आणि समजण्यात सोपं होईल.
- वर्णन (Description): अल्गोरिदमचे संक्षेपात्मक वर्णन अल्गोरिदम लेखकांकरांनी दिला पाहिजे. कोणतेही विशिष्टता येईल त्या बरोबर त्याचे संक्षेपात्मक वर्णन करून दिला पाहिजे.
- प्रारंभिक ठरवात (Initialization): अल्गोरिदम सुरुवातीला आवश्यक ठरवताना विवेचला पाहिजे. तुमचे काम सुरुवातीला कोणत्याही स्थितीमध्ये असल्याचे म्हणजेच कोणत्याही परिणामांमध्ये असल्याचे त्यानुसार सुसंगतपणे ठेवता.
- प्रमाणांक (Input): तुमच्या अल्गोरिदममध्ये कोणतेही प्रक्रिया सुरुवातीला तुम्ही कोणत्याही प्रमाणांक (उपाययुक्ततेने परिणामांमध्ये वापरण्यात येणारं वस्तुंचे) प्रविष्ट करून त्यानुसार काम करावे.

- प्रमाणांक नियोजन (Variable Initialization): तुमचे अल्गोरिदम कोणत्याही प्रमाणांक वापरून काम करण्यासाठी त्याचे निर्दिष्ट आणि स्थानिकपूर्ण आहे. त्या प्रमाणांकांचे सुरुवातीला त्याचे मूल्य स्थापित करावे.
- आउटपुट(Output):

कीवर्ड्स म्हणजेच त्यातील विशिष्ट युझरकरिता संदर्भित केलेल्या वाक्यांतील शब्दांच्या संच असतात. या कीवर्ड्समध्ये चांगला सुसंगत अनुसरण करण्यासाठी आणि प्रोग्राम वाचण्यासाठी किंवा कोडच्या स्ट्रक्चरच्या साठी उपयुक्त आहे.

अशी काही कीवर्ड्स:

- **सुरुवात (Start):** प्रोग्राम किंवा अल्गोरिदमचे सुरुवात करण्यासाठी वापरलेला कीवर्ड.
- **काम (Do)**: काही काम किंवा प्रक्रिया सुरुवात करण्यासाठी वापरलेला कीवर्ड.
- **काम समाप्त (End):** प्रोग्राम किंवा अल्गोरिदमचे समाप्त होऊन किंवा कोडचे अंत होऊन वापरलेला कीवर्ड.
- **समाप्त जर (If):** कोणत्याही शर्तानुसार किंवा निर्दिष्ट स्थितीत फंक्शन करण्यासाठी वापरलेला कीवर्ड.
- **जर इतरप्रमाणांक (Else If):** जर किंवा इतरप्रमाणांक दोन्ही दिलेल्या शर्तांनुसार फंक्शन करण्यासाठी वापरलेला कीवर्ड.
- **इतरप्रमाणांक अन्य (Else):** कोणत्याही इतर प्रमाणांक किंवा शर्तेच्या समाप्तीसाठी वापरलेला कीवर्ड.
- **अन्य निर्देश (Instruction):** कोणत्याही फंक्शन, अद्यतित निर्देश किंवा क्रिया वापरण्यासाठी वापरलेला कीवर्ड.
- **निर्देश परिणाम (Output):** परिणाम दाखवण्यासाठी वापरलेला कीवर्ड.
- **परिणाम चर (Variable):** एक

१.२ इंट्रोडक्शन टू सी प्रोग्रामिंग (Introduction to 'C'programming):

महान संगणक शास्त्रज्ञ डेनिस रिची यांनी १९७२ मध्ये बेल लॅबोरेटरीजमध्ये 'सी' नावाची नवीन प्रोग्रामिंग लँग्वेज तयार केली. 'ALGOL', 'BCPL' आणि 'B' प्रोग्रामिंग भाषांमधून ही तयार केले गेली आहे. 'सी' प्रोग्रामिंग लँग्वेजमध्ये या भाषांची सर्व वैशिष्ट्ये आणि इतर अनेक अतिरिक्त संकल्पनांचा समावेश आहे.

'C' ही एक शक्तिशाली प्रोग्रामिंग लँग्वेज आहे.ही UNIX ऑपरेटिंग सिस्टमशी संबंधित लँग्वेजआहे. युनिक्स ऑपरेटिंग सिस्टीमचाही बहुतांश भाग 'C' मध्ये कोड केलेला आहे. सुरुवातीला 'C' प्रोग्रामिंग हे UNIX ऑपरेटिंग

सिस्टीमपुरते मर्यादित होते.पण जसजसे ते जगभर पसरू लागले, तसतसे ते व्यावसायिक बनले आणि क्रॉस-प्लॅटफॉर्म सिस्टमसाठी अनेक कंपायलर तयार करण्यात आले.

C ला कंपाइल्ड लँग्वेज देखील म्हणतात. याचा अर्थ असा की एकदा तुम्ही तुमचा C प्रोग्राम लिहिला की,तो प्रोग्राम कंप्युटर मध्ये रन करता येण्यासाठी (एक्झिक्युटेबल) तुम्ही तो आधी C कंपायलरद्वारे रन केला पाहिजे. C प्रोग्राम हा मानवी-वाचनीय फॉर्म आहे. कंपाइलरमधून बाहेर पडणारा मशीन-वाचनीय आणि एक्झिक्युटेबल फॉर्म आहे. याचा अर्थ असा आहे की C प्रोग्राम लिहिण्यासाठी आणि चालविण्यासाठी, C कंपाइलर असणे आवश्यक आहे.

'C' विविध ऍप्लिकेशन्समध्ये मोठ्या प्रमाणात वापरली जाणारी लँग्वेज आहे. ही एक सोपी लँग्वेज आहे आणि जलद अंमलबजावणी प्रदान करते.'C' ही एक स्ट्रक्चर्ड प्रोग्रामिंग लँग्वेज आहे.यात प्रोग्राम विविध मॉड्यूल्समध्ये विभागलेला आहे.सर्व मॉड्युल एकत्रितपणे मिळून एक 'C' प्रोग्राम तयार होतो.यामुळे चाचणी, देखभाल आणि डीबगिंग प्रक्रिया सुलभ होते..

C प्रोग्रामिंग चे फीचर्स (Features of C Language)

- सिंपल
- फास्ट स्पीड
- मशीन पोर्टेबल
- पुनरावृत्ती
- स्ट्रक्चर्ड प्रोग्रामिंग लँग्वेज
- पॉइंटर्स
- एक्स्टेनसिबल
- मेमोरी मॅनेजमेंट
- समृद्ध लायब्ररी

C चे उपयोग (Uses of C)

- सिस्टम ऍप्लिकेशन्स विकसित करण्यासाठी C लँग्वेज वापरली जाते .
- ब्राउझर आणि त्यांचे एक्स्टेन्शन विकसित करण्यासाठी C लँग्वेज वापरली जाते .
- डेस्कटॉप ऍप्लिकेशन्स विकसित करण्यासाठी C लँग्वेज वापरली जाते.
- डेटाबेस विकसित करण्यासाठी C लँग्वेज वापरली जाते . MySQL हे डेटाबेस सॉफ्टवेअर आहे जे 'C' वापरून तयार केले आहे.
- एम्बेडेड सिस्टममध्ये सी' लँग्वेज मोठ्या प्रमाणावर वापरली जाते.
- IoT ऍप्लिकेशन्समध्ये C लँग्वेज वापरली जाते.

- ऑपरेटिंग सिस्टम विकसित करण्यासाठी C लँग्वेज वापरली जाते . मायक्रोसॉफ्ट ची विंडोज ऑपरेटिंग सिस्टीम 'C' लँग्वेज वापरून विकसित केली गेली आहे. मोबाईल फोन व डेस्कटॉप ऑपरेटिंग सिस्टीम विकसित करण्यासाठी C चा वापर केला जातो.

१.२.१. जनरल स्ट्रक्चर ऑफ सी प्रोग्राम (General structure of ' C' program)

सी प्रोग्रामचे मूल संरचना एका विशिष्ट स्वरूपात विभाजित केलेली आहे, ज्यामुळे हे प्रोग्राम अधिसूचित केला जाते आणि हे किंवा आपला प्रोग्राम उपयुक्तपणे कंपायल करण्यात आणि क्रियान्वित करण्यात सफळ असण्यात मदत होते. एक शास्त्रीय आणि सुरक्षित आपल्या सी प्रोग्राममध्ये अनुसरण करणे पाहिजे. त्या प्रकारे, अचुकपणे, अनुपयोगी किंवा विचारशील त्रुटिजवळ कोडसह आपले काम करणे सोपे आणि सुरक्षित आहे. त्यासाठी, तुमचे सी प्रोग्राम खासगीतच दिलेल्या 6 भागांच्या ठरावात चालवावे. सी प्रोग्रामचे सामान्य संरचना आपल्याला प्रोग्रामचे स्थान असते आणि फंक्शन निर्दिष्ट करायचे असते. एक सामान्य सी प्रोग्रामचे स्थान आणि संरचना खालीलप्रमाणे आहे:

सी प्रोग्राममध्ये कार्यान्वित होण्यासाठी 6 मौखिक स्टेप आहेत. स्टेप खालीलप्रमाणे दिलेले आहे:

- **डॉक्युमेंटेशन (Documentation)**:

  डॉक्युमेंटेशन चे काम मुख्यत्वेने प्रोग्राम वापरण्यात आलेली लँग्वेज, कार्ये, किंवा आवश्यक माहिती प्रदान करणं आहे. एक सुंदर आणि सुसंगत डॉक्युमेंटेशन प्रोग्राम वापरण्यात योग्य आणि त्याचे समजतात.

- **प्री प्रोसेसर सेक्शन (Preprocessor Section):**

  प्री प्रोसेसर सेक्शन मुख्यप्रक्रिया सुरू करण्यासाठी आवश्यक फाईले समाविष्ट करणारे डायरेक्टिव्ह असतात. उदाहरणार्थ, #include डायरेक्टिव्ह स्टॅन्डर्ड इनपुट आणि आउटपुट फाईल्स समाविष्ट करण्यासाठी वापरले जाते.

- **डेफिनेशन (Definitions):**

  डेफिनेशन कोडाचे मूलप्रकारे परिभाषित करणं. त्यात #define डायरेक्टिव्ह वापरून संख्या किंवा स्ट्रिंग्सला एक नाव **देण्यात येते.**

- **ग्लोबल डिक्लेरेशन (Global Declaration):**

  ग्लोबल डिक्लेरेशन मध्ये फंक्शन्स, मुद्रांकन, आणि इतर महत्वाचे डेटा स्टोर केलेला आहे, ज्याचे उपयोग प्रोग्राम तयार करण्यासाठी सर्व भागांमध्ये केला जातो.

- **मेन फंक्शन (Main() Function):**

  main() फंक्शन हे प्रोग्रामचे मुख्य फंक्शन असते. येथे प्रोग्राम सुरू होतो.

- **सब-प्रोग्राम (Sub Programs):**

सब-प्रोग्राम मध्ये विशिष्ट कार्ये आणि त्यांची घोषणां असतात. सब-प्रोग्राम main() फंक्शन ला मदत करण्यासाठी वापरली जातात.

या प्रक्रियाचे अनुसरण करून, एक सुरक्षित, स्थिर आणि सुसंगत सी प्रोग्राम तयार करणे संभाव होईल.

सी प्रोग्रामचे सामान्य संरचना आपल्याला प्रोग्रामचे स्थान देवायचे आणि फंक्शन निर्दिष्ट करायचे. एक सामान्य सी प्रोग्रामचे स्थान आणि संरचना खालीलप्रमाणे आहे:

```
#include <stdio.h>          //  हेडर फाईल्स स्टॅन्डर्ड इनपुट आणि आउटपुट हेडर फाईल
int main()                          // मुख्य फंक्शन (main function)
{
        return 0;       // सुसंपन्न शोध निश्चित करण्यासाठी 0 रिटर्न करा
}
```

या संरचनेत, हे महत्वाचे आहे:

हेडर फाईल्स (Header Files): #include डायरेक्टिव्ह वापरुन हेडर फाईल्स समाविष्ट केल्यात. युझर नुसार आवडलेले हेडर फाईल्स समाविष्ट करायचे. #include <stdio.h> हे एक उदाहरण आहे ज्याचे उपयोग printf आणि scanf यांसाठी केलेला जाते.

मुख्य फंक्शन (Main Function): सर्व C प्रोग्राममध्ये एक मुख्य फंक्शन असतो ज्याचे नाव main आहे. प्रोग्राम चालविण्याचे प्रारंभ तुमच्या कोडाचे येथे होते आणि त्याचे संरचनांतर return 0; म्हणून सुसंपन्न होते

उदाहरण , "Hello World" प्रोग्राम:

```
#include <stdio.h>
int main()
{
printf("Hello World!\n");
return 0;
}
```

१.२.२ हीडर फाईल (Header file)

हेडर फाईल हे सी प्रोग्राममध्ये वापरण्यात आलेली फाईले आहेत, ज्यातून सिस्टम युझरना किंवा इतर प्रोग्राम विकसकांना आवश्यक फंक्शन, मेनकिंड, वेळ, तसेच इतर वस्तूंसाठी सर्व्हिसेस आणि सुविधांची माहिती मिळते. हेडर फाईल मुख्यत्वेने .h या विस्ताराने जाणलेल्या फाईलें आहेत.

- **stdio.h: इनपुट आणि आउटपुट संबंधित फंक्शनसाठी.**

  #include <stdio.h>

- **stdlib.h: सामान्य प्रणालीसाठी अनिवार्य फंक्शनसाठी.**

#include <stdlib.h>

- **math.h: गणितीय फंक्शनसाठी.**

  #include <math.h>

- **string.h: स्ट्रिंग प्रोसेसिंगसाठी फंक्शन.**

  #include <string.h>

- **time.h: समय संबंधित फंक्शनसाठी.**

  #include <time.h>

या हेडर फाईलें युझरना अनेक प्रकारे मदत करतात, आणि त्यांचे सर्व्हिसेस आणि फंक्शन्स प्रोग्राम विकसकांना शीघ्रपणे विकसित करण्यात मदत करतात.

१.२.३ मेन फंक्शन ('main ()' function)

main() फंक्शन चे स्वरूप खालीलप्रमाणे आहे:

main() फंक्शन हे सी प्रोग्राममध्ये मुख्य फंक्शन असते. याने प्रोग्राम ची सुरुवात होते आणि प्रोग्रामचे संचालन सुरू होते आणि समाप्त होते.

```
int main() {
        // तुमचे कोड येथे लिहा
   return 0; // सुसंपन्न शोध निश्चित करण्यासाठी 0 रिटर्न करा
}
```

येथे,

int: या कार्याचे परिणामानुसार पूर्णांक रिटर्न करतात. 0 म्हणजेच फंक्शन सुसंपन्नपूर्वक समाप्त झाले आहे.

main(): हे आपला मुख्य फंक्शन आहे, ज्याचे प्रोग्राम प्रथम विचारलाय त्याचे निष्कर्षण करते.

{ }: एक कोड ब्लॉकचे सुरुवात आणि समाप्ती करते. आपला सर्व कोड येथे असते.

आपला खालीलप्रमाणे कोड लिहू शकता:

```
#include <stdio.h>
int main() {
    Printf ("मुख्य फंक्शन सुरू होते!\n");
   return 0;
}
```

जेव्हा आपला सी प्रोग्राम चालवतो, पहिला main() फंक्शन प्रविष्ट केला जाते आणि त्यानंतर तुमचे कोड समाप्त होते.

१.३ फंडामेंटल कन्स्ट्रक्ट ऑफ सी (Fundamental constructs of 'C'):

C प्रोग्रामिंगमध्ये मौलिक संरचनांमध्ये काही मुख्य घटक आहेत. या मुख्य घटकांची मदत करताना, तुम्ही C प्रोग्रामिंगमध्ये कोड लिहण्यासाठी वापरलेल्या मौलिक संरचनांचे अध्ययन करू.

- **डेटा टाइप्स (Data Types):**

  C मध्ये विविध प्रकारच्या डेटा टाइप्स आहेत, जसे की int (पूर्णांक), float (दशांक), char (वर्ण), double (दशांक संख्या), इत्यादी.

```
int age = 25;
float salary = 50000.50;
char grade = 'A';
```

- **वेरिएबल्स (Variables):**

  वेरिएबल्स हे माहिती संग्रहणासाठी वापरलेल्या नावे आहेत.

```
int count;
float price;
char initial;
```

- **ऑपरेटर्स (Operators):**

  ऑपरेटर्स हे विविध गणनासाठी वापरलेल्या संकेतांचे वापर करतात, जसे की +, -, *, / , % .

```
int sum = a + b;
float result = x / y;
int remainder = a % b;
```

- **कंट्रोल संरचना (Control Structures):**

  कंट्रोल संरचनांमध्ये if, else, switch, while, for यांच्या मदतीने तुम्ही निर्णय आणि पुनरावलोकन करू शकता.

```
if (condition)
{
                    // क्रिया 1
} else
{
                    // क्रिया 2
}
for (int i = 0; i < 5; i++)
```

```
{
                    // क्रिया
}
```

- **फंक्शन्स (Functions):**

  फंक्शन्स हे विशिष्ट कार्ये करणारे भाग आहेत जे कोडमध्ये पुनरावृत्ति करून काम करतात.

```
// फंक्शन डिफायन करणे
int add(int a, int b)
 {
   return a + b;
}
                    // फंक्शन वापरणे
int result = add(5, 3);
```

- **मेन () फंक्शन (Main Function):**

  main() फंक्शन हे प्रोग्राममध्ये मुख्य फंक्शन असते. येथे प्रोग्रामचे संचालन सुरू होते आणि समाप्त होते.

```
int main() {
                    // तुमचे कोड येथे लिहा
   return 0;        // सुसंपन्न शोध निश्चित करण्यासाठी 0 रिटर्न करा
    }
```

या मौलिक संरचनांचे वापर करून, तुम्ही C प्रोग्राम लिहू शकता आणि प्रोग्रामिंग आणि लॉजिक विकसित करू शकता.

१.३.१ कॅरेक्टर सेट (Character set)

C प्रोग्रामिंगमध्ये वापरलेला वर्ण संच ASCII (अमेरिकन स्टँडर्ड कोड फॉर इन्फॉर्मेशन इंटरचेंज) वर्ण एन्कोडिंगवर आधारित आहे. ASCII ही एक प्रमाणित वर्ण एन्कोडिंग योजना आहे जी अक्षरे, अंक, विरामचिन्हे आणि नियंत्रण वर्णांसह विविध वर्णांना अद्वितीय संख्यात्मक व्हॅल्यू नियुक्त करते.

- **अक्षरे (अपरकेस आणि लोअरकेस)(Alphabets (uppercase and lowercase):):**

  अप्परकेस अक्षरे: A-Z (ASCII मध्ये 65-90)

  लोअरकेस अक्षरे: a-z (ASCII मध्ये 97-122)

- **अंक(Digit)**

0-9 (ASCII मध्ये 48-57)

- **विरामचिन्हे(Punctuation Marks)**

  ! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ ` { | } ~

- **व्हाइटस्पेस वर्ण(Whitespace Characters)**

  जागा(Space) (ASCII मध्ये 32)

  टॅब(Tab) (ASCII मध्ये 9)

  न्यूलाइन(Newline) (ASCII मध्ये 10)

  कॅरेज रिटर्न(Carriage return)(ASCII मध्ये 13)

- **कंट्रोल कॅरेक्टर (Control Characters)**

32 पेक्षा कमी ASCII मूल्यांसह विविध नियंत्रण वर्ण.

सी प्रोग्रामिंग स्ट्रिंग, वर्ण आणि चिन्हे तयार करण्यासाठी आणि हाताळण्यासाठी या ASCII वर्णांवर खूप अवलाबून आहे.

१.३.२ टोकन (tokens)

टोकन (Tokens) हे C प्रोग्रामिंगमध्ये वापरलेले मौलिक अंश आहे, जे कोड संरचनेतून तयार होतात. एका C प्रोग्राममध्ये टोकन संच असते आणि त्यामुळे कंपायलरला कोड समजावतो.

C प्रोग्रामिंग मध्ये वापरलेले टोकन त्यांच्या कामानुसार सहा प्रकारांत वर्गीकृत केले जातात:

- कीवर्ड (Keywords):
- आयडेंटिफायर (Identifiers):
- कॉन्स्टन्स (Constants):
- स्ट्रिंग्स (Strings):
- स्पेशल सिम्बॉल (Special Symbols):
- ऑपरेटर्स (Operators):

ते पुढील प्रमाणे अभ्यास करू.

१.३.३ कीवर्ड (keywords)

कीवर्ड्स (Keywords) हे प्रोग्रामिंग लँग्वेज मधील पूर्वनिर्धारित किंवा आरक्षित शब्द आहेत. प्रत्येक कीवर्ड एक विशिष्ट फंक्शन करण्यासाठी डिझाइन केलेला आहे. कीवर्ड्स हे कॉम्पायलरसाठी ओळखपूर्वक नावे आहेत, म्हणजे त्यांना एक सापडलेला आणि ओळखलेला शब्द आहे. कीवर्ड्ससाठी निर्दिष्ट व्याख्याने आहेत आणि त्यांचा प्रतिसाद हे निर्दिष्ट आहे.

कीवर्ड्स वापरताना वेरीएबल नावे वापरणे नाकारयाचे आहे.

C प्रोग्रामिंग भाषेत 32 कीवर्ड्स आहेत, ज्यांनी खालीलप्रमाणे आहेत:

auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while

१.३.४ अडेंटिफायर (Identifiers)

अडेंटिफायर ह्या सामान्य शब्दाचा उपयोग वेरीएबल्स, फंक्शन्स, आणि अॅरे नावांसाठी करण्यासाठी केला जाते. आपल्याला स्वतेत्रपणे ठरविलेला नाव, ज्यामुळे त्याचे वापर तयार केले जाते, असं इन्फॉर्मली अडेंटिफायर म्हणले जाते. ह्या अडेंटिफायर नावांमध्ये शब्दांचे आणि अंकांचे संयोजन असते.

अडेंटिफायर नावांचे पहिला अक्षर किंवा अंकांना असलेला एक अक्षर किंवा अंडरस्कोर (_) असलेला एक अक्षराने सुरु होते. त्यानंतर, कोणत्याही अंक, अक्षर किंवा अंडरस्कोराने संयोजन करू शकता. अडेंटिफायर वर्तमान क्रमांकीय केवळ आणि केससह विभिन्न असणारे चिन्ह न घेतलेले असलेले हवे, आणि त्यांचे वापर केवळ असेही आहे की क्रियांच्या वर्णनांतर आपटे घेतलेला नाव सापडते. आपल्याला कीवर्ड्ससोबत विरोधीत केलेला आणि त्यांचे विरोधीत उपयोग केलेला अडेंटिफायर नाव सुरक्षित ठरविणे आवश्यक आहे, कारण हे त्याचे वापर केवळ स्थानिक आवश्यकतेसाठी हे विशेषपणे ठरविणारं हवे.

१.३.५ कॉन्स्टंट- नंबर कॉन्स्टंट्स (Constants - number constants)

कॉन्स्टंट्स हे अपरिवर्तनीय (Immutable) वेरियेबल्स आहेत, ज्याची किंमत प्रोग्राममध्ये एकदमच घोषित केलीली असते. त्यामुळे त्या किंमती एकदमच बदलू शकत नाहीत. const कीवर्ड हे एक qualifier आहे ज्याचा वापर कॉन्स्टंट वेरियेबल्स घोषित करण्यासाठी केला जातो. कॉन्स्टंट वेरियेबल एकदमच घोषित केल्यानंतर त्याची किंमत बदल करण्यात क्षमता नसते.

मांडणी

const data_type var_name = value;

नंबर कॉन्स्टंट्स

const int INTEGER_CONSTANT = 42;

१.३.६ कॅरेक्टर कॉन्स्टंट (character constants)

कॅरेक्टर कॉन्स्टंट म्हणजेच सिंगल अक्षर ,त्या किंमती एकदमच बदलू शकत नाहीत, तसेच त्या सिंगल कोट्समध्ये लिहिलेल्या असतात.

मांडणी

```
const char CHARACTER_CONSTANT = 'A';
```

प्रोग्राम

```
#include <stdio.h>
  int main() {
char character = 'A';
printf("The character constant is: %c\n", character);
return 0;
}
```

१.३.७ स्ट्रिंग कॉन्स्टंट (string constants)

एका पेक्षा अधिक अक्षरांची एक सिरिझ ज्याची पुरावी डबल कोट्समध्ये आवर्तित असते, तिचा प्रतिष्ठान "स्ट्रिंग कॉन्स्टंट" म्हणले जाते . हे एक अक्षराचे आचारधर्म आहे ज्यानंतर एक शून्य चरित्र '\0' म्हणजे नल कॅरेक्टरने समाप्त होते.

प्रोग्राम

```
#include <stdio.h>
 int main() {
char string[] = "Hello, World!";
printf("The string constant is: %s\n", string);
return 0;
}
```

आउटपुट

```
The string constant is: Hello, World!
```

१.३.८ व्हेरिएबल (Variables)

C प्रोग्रामिंग मधील व्हेरिएबल हे नाव असलेले मेमरी स्थान आहे ,जे काही स्वरूपाचा डेटा संग्रहित करण्यात मदत करते आणि आवश्यकतेनुसार ते पुनर्प्राप्त करते. आपण व्हेरिएबलमध्ये विविध प्रकारचे डेटा संचयित करू शकतो आणि तेच व्हेरिएबल इतर डेटा कितीही वेळा संचयित करण्यासाठी पुन्हा वापरू शकतो.

मांडणी

```
data_type variable_name = value;
```

किंवा

data_type variable_name1, variable_name2;

- **C मध्ये व्हेरिएबलचे नाव देण्याचे करण्याचे नियम:**

तुम्ही व्हेरिएबलला कोणतेही नाव नियुक्त करू शकता ,जोपर्यंत ते खालील नियमांचे पालन करते.

- व्हेरिएबल नावामध्ये फक्त अक्षरे, अंक आणि अंडरस्कोर असणे आवश्यक आहे.
- व्हेरिएबलचे नाव केवळ अल्फाबेट किंवा अंडरस्कोरने सुरू होणे आवश्यक आहे. त्याची सुरुवात अंकाने होऊ शकत नाही.
- व्हेरिएबलच्या नावामध्ये कोणत्याही स्पेसला परवानगी नाही.
- व्हेरिएबल नाव कोणताही आरक्षित शब्द किंवा कीवर्ड नसावा.

सी व्हेरिएबल प्रकार

सी व्हेरिएबलचे खालील प्रकारांमध्ये वर्गीकरण केले जाते.

१.स्थानिक व्हेरिएबल

२.ग्लोबल व्हेरिएबल

१. स्थानिक व्हेरिएबल (Local Variable): C मधील स्थानिक व्हेरिएबल हे एक व्हेरिएबल आहे, जे फंक्शन किंवा कोडच्या ब्लॉकमध्ये घोषित केले जाते. त्याची व्याप्ती ज्या ब्लॉक किंवा फंक्शनमध्ये घोषित केली आहे त्यापुरती मर्यादित आहे.

२. ग्लोबल व्हेरिएबल (Global Veriable): C मधील ग्लोबल व्हेरिएबल हे एक व्हेरिएबल आहे, जे फंक्शन किंवा कोडच्या ब्लॉकच्या बाहेर घोषित केले जाते. त्याची व्याप्ती संपूर्ण प्रोग्रॅममध्ये असते. आपण C प्रोग्राममध्ये कुठेही ग्लोबल व्हेरिएबल घोषित केल्यावर ऍक्सेस करू शकतो.

१.३.९ डेटा टाइप इन सी (Data types in 'C') :

C प्रोग्रामिंग मध्ये विविध प्रकारच्या डेटा साठवण्यात आणि प्रोग्राममध्ये विविध प्रकारच्या डेटा साधून घेण्यात सुधारित करण्यात आलेले व्हेरिएबल हे विविध डेटा टाइप वापरून घेतलेले जाते.

- पूर्णांक (int): हे डेटा टाइप पूर्णांकांसाठी आहे

  int myInteger = 42;

- फ्लोटिंग पॉईंट (float): हे डेटा टाइप दशांशांसह संख्यांच्या व्हॅल्यू घेतली जातात.

  float myFloat = 3.14;

- डबल प्रेसिजन फ्लोटिंग पॉईंट (double): हे डेटा टाइप फक्त फ्लोटिंग पॉईंटसाठी आहे परंतु त्यातील व्हॅल्यू फक्त वाचनासाठी सुधारित केली जातात.

double myDouble = 2.71828;

- क्यार (char): हे डेटा टाइप एकच अक्षर साठवण्यात .

  char myChar = 'A';

- स्ट्रिंग (String): हे डेटा टाइप एक अजुन्य अक्षरांची सिरीज घेतलेली आणि वाचनासाठी वापरून घेतलेली जाते.

  char myString[] = "MY STRING";

  या विविध प्रकारची डेटा टाइप्स वापरले जातात.

१.३.१० डिक्लेरिंग व्हेरिएबल (Declaring variables)

पुढील प्रमाणे व्हेरिएबल डिक्लेर केले जाते.

मांडणी

data_type variable_name = value;

किंवा

data_type variable_name1, variable_name2;

१.३.११ डेटा टाईप कन्वर्जन (data type conversion)

तुम्हाला एका डेटा प्रकाराचे व्हॅल्यू दुसऱ्या प्रकारात रूपांतरित करावे लागेल. हे प्रकार डेटा टाईप कन्वर्जन म्हणून ओळखले जाते.

C मध्ये रूपांतरणाचे दोन प्रकार आहेत:

१.इम्प्लीसीट टाईप कन्वर्जन

२.एक्यप्लीसीट टाईप कन्वर्जन

इम्प्लीसीट टाईप कन्वर्जन(Implicit Conversion): जेव्हा तुम्ही एका प्रकाराचे व्हॅल्यू दुसऱ्या प्रकाराला नियुक्त करतात ,यालाच automatic conversion असे पण म्हणतात.

उदाहरण :

```
#include <stdio.h>
int main()
{
   int x = 10; // integer x
   char y = 'a'; // character c
```

```
        // y implicitly converted to int. ASCII
           // value of 'a' is 97
        x = x + y;
        // x is implicitly converted to float
        float z = x + 1.0;
        printf("x = %d, z = %f", x, z);
        return 0;
}
```

आउटपुट

```
x = 107, z = 108.000000
```

एक्यप्लीसीट टाईप कन्वर्जन (Explicit Type Conversion): या प्रक्रियेला प्रकार कास्टिंग देखील म्हणतात ,आणि ती यूझरने परिभाषित केली आहे. येथे यूझर टाईपकास्ट करू शकतो.

```
#include<stdio.h>
 int main()
{
   double x = 1.2;
     // Explicit conversion from double to int
   int sum = (int)x + 1;
     printf("sum = %d", sum);
     return 0;
}
```

आउटपुट

```
sum = 2
```

१.४ बेसिक इनपुट आणि आउटपुट फंक्शन (Basic Input and Output functions) :

सी लँग्वेजमध्ये स्टँडर्ड लायब्ररी आहेत, जी प्रोग्राममध्ये इनपुट आणि आउटपुटला परवानगी देतात. C मधील stdio.h किंवा मानक इनपुट आउटपुट लायब्ररी ज्यामध्ये इनपुट आणि आउटपुटसाठी पद्धती आहेत.

स्कॅनएफ()scanf()पद्धत: यामध्ये व्हॅल्यू युझर कडून कॉम्पुटर ला दिल्या जातात .

मांडणी:

scanf("%X", &variableOfXType);

प्रिंटएफ() printf():यामध्ये कॉम्पुटर स्क्रीन वर प्रोग्राम चा आउटपुट दाखवतो.

मांडणी:

printf("%X", variableOfXType);

काही कॉमन कन्वर्जन स्पेसिफायर खालील प्रमाणे:

- `%d`: रीड इंतीजर व्हॅल्यू
- `%f`: रीड फ्लोटिंग पॉईंट व्हॅल्यू
- `%c`: रीड सिंगल करेक्टर
- `%s`: स्ट्रिंग कॅरेक्टर
- `%lf`: रीड डबल प्रेसिजन फ्लोटिंग पॉईंट व्हॅल्यूज

१.५ सी प्रोग्रामिंग मधील ऑपरेटर(Operator in C programming):

ऑपरेटर हे एक चिन्ह आहे जे मूल्य किंवा व्हेरिएबलवर फंक्शन करते. उदाहरणार्थ: + एडिशन करण्यासाठी वापरले जाणारे ऑपरेटर आहे

सी प्रोग्रामिंग ऑपरेटरना खालील गटांमध्ये विभाजित करते:

- अर्थमॅटिक ऑपरेटर(Arithmetic operators)
- असाइनमेंट ऑपरेटर(Assignment operators)
- कम्पॅरिझन ऑपरेटर(Comparison operators)
- लॉजिकल ऑपरेटर(Logical operators)
- बिटवाइज ऑपरेटर(Bitwise operators)
- साईज ऑफ ऑपरेटर(sizeof operator)

१.५.१ अर्थमॅटिक ऑपरेटर(Arithmetic operators):

अर्थमॅटिक ऑपरेटर चा उपयोग खालील प्रमाणे केला जातो.

Operator	Name	Description	Example
+	Addition	दोन नंबरची एडिशन करण्यासाठी	x + y
-	Subtraction	एका व्हेरिएबल मधून दुसऱ्या व्हेरिएबलची व्हॅल्यू मायनस करण्यासाठी	x - y
*	Multiplication	दोन नंबरची मल्टिप्लिकेशन करण्यासाठी	x * y

/	Division	एका नंबरने दुसऱ्या नंबरला डीवाईड करण्यासाठी	x / y
%	Modulus	डिव्हिजन रिमाइंडर रिटर्न करण्यासाठी	x % y
++	Increment	वेरिएबलची व्हॅल्यू एक ने वाढवण्यासाठी	++x
--	Decrement	व्हेरिएबलची व्हॅल्यू एक न कमी करण्यासाठी	--x

१.५.२ असाइनमेंट ऑपरेटर(Assignment operators):

असाइनमेंट ऑपरेटर व्हेरिएबल्सना व्हॅल्यू नियुक्त करण्यासाठी वापरले जातात.

खालील उदाहरणात, x नावाच्या व्हेरिएबलला १० व्हॅल्यू नियुक्त करण्यासाठी आम्ही असाइनमेंट ऑपरेटर (=) वापरतो:

```
int x = 10;
```

अतिरिक्त असाइनमेंट ऑपरेटर (+=) व्हेरिएबलमध्ये मूल्य जोडतो:

```
int x = 10;
x += 5;
```

आउटपुट

```
15
```

Operator	Description	Work
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3

>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

१.५.३ कम्पॅरिझन ऑपरेटर(Comparison operators):

कम्पॅरिझन ऑपरेटर दोन व्हॅल्यू ची कम्पॅरिझन करण्यासाठी वापरली जाते. प्रोग्रामिंग मध्ये कम्पॅरिझन ऑपरेटर महत्त्वाचे आहे , ते आपणास कंपॅरिझन करण्यासाठी उपयोगी येते. तसेच या ऑपरेटर च्या मदतीने आपणास 1 किंवा 0 या पद्धतीने आऊटपुट रिटर्न होतो.

खालील उदाहरणात, 5 हे 3 पेक्षा मोठे आहे की नाही हे शोधण्यासाठी आम्ही ऑपरेटर (>) पेक्षा मोठे वापरतो:

```
int x = 5;
int y = 3;
printf("%d", x > y); // returns 1 (true) because 5 is greater than 3
```

आउटपुट

```
1
```

ऑपरेटर	ऑपरेटर चे नाव	उदाहरण	माहिती
==	Equal to	x == y	दोन व्हेरिएबल ची व्हॅल्यू जर इक्वल असेल तर रिटर्न वन होतो.
!=	Not equal	x != y	जर दोन वेरिएबलची व्हॅल्यू वेगळी असेल तर रिटर्न ऑन होतो.
>	Greater than	x > y	जर पहिल्या वेरीएबलची व्हॅल्यू दुसऱ्यापेक्षा मोठी असेल तर रिटर्न एक होतो.
<	Less than	x < y	जर दुसऱ्या वेरिएबलची व्हॅल्यू पहिल्यापेक्षा मोठी असेल तर रिटर्न वन होतो.
>=	Greater than or equal to	x >= y	जर पहिल्या व्हेरिएबलची व्हॅल्यू दुसऱ्यापेक्षा मोठी किंवा इक्वल असेल तर एक रिटर्न होतो.
<=	Less than or equal to	x <= y	जर दुसऱ्या वेरीएबलची व्हॅल्यू पहिला व्हेरिएबल पेक्षा मोठी किंवा इक्वल असेल तर रिटर्न वन होतो.

१.५.४.लॉजिकल ऑपरेटर(Logical operators):

आपण लॉजिकल ऑपरेटरसह सत्य किंवा असत्य व्हॅल्यू ची चाचणी देखील करू शकता.

ऑपरेटर	ऑपरेटर चे नाव	उदाहरण	माहिती
&&	Logical and	x < 5 && x < 10	दोन्हीही स्टेटमेंट बरोबर असेल तर रिटर्न 1 होतो.
||	Logical or	x < 5 || x < 4	एक स्टेटमेंट जर बरोबर असेल तर 1 रिटर्न होतो.
!	Logical not	!(x < 5 && x < 10)	जर रिझल्ट 1 असेल तर रिटर्न 0 होतो.

१.५.५.बिटवाइज ऑपरेटर(Bitwise operators):

बेरीज, वजाबाकी, गुणाकार आणि भागाकार बिट-स्तरावर केले जातात. सी प्रोग्रामिंगमध्ये बिट-लेव्हल ऑपरेशन्स करण्यासाठी, बिटवाइज ऑपरेटर वापरले जातात.

ऑपरेटर	माहिती
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise complement
<<	Shift left
>>	Shift right

- C मधील & (bitwise AND) दोन संख्या ऑपरेंड म्हणून घेतो आणि दोन संख्यांच्या प्रत्येक बिटवर AND करतो. दोन्ही बिट 1 असल्यासच AND चा परिणाम 1 आहे.
- C मध्ये (bitwise OR) दोन संख्या ऑपरेंड म्हणून घेते आणि दोन संख्यांच्या प्रत्येक बिटवर OR करते. OR चा परिणाम 1 असेल जर दोन बिटपैकी कोणतेही 1 असेल.
- C मधील ^ (bitwise XOR) दोन संख्या ऑपरेंड म्हणून घेते आणि दोन संख्यांच्या प्रत्येक बिटवर XOR करते. दोन बिट भिन्न असल्यास XOR चा परिणाम 1 आहे.
- C मधील << (डावी शिफ्ट) दोन संख्या घेते, डावीकडे पहिल्या ऑपरेंडचे बिट शिफ्ट करते आणि दुसरे ऑपरेंड शिफ्ट करण्याच्या ठिकाणांची संख्या ठरवते.

- C मधील >> (उजवी शिफ्ट) दोन संख्या घेते, उजवीकडे पहिल्या ऑपरेंडचे बिट शिफ्ट करते आणि दुसरे ऑपरेंड शिफ्ट करण्याच्या ठिकाणांची संख्या ठरवते.
- C मधील ~ (bitwise NOT) एक संख्या घेते आणि त्यातील सर्व बिट्स उलटते.

X	Y	X & Y	X | Y	X ^ Y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

प्रोग्राम:

```
#include <stdio.h>
int main()
{
   // a = 5(00000101), b = 9(00001001)
   unsigned char a = 5, b = 9;
    // The result is 00000001
   printf("a = %d, b = %d\n", a, b);
   printf("a&b = %d\n", a & b);
    // The result is 00001101
   printf("a|b = %d\n", a | b);
    // The result is 00001100
   printf("a^b = %d\n", a ^ b);
    // The result is 11111010
   printf("~a = %d\n", a = ~a);
    // The result is 00010010
   printf("b<<1 = %d\n", b << 1);
    // The result is 00000100
   printf("b>>1 = %d\n", b >> 1);
    return 0;
}
```

आउटपुट

```
a = 5, b = 9
a&b = 1
```

a|b = 13

a^b = 12

~a = 250

b<<1 = 18

b>>1 = 4

१.५.६ साईज ऑफ ऑपरेटर(sizeof operator):

डेटा टाईप किंवा व्हेरिएबल मेमरी साईज (बाईट मध्ये) साईज ऑफ ऑपरेटर्स आढळून येते

उदाहरण

```
int myInt;
float myFloat;
double myDouble;
char myChar;
printf("%lu\n", sizeof(myInt));
printf("%lu\n", sizeof(myFloat));
printf("%lu\n", sizeof(myDouble));
printf("%lu\n", sizeof(myChar));
```

आउटपुट

4

4

8

1

संदर्भ:

१. https://www.geeksforgeeks.org/printf-in-c/?ref=lbp

२. https://www.geeksforgeeks.org/scanf-in-c/?ref=lbp

३. https://www.geeksforgeeks.org/logical-operators-in-c/?ref=lbp

४. https://www.geeksforgeeks.org/tokens-in-c/

५. https://www.geeksforgeeks.org/basic-input-and-output-in-c/?ref=lbp

६. https://topperworld.in/c-programming-notes/

७. https://www.geeksforgeeks.org/printf-in-c/?ref=lbp

८. https://www.geeksforgeeks.org/c-language-introduction/

९. https://www.vssut.ac.in/lecture_notes/lecture1422486950.pdf

१०. https://byjus.com/gate/introduction-to-c-programming/

११. https://vardhaman.org/wp-content/uploads/2021/03/CP.pdf

घटक २

कंट्रोल स्ट्रक्चर्स

(Control structures)

गुण (१६)

विषय निष्पत्ती (Course Outcome) :ब्रँचिंग आणि लूपिंग स्टेटमेंटचा समावेश असलेला C प्रोग्राम विकसित करा.

घटक निष्पत्ती (Unit Outcome):

१ डिसीजेन मेकिंग स्टेटमेंट वापरून 'C' प्रोग्राम लिहा.

२ पुनरावृत्ती समस्या सोडवण्यासाठी C प्रोग्राममधील लूप स्टेटमेंट वापरा.

३ लूपमधील प्रोग्राम फ्लो बदलण्यासाठी योग्य स्टेटमेंट वापरा.

---

२१. कंडीशनल स्टेटमेंट (Conditional Statement)

C प्रोग्रॅमिंगमधील कंडिशनल स्टेटमेंटचा वापर तुम्हाला विशिष्ट कंडिशन सत्य की असत्य आहे यांचे मूल्यमापन करून यावर आधारित ठराविक कोड कार्यान्वित (execute) करण्यासाठी केला जातो.

वेगवेगळ्या कंडिशनवर आधारित तुमच्या प्रोग्रामचा प्रवाह (program flow) नियंत्रित करण्यासाठी कंडिशनल स्टेटमेंट वापरली जातात. कंडिशनल स्टेटमेंटचा वापर करण्यासाठी प्रथमतः आपल्याला संबंधात्मक ऑपरेटर्स (relational operators) व तार्किक ऑपरेटर्स(Logical Operator) चा अभ्यास करणे गरजेचे आहे.

➢ रिलेशनल ऑपरेटर (reletional operators):

C प्रोग्रामिंगमध्ये संबंधात्मक ऑपरेटर्स (relational operators) हे सामान्यत: मूल्ये तुलना (value compare) करण्यासाठी वापरले जातात.

संबंधात्मक ऑपरेटर्स (relational operators) खालील प्रमाणे आहेत.

1. == समान (equal to)
2. != समान नसल्यास (not equal to)
3. < च्या पेक्षा लहान (less than))
4. > च्या पेक्षा मोठे (greater than)
5. <= च्या पेक्षा लहान किंवा समान (less than or equal to)
6. >= च्या पेक्षामोठे किंवा समान (greater than or equal to)

उदाहरणार्थ:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a = 5, b = 10;
if (a == b)
 {
    // Code executes if 'a' is equal to 'b'
    printf("'a' is equal to 'b'");
}
if (a < b)
 {
    // Code executes if 'a' is less than 'b'
    printf("'a' is less than 'b'");
 }
getch();
}
```

Output:

'a' is less than 'b'

हे ऑपरेटर्स तुलनात्मक परिणाम (true किंवा false) देतात ज्यांच्यामध्ये तुलना केली जाते.

- **तार्किक ऑपरेटर्स (Logical Operator):**

C प्रोग्रामिंगमध्ये, तार्किक ऑपरेटर्स(Logical Operator) हे सामान्यत: बूलियन मूल्यांवर तार्किक कामे करण्यासाठी वापरले जातात.

तार्किक ऑपरेटर्स(Logical Operator) खालील प्रमाणे आहेत.

1. && (logical AND)
2. || (logical OR)
3. ! (logical NOT)

उदाहरणार्थ:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int x = 5, y = 10;
if (x > 0 && y > 0)
 {
   // Code executes if both x and y are greater than 0
            printf("Both x and y are greater than 0\n");
}
if (x > 0 || y > 0)
{
   // Code executes if either x or y is greater than 0
             printf("Either x or y is greater than 0\n");
}
if (!(x > 0))
 {
   // Code executes if x is not greater than 0
printf("x is not greater than 0\n");
}
getch();
}
```

Output:

```
Both x and y are greater than 0
Either x or y is greater than 0
```

इथे, && तार्किक आणि (Logical AND) कार्य करते, || तार्किक किंवा (Logical OR) कार्य करते, आणि ! तार्किक नाही (Logical NOT) कार्य करते.

C प्रोग्रामिंगमध्ये खालील प्रमाणे कंडीशनल स्टेटमेंट (Conditional Statements) आहेत.

- **if स्टेटमेन्ट (if Statement):**

if **स्टेटमेन्ट कोडच्या ब्लॉकच्या सशर्त अंमलबजावणी** (conditional execution) **साठी वापरला जातो**.

Syntax:

```
if (condition)
{
    // Code to execute if the condition is true
}
```

उदाहरणार्थ:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int num = 10;
if (num > 0)
{
printf("The number is positive.\n");
}
getch();
}
```

Output:

The number is positive.

- **if-else स्टेटमेन्ट (if-else Statement):**

if-else **स्टेटमेन्ट कंडिशन असत्य असताना अंमलात आणण्यासाठी कोडचा पर्यायी ब्लॉक प्रदान करते**.

Syntax:

```
if (condition)
```

```
{
    // Code to execute if the condition is true
 } else {
    // Code to execute if the condition is false
 }
```

उदाहरणार्थ:

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int num = -5;
 if (num > 0)
{
    printf("The number is positive.\n");
 }
 else
 {
    printf("The number is non-positive.\n");
 }
 getch();
}
```

Output:

The number is non-positive

- **Nested if-else स्टेटमेन्ट (Nested if-else Statement):**

नेस्टेड if-else **स्टेटमेंट हे दुसऱ्या** if **स्टेटमेंटमधील** if **स्टेटमेंट असते**.

Syntax:

```
if (condition1)
```

```
{
  /* code to be executed if condition1 is true */
  if (condition2)
{
    /* code to be executed if condition2 is true */
  }
else
 {
    /* code to be executed if condition2 is false */
  }
}
else
 {
  /* code to be executed if condition1 is false */
}
```

उदाहरणार्थ:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int num;
printf("Enter a number: ");
scanf("%d", &num);
if (num > 0)
 {
printf("%d is positive.\n", num);
 }
else
 {
```

```
    if (num < 0)
{
printf("%d is negative.\n", num);
}
else
 {
printf("%d is zero.\n", num);
  }
 }

getch();
}
```

Output:

Enter a number: 10

10 is positive.

- **if-else ladder स्टेटमेन्ट (if-else ladder Statement):**

C प्रोग्रामिंग **तुम्हाला एका** प्रोग्राममध्ये **अनेक कंडिशन तपासण्याची परवानगी देते.जर पहिली** if **कंडिशन असत्य असेल**, **तर ती पुढील** else if **कंडिशन तपासली जाते आणि कोणतीही** else if **कंडिशन सत्य नसल्यास**, else **ब्लॉक** (**उपस्थित असल्यास**) **कार्यान्वित**(execute) **केला जातो**.

Syntax:

```
if (condition1)
 {
   // code to be executed if condition1 is true
}
 else if (condition2)
 {
   // code to be executed if condition2 is true
}
 else if (condition3)
```

```
{
   // code to be executed if condition3 is true
}
 else
{
   // code to be executed if none of the conditions is true
}
```

उदाहरणार्थ:

```
#include <stdio.h>
#include<conio.h>
void main()
 {
    int number;
    printf("Enter a number: .\n");
    scanf("%d", &number);
if (number > 0)
{
 printf("The number is positive.\n");
 }
else if (number == 0)
{
  printf("The number is Zero.\n");
 }
 else
 {
 printf("The number is negative.\n");
 }
getch();
}
```

Output:

Enter a number:

5

The number is positive.

- **स्विच स्टेटमेन्ट (switch statement):**

C प्रोग्रामिंग मधील स्विच स्टेटमेंट हा एकापेक्षा जास्त कंडिशन हाताळण्याचा दुसरा मार्ग आहे. जेव्हा तुमच्याकडे एकाच व्हेरिएबलची तपासणी करण्यासाठी अनेक कंडिशन असतात तेव्हा ते स्विच स्टेटमेंट एक पर्याय प्रदान करते.

Syntax:

```
Switch  (expression)
{
   case constant1:
      // code to be executed if expression equals constant1
      break;
   case constant2:
      // code to be executed if expression equals constant2
      break;
   // more cases as needed
   default:
      // code to be executed if none of the cases match
}
```

उदाहरणार्थ:

```
#include <stdio.h>
#include<conio.h>
void main()
 {
```

```
    char grade;
    printf("Enter your grade (A, B, C, D, or F): ");
    scanf(" %c", &grade);  // Note the space before %c to consume any leading whitespace.

    switch (grade)
{
        case 'A':
            printf("Excellent!\n");
            break;
        case 'B':
            printf("Good!\n");
            break;
        case 'C':
            printf("Satisfactory.\n");
            break;
        case 'D':
            printf("Needs improvement.\n");
            break;
        case 'F':
            printf("Fail.\n");
            break;
        default:
            printf("Invalid grade.\n");
    }
getch();
}
```

Output:

Enter your grade

'A'

Excellent!

या उदाहरणात, स्विच स्टेटमेंट व्हेरिएबल ग्रेडचे **value** तपासते. **Value** वर अवलंबून, ते संबंधित केस **executes** करते . कोणतीही केस जुळत नसल्यास, डीफॉल्ट केस **executes** केली जाते.

२. २ लूपिंग स्टेटमेंट (Looping statements):

लूपिंग म्हणजे समान पुनरावृत्ती होय, विशिष्ट स्थिती प्राप्त होईपर्यंत अनेक वेळा प्रक्रिया करणे म्हणजे लूपिंग होय. हे iteration म्हणून देखील ओळखले जाते.

C प्रोग्रामिंगमध्ये खालील प्रमाणे लूपिंग स्टेटमेंट (Looping statements) आहेत.

- **व्हाईल लूप (while loop):**

व्हाईल लूप (while loop) हे जोपर्यंत निर्दिष्ट कंडिशन सत्य आहे तोपर्यंत कोडच्या ब्लॉकच्या वारंवार अंमलबजावणीसाठी **(repeated execution of a block of code)** वापरला जातो.

Syntax:

```
 while (condition)
{
    // Code to execute while the condition is true
 }
```

उदाहरणार्थ:

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int count = 1;
 while (count <=5)
{
    printf("Count: %d\n", count);
    count++;
 }
  getch();
```

```
}
```

Output:

```
Count:  1
Count:  2
Count:  3
Count:  4
Count:  5
```

- **डू व्हाईल लूप (do-while Loop):**

  डू व्हाईल लूप **(do-while Loop)** हे **while** लूप प्रमाणेच आहे**,** परंतु कोडच्या ब्लॉकच्या अंमलबजावणीनंतर **(after the execution of the block of code)** कंडिशन तपासली जाते**.**

  - कोडचा ब्लॉक किमान एकदा अंमलात आणला गेला आहे **(executed at least once)** याची खात्री करते**.**

Syntax:

```
 do
{
    // Code to execute
 } while (condition);
```

उदाहरणार्थ:

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int num = 1;
 do {
    printf("Number: %d\n", num);
    num++;
 } while (num <= 5);
   getch();
}
```

Output:

Number: 1

Number: 2

Number: 3

Number: 4

Number: 5

- **फॉर लूप (for Loop):**

C मध्ये, फॉर लूप चा वापर ठराविक वेळा कोडचा ब्लॉक पुन्हा पुन्हा कार्यान्वित करण्यासाठी केला जातो.

Syntax:

```
for (initialization; condition; update)
{
   // code to be executed in each iteration
}
```

उदाहरणार्थ:

```
#include <stdio.h>
#include<conio.h>
void main()
 {
   for (int i = 1; i <= 5; i++)
{
       printf("%d ", i);
  }
getch();
}
```

Output:

1 2 3 4 5

स्पष्टीकरण:

- int i = 1: Initialization, व्हेरिएबल i घोषित करते आणि त्याचे प्रारंभिक value 1 वर सेट करते.

- i <= 5: Condition, जोपर्यंत ही कंडिशन सत्य आहे तोपर्यंत लूप चालू राहते.

- i++: Update, प्रत्येक पुनरावृत्तीमध्ये**(iteration) i** चे **value 1** ने वाढवते.

२.३ ब्रांचिंग स्टेटमेंट **(Branching statements):**

C प्रोग्रामिंगमध्ये ब्रांचिंग स्टेटमेंट ही **C** मधील असे स्टेटमेंट आहेत जी प्रोग्रामरला आवश्यकतेनुसार प्रोग्रामच्या अंमलबजावणीचा प्रवाह **(the flow of execution of the program)** नियंत्रित करण्यास मदत करतात.

C प्रोग्रामिंगमध्ये खालील प्रमाणे ब्रांचिंग स्टेटमेंट **(Branching statements)** आहेत.

- गो टू स्टेटमेंट**(goto Statement):**

गो टू स्टेटमेंट**(goto Statement)** समान फंक्शनमध्ये लेबल केलेल्या स्टेटमेंटवर जाण्याची परवानगी देते.सामान्यतः गो टू स्टेटमेंट**(goto Statement)** चा वापर करणे हे **bad practice** मानले जाते कारण ते कोड कमी वाचनीय आणि देखरेख करणे कठीण बनवू शकते.

Syntax:

```
goto label;
// ...
label:
// Code to execute
```

उदाहरणार्थ:

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int i = 0;
 start:
    printf("%d ", i);
    i++;
    if (i < 5)
       goto start;
 getch();
}
```

Output:

0 2 3 4

- **ब्रेक स्टेटमेंट(break Statement):**

ब्रेक स्टेटमेंट**(break Statement)** सर्वात आतील लूप किंवा स्विच स्टेटमेंट समाप्त करण्यासाठी वापरले जाते.

```
#include<stdio.h>
#include<conio.h>
void main()
{
  for (int i = 0; i < 10; i++)
 {
     if (i == 5)
     {
        break; // Terminate the loop when i is 5
     }
     printf("%d ", i);
  }
  getch();
}
```

Output:

0 2 3 4

- **कंटिन्यूव स्टेटमेंट(continue Statement) :**

कंटिन्यूव स्टेटमेंट**(continue Statement)** उर्वरित लूप बॉडी वगळते आणि पुढील पुनरावृत्तीकडे जाते.

- Skips the rest of the loop body and proceeds to the next iteration.

- Syntax:

```
Continue;
```

```
#include<stdio.h>
#include<conio.h>
void main()
{
 for (int i = 0; i < 10; i++)
{
    if (i % 2 == 0)
{
        continue; // Skip even numbers
    }
    printf("%d ", i);
 }
```

Output:

```
 1  3 5 7 9
```

संदर्भ:

१ https://www.geeksforgeeks.org/c-programming-language/

२ https://www.w3schools.com/c/c_intro.php

३ Programming in Ansi C By E. Balagurusamy (Author)

४ Let Us C by Yashavant Kanetkar (Author)

घटक –३

अॅरे आणि संरचना

(Arrays and structure)

गुण:१६

विषय निष्पत्ती (Course Outcome): CO3: C-प्रोग्राम वापरून अॅरे आणि संरचना लागू करा.

घटक निष्पत्ती (Unit Outcome):

१. एक-आयामी(Onedimensional)अॅरेवर ऑपरेशन्स करण्यासाठी C प्रोग्राम लिहा.(Write a C Program to perform operations on onedimensional array.)
२. द्विमितीय अॅरेचे घटक घोषित करा, आरंभ करा आणि प्रवेश करा.(Declare, initialize,and access elements of twodimensional array.)
३. संरचना वापरून डेटा घोषित करा, आरंभ करा आणि प्रवेश करा. (Declare ,initializeand access data usingStructure.)
४. टाइपडेफ आणि एनम स्पष्ट करा. (Explain typedef and enum)

३.१ अॅरेची वैशिष्ट्ये, एक आयाम आणि द्विमितीय अॅरे, बहु-आयामी अॅरेची संकल्पना

(Characteristics of an array, One dimension and twodimensional arrays, concept of multi-dimensionalarrays)

३.१.१. अॅरेची वैशिष्ट्ये (Characteristics of an array)

अॅरे ही प्रोग्रामिंगमधील मूलभूत डेटा स्ट्रक्चर्स आहेत जी समान डेटा प्रकाराचे घटक संलग्न मेमरी स्थानांमध्ये संग्रहित करतात. येथे अॅरेची प्रमुख वैशिष्ट्ये आहेत:

१.एकसंध घटक:

ॲरेमधील सर्व घटक समान डेटा प्रकाराचे असणे आवश्यक आहे. उदाहरणार्थ, ॲरे पूर्णांक, फ्लोटिंग-पॉइंट संख्या, वर्ण इत्यादी संचयित करू शकते, परंतु सर्व घटक एकाच प्रकारचे असले पाहिजेत.

२.सलग मेमरी वाटप:

ॲरे घटक जवळच्या मेमरी स्थानांमध्ये संग्रहित केले जातात. याचा अर्थ ॲरेचे घटक एकामागून एक मेमरीमध्ये साठवले जातात.

३.अनुक्रमित प्रवेश:

अनुक्रमणिका किंवा सबस्क्रिप्ट वापरून ॲरेमधील घटकांमध्ये प्रवेश केला जातो. पहिल्या घटकासाठी अनुक्रमणिका 0 पासून सुरू होते आणि शेवटच्या घटकासाठी (ॲरे आकार - 1) पर्यंत जाते.

४.निश्चित आकार:

ॲरेचा निश्चित आकार असतो, जो घोषणेच्या वेळी निर्धारित केला जातो. एकदा आकार परिभाषित केल्यानंतर, तो रनटाइम दरम्यान बदलला जाऊ शकत नाही.

५.यादृच्छिक प्रवेश:

ॲरेमधील घटकांना त्यांच्या अनुक्रमणिकेचा वापर करून थेट प्रवेश केला जाऊ शकतो, कोणत्याही घटकास यादृच्छिक प्रवेशास अनुमती देऊन. यामुळे घटकांमध्ये प्रवेश करण्यासाठी सतत-वेळची गुंतागुंत निर्माण होते.

६.रेखीय डेटा संरचना:

ॲरे एक रेखीय डेटा स्ट्रक्चर दर्शवतात, याचा अर्थ घटक एका क्रमाने संग्रहित केले जातात. ही रेखीय मांडणी घटकांचे ट्रॅव्हर्सल आणि हाताळणी सुलभ करते.

७.अनुक्रमिक प्रवेशासाठी कार्यक्षम:

घटकांच्या अनुक्रमिक प्रवेशासाठी ॲरे कार्यक्षम आहेत. ॲरेमधील घटकांद्वारे पुनरावृत्ती करणे हे लिंक केलेल्या सूचीसारख्या इतर डेटा स्ट्रक्चरच्या तुलनेत जलद आहे.

८.कार्यक्षम मेमरी वापर:

ॲरे मेमरी-कार्यक्षम असतात, कारण ते संलग्न मेमरी ब्लॉक्सचे वाटप करतात. हे बेस ॲड्रेस आणि इंडेक्स वापरून कोणत्याही घटकाच्या मेमरी ॲड्रेसची गणना करणे सोपे करते.

९.स्थिर मेमरी वाटप:

बऱ्याच प्रोग्रामिंग भाषांमध्ये, ॲरे हे स्थिरपणे वाटप केलेल्या मेमरी स्ट्रक्चर्स असतात. घटकांचे आकार आणि प्रकार कंपाइल-टाइमवर निर्धारित केले जातात.

१०.डेटा संकलनासाठी वापरले जाते:

ॲरे सामान्यतः डेटाच्या संग्रहाचे प्रतिनिधित्व करण्यासाठी वापरले जातात, जसे की संख्या, स्ट्रिंग किंवा ऑब्जेक्ट्सची सूची.

प्रोग्रामिंगमध्ये ॲरे प्रभावीपणे वापरण्यासाठी आणि विविध संगणकीय समस्या सोडवण्यासाठी ही वैशिष्ट्ये समजून घेणे आवश्यक आहे.

३.१.२ एक आयाम आणि द्विमितीय ॲरे(One Dimensional and Two Dimensional Array)

A **one-dimensional array** in Marathi is referred to as "**एक आयामी सरणी".**

एक आयामी सरणी (One-dimensional Array) हे प्रोग्रामिंगमध्ये एक महत्त्वपूर्ण डेटा संरचना आहे, ज्यात एक समूहातील सर्व आइटम्स (माने) एक आकारात, साधारित प्रकारात राखण्यात आता येते. एक आयामी सरणीची सर्व माने एका दिशेने एकत्र स्थानांकित राहतात आणि त्या मानांची ओळख स्थानांक किंवा इंडेक्सद्वारे केली जाते.

I. **एका मितीय ॲरेशी संबंधित काही महत्त्वाचे मुद्दे**

१.घोषणा (Declaration):

- सरणीला घोषित करण्यासाठी त्याच्या डेटा प्रकार आणि नावानुसार स्थानिक वर्णन केला जातो, त्यानंतर त्याच्या आकारानुसार सरणीला स्थानांक निर्दिष्ट केले जाते. उदाहरणार्थ:

```
int numbers[5]; // 5 मानांची एक पंक्ति घोषणा
```

२.सुरूवातीलिया किंवा पुनःस्थापना (Initialization) :

- सरणीच्या मानांची सुरूवात किंवा पुनःस्थापना केली जाऊ शकते. उदाहरणार्थ:

```
int numbers[5] = {1, 2, 3, 4, 5}; // घोषणा केरलेल्या वेळेस सरणीला पुनःस्थापित करा
```

३.ऍक्सेसिंग एलिमेंट्स (Accessing Elements):

एक-आयामी ॲरेमधील घटक त्यांच्या अनुक्रमणिका किंवा स्थितीचा वापर करून प्रवेश करतात. पहिल्या घटकासाठी निर्देशांक 0 पासून सुरू होतो आणि शेवटच्या घटकासाठी (आकार - 1) पर्यंत जातो.

उदाहरणार्थ: `int x = numbers[2]; // Accesses the third element of the array (index 2)`

४.सलग मेमरी वाटप:

एक-आयामी ॲरेमधील घटक संलग्न मेमरी स्थानांमध्ये संग्रहित केले जातात. हे निर्देशांक वापरून कोणत्याही घटकापर्यंत कार्यक्षम यादृच्छिक प्रवेशास अनुमती देते.

५.निश्चित आकार:

एक-आयामी ॲरेचा आकार घोषणेच्या वेळी निश्चित केला जातो आणि रनटाइम दरम्यान बदलला जाऊ शकत नाही.

उदाहरणार्थ: float values[10]; // Declares a float array with 10 elements.

६.ट्रॅव्हर्सल(Traversal):

एक-आयामी ॲरेच्या सर्व घटकांमधून जाण्यासाठी लूप वापरणे सामान्य आहे, जसे की लूपसाठी किंवा असताना. हे प्रत्येक घटकावरील ऑपरेशन्स सुलभ करते.

७.वापर(Usage):

एक-आयामी ॲरे मोठ्या प्रमाणावर घटकांच्या सूची संग्रहित करण्यासाठी आणि हाताळण्यासाठी वापरले जातात, जसे की संख्या, वर्ण किंवा वस्तूंची सूची.

एक डायमेंशनल ॲरे मराठीचे उदाहरण

```
#include <stdio.h>
int main() {
   // Declare an array of integers
   int numbers[5];

   // Input data into the array
   printf("Enter five integers:\n");
   for (int i = 0; i < 5; ++i) {
      printf("Enter number %d: ", i + 1);
      scanf("%d", &numbers[i]);
   }
```

```
    // Display the elements of the array
    printf("\nEntered numbers:\n");
    for (int i = 0; i < 5; ++i) {
        printf("%d ", numbers[i]);
    }
    printf("\n");
    // Calculate and display the sum of the array elements
    int sum = 0;
    for (int i = 0; i < 5; ++i) {
        sum += numbers[i];
    }
    printf("Sum of the numbers: %d\n", sum);
    return 0;
}
```

II)Two-dimensional Array): C मध्ये द्विमितीय ॲरेद्विमितीय ॲरेला ॲरेचा ॲरे म्हणून परिभाषित केले जाऊ शकते. 2D ॲरे मॅट्रिक्स म्हणून आयोजित केले आहे जे पंक्ती आणि स्तंभांचे संकलन म्हणून प्रस्तुत केले जाऊ शकते. तथापि, रिलेशनल डेटाबेस सारखी डेटा संरचना लागू करण्यासाठी 2D ॲरे तयार केले जातात. हे एकाच वेळी मोठ्या प्रमाणात डेटा ठेवण्याची सोय प्रदान करते जे आवश्यक तेथे कोणत्याही फंक्शन्समध्ये पास केले जाऊ शकते.

१. घोषणा आणि स्थापना (Declaration अंड Initialization):

एक दोन-आयामी सरणीची घोषणा करण्यासाठी त्याच्या डेटा प्रकार, साइज, आणि नावानुसार स्थानिक वर्णन केला जातो. उदाहरणार्थ:

int matrix[3][4]; // 3x4 मात्रिक्सची एक दोन-आयामी सरणी

२.स्थानांक प्रवाह(Accessing Elements):

- दोन-आयामी सरणीमध्ये प्रत्येक मानाच्या स्थानांकांचा प्रवाह दोनद्वारे होतो. उदाहरणार्थ:

matrix[0][0] = 1; // पहिल्या पंक्तीच्या पहिल्या स्तंभात 1 ठेवा.

३.प्रिंट करणे:

- दोन-आयामी सरणीला प्रिंट करण्यासाठी, नेस्टेड लूप्स वापरले जातात, ज्यामध्ये एक लूप द्वारे पंक्तिच्या संख्येने आणि दुसर्याने स्तंभाच्या संख्येने माने प्रिंट होतात. उदाहरणार्थ:

```
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 4; j++) {
        printf("%d ", matrix[i][j]);
    }
    printf("\n");
}
```

<u>उदाहरण</u>

```
#include <stdio.h>

int main() {

    // Declare a 2D array of integers (3x3 matrix)

    int matrix[3][3];

    // Input data into the matrix

    printf("Enter elements for a 3x3 matrix:\n");

    for (int i = 0; i < 3; ++i) {

        for (int j = 0; j < 3; ++j) {

            printf("Enter element at position (%d, %d): ", i + 1, j + 1);

            scanf("%d", &matrix[i][j]);

        }

    }
```

```
    // Display the elements of the matrix
    printf("\nEntered matrix:\n");
    for (int i = 0; i < 3; ++i) {
        for (int j = 0; j < 3; ++j) {
            printf("%d\t", matrix[i][j]);
        }
        printf("\n");
    }
    // Calculate and display the sum of the matrix elements
    int sum = 0;
    for (int i = 0; i < 3; ++i) {
        for (int j = 0; j < 3; ++j) {
            sum += matrix[i][j];
        }
    }
    printf("\nSum of the matrix elements: %d\n", sum);
    return 0;
}
```

३.१.३ बहुआयामी अॅरे (concept of multi-dimensionalarrays)

बहुआयामी अॅरे सी प्रोग्रामिंग भाषेतील सर्वात शक्तिशाली वैशिष्ट्यांपैकी एक आहेत. ते तुम्हाला टेबल सारख्या फॉरमॅटमध्ये डेटा साठवण्याची परवानगी देतात, जिथे प्रत्येक पंक्ती आणि स्तंभ अनुक्रमणिका वापरून प्रवेश केला जाऊ शकतो. या ब्लॉग पोस्टमध्ये, आम्ही C मधील बहुआयामी अॅरे पाहणार आहोत, ज्यामध्ये त्यांची वाक्यरचना, उदाहरण वापर आणि आउटपुट समाविष्ट आहे.

बहु-आयामी सरण्यांच्या प्रमुख विशेषतांमध्ये खासगी वाचनायाच्या गोष्टी आहेत:

1. **घोषणा आणि स्थापना(Declaration अंड Initialization):**

   - बहु-आयामी सरण्यांची घोषणा करण्यासाठी त्याच्या डेटा प्रकार, साइज, आणि नावानुसार स्थानिक वर्णन केला जातो. उदाहरणार्थ:

     int matrix[3][4]; // 3x4 मात्रिक्सची बहु-आयामी सरणी

2.स्थानांक प्रवाह(Accessing elements):

- बहु-आयामी सरण्यांत प्रत्येक मानाच्या स्थानिक निर्देशांकांचा प्रवाह त्याच्या आयामानुसार असतो. उदाहरणार्थ:

  matrix[0][0] = 1; // पहिल्या पंक्तीच्या पहिल्या स्तंभात 1 ठेवा.

3.प्रिंट करणे:

- बहु-आयामी सरणीला प्रिंट करण्यासाठी, एकाधिक नेस्टेड लूप्स वापरले जातात. उदाहरणार्थ:

```
for (int i = 0; i < 3; i++) {
   for (int j = 0; j < 4; j++) {
      printf("%d ", matrix[i][j]);
   }
   printf("\n");
}
```

4.बहु-आयामी सरणीचा निर्माण:

- बहु-आयामी सरणीचा निर्माण सामान्यतः एका आयामातील सरणीच्या आधारे होतो. उदाहरणार्थ, 3x4 मॅट्रिक्सची बहु-आयामी सरणी ही ती केलेली जाते:

```
int matrix[3][4] = {
   {1, 2, 3, 4},
   {5, 6, 7, 8},
```

```
    {9, 10, 11, 12}
};
```

एक उदाहरणासह बहु-आयामी सरणी

```
#include <stdio.h>
int main() {
    // 3x4 मॅट्रिक्सची बहु-आयामी सरणी घोषणा
    int matrix[3][4] = {
        {1, 2, 3, 4},
        {5, 6, 7, 8},
        {9, 10, 11, 12}
    };
    // सरणीची माने प्रिंट करा
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 4; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

उपरोक्त प्रोग्राममध्ये **matrix** ही एक 3x4 मात्रिक्स आहे आणि त्यातील माने प्रिंट केली जातात .

प्रिंट केलेले परिणाम:

1 2 3 4

5 6 7 8

9 10 11 12

३.२ ॲरे घोषणा आणि आरंभीकरण. (Array declaration and Initialization)

घोषणा (Declaration):

- आयामांची घोषणा करण्यासाठी, आपल्याला त्याच्या प्रकार, नावानुसार आणि साइज नुसार स्थानिक वर्णन करावंत. उदाहरणार्थ:

int numbers[5]; // पाच मानांची एक आयामी सरणीची घोषणा

पुनर्निर्माण (Initialization):

- आयामांची पुनर्निर्माण करण्यासाठी, आपल्याला त्याच्या प्रकार, नावानुसार आणि साइज नुसार माने पुनःस्थापित करावेत. त्याचे मौखिक उदाहरण:

int numbers[5] = {1, 2, 3, 4, 5}; // पाच मानांची स्थापित आहे.

जर आपल्याला सरणी विना प्रमाणे माने सोडण्याची आवड आहे, तर आपण साइज सुचलेलीस काढून देऊ शकता. प्रकारानुसार माने स्थापित केलेले उदाहरण:

char vowels[] = {'a', 'e', 'i', 'o', 'u'}; // स्वतंत्र साइज नुसार अक्षर स्थापित केले गेले आह

- आपल्याला स्वतंत्र साइज आणि साइज सुचलेलीस माने असलेल्या सरण्यांसह वापरकर्त्याला सरणीत माने सोडण्याची स्वतंत्रता मिळते.

उपरोक्त घडयाच्या उदाहरणात, numbers ही पाच मानांची आयामी सरणी घोषित केलेली आहे आणि त्यातील माने 1, 2, 3, 4, 5 स्थापित केलेली आहे.

३.३ ॲरेवरील ऑपरेशन्स

एकाधिक प्रकारच्या प्रक्रियांसह सरणीवर केलेल्या ऑपरेशन्सचे उदाहरण देऊन, सरणीवर कसे काम केले जाते, हे सांगता येईल:

१.सरणीतील माने समाहित करणे (Sum of Array Elements):

सरणीतील सर्व माने जोडून त्याची एकूण मिळविता येते. उदाहरणार्थ, पुनःस्थापित केलेल्या सरणीवर ऑपरेशन:

```
int numbers[] = {1, 2, 3, 4, 5};

int sum = 0;

for (int i = 0; i < 5; i++) {

   sum += numbers[i];

}

// sum मध्ये एकूण मिळविलेले मान आहे
```

२.सरणीतील माने सुचलेल्या मानांची शोध (Search for a Value in Array):

- सरणीतील माने सुचल्यास त्याच्या स्थानिक निर्देशांकांमुळे शोध केले जाते. उदाहरणार्थ:

```
int numbers[] = {10, 20, 30, 40, 50};

int searchValue = 30;

int position = -1;

for (int i = 0; i < 5; i++) {

   if (numbers[i] == searchValue) {

      position = i;

      break; // सुचलेल्या मानाची पहिली स्थिती मिळताना लूप बंद करा

   }

}
```

// position मध्ये सुचलेल्या मानाची स्थिती आहे, किंवा -1 आहे जर सुचलेल्या मानाची स्थिती मिळली नाही

३.सरणीतील माने वर्गमूल (Square Root of Array Elements):

- सरणीतील प्रत्येक मानाचे वर्गमूल घेतले जाते. उदाहरणार्थ:

```
#include <math.h>

double numbers[] = {4.0, 9.0, 16.0, 25.0};

double squareRoots[4];

for (int i = 0; i < 4; i++) {

   squareRoots[i] = sqrt(numbers[i]);

}

// squareRoots मध्ये सरणीतील प्रत्येक मानाचे वर्गमूल असेल
```

४.सरणीतील माने क्रमवार सांगणे (Display Array Elements in Reverse Order):

- सरणीतील माने क्रमवार पुनर्निर्माण केल्यास, त्या मानांची वाचनायाची क्रमवारी बदलते. उदाहरणार्थ:

```
int numbers[] = {1, 2, 3, 4, 5};

for (int i = 4; i >= 0; i--) {

   // numbers[i] मध्ये सरणीतील मानांची वाचनायाची क्रमवारी बदलते

}
```

५.सरणीतील सर्व माने १ वाढवणे (Increment All Array Elements by 1):

सरणीतील सर्व माने १ वाढवून त्याची मूल्ये अद्यतित होतात. उदाहरणार्थ:

```
 int numbers[] = {1, 2, 3, 4, 5};

for (int i = 0; i < 5; i++) {

   numbers[i] += 1;

}
```

// numbers मध्ये सरणीतील सर्व माने १ वाढविलेली आहेत.

इतर काही महत्त्वाचे ऑपरेशन्स

ट्रॅव्हर्सल: ॲरेच्या घटकांमधून मार्गक्रमण करा.

समाविष्ट करणे: ॲरेमध्ये नवीन घटक समाविष्ट करणे.

हटवणे: ॲरेमधून घटक हटवणे.

शोधत आहे: ॲरेमधील घटक शोधा.

वर्गीकरण: ॲरेमधील घटकांचा क्रम राखणे.

३.४ वर्ण आणि स्ट्रिंग इनपुट/आउटपुट आणि स्ट्रिंग संबंधित ऑपरेशन्स (Character and String input/output and String related operations)

सी प्रोग्रामिंगमधील स्ट्रिंग म्हणजे शून्य वर्ण '\0' सह समाप्त केलेल्या वर्णांचा एक क्रम आहे. C स्ट्रिंग कॅरेक्टर्सच्या ॲरेच्या रूपात साठवले जाते. कॅरेक्टर ॲरे आणि C स्ट्रिंगमधील फरक असा आहे की C मधील स्ट्रिंग '\0' अनन्य अक्षराने संपवली जाते.

मजकूर/वर्ण संग्रहित करण्यासाठी स्ट्रिंगचा वापर केला जातो.

इतर अनेक प्रोग्रामिंग भाषांप्रमाणे, C मध्ये स्ट्रिंग व्हेरिएबल्स सहजपणे तयार करण्यासाठी स्ट्रिंग प्रकार नाही. त्याऐवजी, C मध्ये स्ट्रिंग करण्यासाठी तुम्हाला चार प्रकार वापरणे आवश्यक आहे आणि वर्णांची ॲरे तयार करणे आवश्यक आहे:

उदाहरणार्थ, "हॅलो वर्ल्ड" वर्णांची एक स्ट्रिंग आहे.

```
char greetings[] = "Hello World!";
```

C स्ट्रिंग घोषणा वाक्यरचना (String Declaration Syntax)

C मध्ये स्ट्रिंग घोषित करणे हे एक-आयामी ॲरे घोषित करण्याइतके सोपे आहे. खाली स्ट्रिंग घोषित करण्यासाठी मूलभूत वाक्यरचना आहे.

```
char string_name[size];
```

वरील सिंटॅक्समध्ये string_name हे स्ट्रिंग व्हेरिएबलला दिलेले कोणतेही नाव आहे आणि स्ट्रिंगची लांबी परिभाषित करण्यासाठी आकार वापरला जातो, म्हणजेच स्ट्रिंगमध्ये किती अक्षरे साठवली जातील.

एक अतिरिक्त टर्मिनेटिंग कॅरेक्टर आहे जे नल कॅरेक्टर ('\0') आहे जे सामान्य कॅरेक्टर ॲरेपेक्षा भिन्न असलेल्या स्ट्रिंगची समाप्ती दर्शवण्यासाठी वापरले जाते.

C स्ट्रिंग इनिशियलायझेशन

C मधील स्ट्रिंग वेगवेगळ्या प्रकारे सुरू करता येते. आम्ही एका उदाहरणाच्या मदतीने हे स्पष्ट करू. खाली str नावाची स्ट्रिंग घोषित करण्यासाठी आणि "welcome" सह प्रारंभ करण्यासाठी उदाहरणे आहेत.

आपण C स्ट्रिंग 4 वेगवेगळ्या प्रकारे सुरू करू शकतो जे खालीलप्रमाणे आहेतः

1. आकाराशिवाय स्ट्रिंग लिटरल नियुक्त करणे

स्ट्रिंग लिटरल आकाराशिवाय नियुक्त केले जाऊ शकतात. येथे, स्ट्रिंगचे नाव पॉइंटर म्हणून कार्य करते कारण ते ॲरे आहे.

```
char str[] = "welcome";
```

2. पूर्वनिर्धारित आकारासह स्ट्रिंग लिटरल नियुक्त करणे.

स्ट्रिंग लिटरल पूर्वनिर्धारित आकारासह नियुक्त केले जाऊ शकतात. परंतु आपण नेहमी एका अतिरिक्त जागेचा हिशेब ठेवला पाहिजे जो शून्य वर्णाला नियुक्त केला जाईल. जर आपल्याला n आकाराची स्ट्रिंग संग्रहित करायची असेल तर आपण नेहमी n+1 च्या बरोबरीची किंवा जास्त आकाराची स्ट्रिंग घोषित केली पाहिजे.

```
char str[50] = "welcome";
```

3. आकारासह वर्णानुसार वर्ण नियुक्त करणे.

आपण अक्षरानुसार स्ट्रिंग कॅरेक्टर देखील नियुक्त करू शकतो. परंतु आपण शेवटचे अक्षर '\0' असे सेट करणे लक्षात ठेवले पाहिजे जे शून्य वर्ण आहे.

```
char str[14] = { 'w','e','l','c','o','m','e','\0'};
```

4. Size न करता वर्णानुसार वर्ण नियुक्त करणे.

आम्ही शेवटी NULL अक्षरासह आकाराशिवाय वर्णानुसार वर्ण नियुक्त करू शकतो. स्ट्रिंगचा आकार कंपाइलरद्वारे स्वयंचलितपणे निर्धारित केला जातो.

```
char str[] = { 'w','e','l','c','o','m','e','\0'};
```

साधे उदाहरण

```
#include <stdio.h>
#include <string.h>
int main() {
    // Declare and initialize a Marathi string
    char marathiString[] = "मराठी स्ट्रिंग उदाहरण";

    // Print the Marathi string
    printf("Marathi String: %s\n", marathiString);
    // Get the length of the Marathi string
```

```
    int length = strlen(marathiString);
    printf("Length of Marathi String: %d characters\n", length);
    // Access individual characters in the Marathi string
    printf("Individual characters in Marathi String: ");
    for (int i = 0; i < length; i++) {
        printf("%c ", marathiString[i]);
    }
    printf("\n");
    return 0;
}
```

C स्ट्रिंग फंक्शन्स <string.h>

सी प्रोग्रामिंगमधील स्ट्रिंग्स हाताळण्यासाठी खालील फंक्शन्स वापरली जातात.

1. स्ट्रिंग कॉपी: strncpy()

हे फंक्शन एका स्ट्रिंगमधून दुसऱ्या स्ट्रिंगमध्ये निर्दिष्ट वर्णांची कॉपी करते.

2. स्ट्रिंग कॉन्केट: strncat()

हे फंक्शन दुसऱ्या स्ट्रिंगपासून पहिल्या स्ट्रिंगपर्यंत विशिष्ट वर्णांची संख्या जोडते.

3. स्ट्रिंग तुलना: strncmp()

हे फंक्शन दोन स्ट्रिंगमधील वर्णांच्या निर्दिष्ट संख्येची तुलना करते.

4. प्रथम वर्ण घटना: strchr()

हे फंक्शन स्ट्रिंगमधील वर्णाची पहिली घटना शोधते.

5. शेवटची वर्ण घटना: strrchr()

हे फंक्शन स्ट्रिंगमधील वर्णाची शेवटची घटना शोधते.

6. स्ट्रिंग शोध: strstr()

हे फंक्शन स्ट्रिंगमधील सबस्ट्रिंगची पहिली घटना शोधते.

Example using String Handling Function

```
#include <stdio.h>
#include <string.h>
int main() {
    char str1[50], str2[50];
        // Input two strings
    printf("Enter the first string: ");
    fgets(str1, sizeof(str1), stdin);
    printf("Enter the second string: ");
    fgets(str2, sizeof(str2), stdin);
    // Remove the newline characters from the input
    str1[strcspn(str1, "\n")] = '\0';
    str2[strcspn(str2, "\n")] = '\0';
    // Display the length of each string
    printf("\nLength of the first string: %lu\n", strlen(str1));
    printf("Length of the second string: %lu\n", strlen(str2));


    // Concatenate and display the strings
    strcat(str1, str2);
```

```
    printf("\nConcatenated string: %s\n", str1);
    // Compare the strings
    int compareResult = strcmp(str1, str2);
    if (compareResult == 0) {
        printf("The strings are equal.\n");
    } else {
        printf("The strings are not equal.\n");
    }
    // Copy and display the second string
    strcpy(str1, str2);
    printf("\nCopied string: %s\n", str1);
    return 0;
}
```

३.५ संरचनांची ओळख आणि वैशिष्ट्ये, (Introduction and Features of Structures)

Declaration and Initialization of Structures, array ofstructures. स्ट्रक्चर्सची घोषणा आणि आरंभिकरण, स्ट्रक्चर्सची ॲरे.

C मधील रचना ही एक वापरकर्ता-परिभाषित डेटा प्रकार आहे ज्याचा वापर शक्यतो भिन्न प्रकारच्या वस्तूंचे एकाच प्रकारात गट करण्यासाठी केला जाऊ शकतो. स्ट्रक्ट कीवर्डचा वापर सी प्रोग्रामिंग भाषेतील रचना परिभाषित करण्यासाठी केला जातो. संरचनेतील आयटमला त्याचे सदस्य म्हटले जाते आणि ते कोणत्याही वैध डेटा प्रकाराचे असू शकतात.

सी संरचना घोषणा (C Structure Declaration)

आमच्या प्रोग्राममध्ये वापरण्यापूर्वी आम्हाला C मध्ये रचना घोषित करावी लागेल. स्ट्रक्चर डिक्लेरेशनमध्ये, आम्ही सदस्य व्हेरिएबल्स त्यांच्या डेटाटाइपसह निर्दिष्ट करतो. खालील वाक्यरचना वापरून C मधील रचना घोषित करण्यासाठी आपण struct कीवर्ड वापरू शकतो.

Syntax

```
struct structure_name {
    data_type member_name1;
    data_type member_name1;
    ....
    ....
};
```

प्रोग्राममध्ये रचना वापरण्यासाठी, आपल्याला त्याचे उदाहरण परिभाषित करावे लागेल. रचना प्रकाराचे व्हेरिएबल्स तयार करून आपण ते करू शकतो. आपण दोन पद्धती वापरून स्ट्रक्चर व्हेरिएबल्स परिभाषित करू शकतो:

1. Structure Variable Declaration with Structure Template

```
struct structure_name {
    data_type member_name1;
    data_type member_name1;
    ....
    ....
}variable1, varaible2, ...;
```

2. Structure Variable Declaration after Structure Template

```
// structure declared beforehand
struct structure_name variable1, variable2, .....;
```

Access Structure Members

We can access structure members by using the **(.) dot operator.**

Syntax

structure_name.member1;

strcuture_name.member2;

सी प्रोग्रामिंगमध्ये स्ट्रक्चरचे एक उदाहरण

```
#include <stdio.h>

// Define a structure named 'Person'

struct Person {

    char name[50];

    int age;

    float height;

};

int main() {

    // Declare a variable of type 'struct Person'

    struct Person person1;

    // Input data for the first person

    printf("Enter name: ");

    fgets(person1.name, sizeof(person1.name), stdin);

    printf("Enter age: ");

    scanf("%d", &person1.age);

    printf("Enter height (in meters): ");
```

```
    scanf("%f", &person1.height);

    // Display information about the first person

    printf("\nPerson Information:\n");

    printf("Name: %s", person1.name);

    printf("Age: %d\n", person1.age);

    printf("Height: %.2f meters\n", person1.height);

    return 0;

}
```

स्ट्रक्चर्सचे ॲरे(Array of structure): स्ट्रक्चर हा C मधील डेटा प्रकार आहे जो संबंधित व्हेरिएबल्सच्या गटाला स्वतंत्र घटकांऐवजी एकच एकक म्हणून हाताळण्याची परवानगी देतो. संरचनेत विविध डेटा प्रकारांचे घटक असू शकतात - int, char, float, double, इ. त्यात सदस्य म्हणून ॲरे देखील असू शकतात. अशा ॲरेला स्ट्रक्चरमधील ॲरे म्हणतात. स्ट्रक्चरमधील ॲरे हा स्ट्रक्चरचा सदस्य असतो आणि ज्याप्रमाणे आपण स्ट्रक्चरच्या इतर घटकांमध्ये प्रवेश करतो त्याप्रमाणे त्यात प्रवेश केला जाऊ शकतो.खाली एका प्रोग्रॅमचे प्रात्यक्षिक आहे जे स्ट्रक्चरमध्ये ॲरेची संकल्पना वापरते. हा कार्यक्रम रोल नंबर, ग्रेड आणि विविध विषयांमध्ये मिळवलेले गुण असलेल्या विद्यार्थ्याचे रेकॉर्ड प्रदर्शित करतो. विविध विषयांतील गुण मार्क्स नावाच्या ॲरेखाली साठवले जातात. संपूर्ण रेकॉर्ड उमेदवार नावाच्या संरचनेखाली संग्रहित केला जातो.

// C प्रोग्रामिंगमध्ये **स्ट्रक्चर्सचे ॲरे उदाहरण**

```
#include <stdio.h>

// Define a structure named 'Student'

struct Student {

    char name[50];

    int rollNumber;

    float marks;

};
```

```
int main() {
    // Declare an array of structures of type 'struct Student'
    struct Student students[3];
    // Input data for three students
    for (int i = 0; i < 3; ++i) {
        printf("\nEnter details for student %d:\n", i + 1);
            // Input student name
        printf("Enter name: ");
        fgets(students[i].name, sizeof(students[i].name), stdin);
           // Input roll number
        printf("Enter roll number: ");
        scanf("%d", &students[i].rollNumber);
            // Input marks
        printf("Enter marks: ");
        scanf("%f", &students[i].marks);
            // Consume the newline character in the input buffer
        getchar();
    }
    // Display information about the students
    printf("\nStudent Information:\n");
    for (int i = 0; i < 3; ++i) {
        printf("\nDetails for student %d:\n", i + 1);
```

```
        printf("Name: %s", students[i].name);
        printf("Roll Number: %d\n", students[i].rollNumber);
        printf("Marks: %.2f\n", students[i].marks);
    }
    return 0;
}
```

३.६ प्रकार def, प्रगणित डेटा प्रकार. (Type def, Enumerated Data Type)

3.6.1 सी टाइपडेफ (C typedef)

टाइपडेफ हा एक कीवर्ड आहे जो नवीन नावासह विद्यमान डेटा प्रकार प्रदान करण्यासाठी वापरला जातो. C typedef कीवर्ड आधीपासून अस्तित्वात असलेल्या डेटा प्रकारांचे नाव पुन्हा परिभाषित करण्यासाठी वापरले जाते.जेव्हा प्रोग्राम्समध्ये डेटाटाइपची नावे वापरणे कठीण होते, तेव्हा टाइपडेफ वापरकर्त्याने परिभाषित केलेल्या डेटाटाइपसह वापरले जाते, जे कमांडसाठी उपनाव परिभाषित करण्यासारखेच वागतात.

सी टाइपडेफ सिंटॅक्स

```
typedef existing_name alias_name;
```

C मधील typedef चे उदाहरण

```
typedef long long ll;
```

Program:

```
#include <stdio.h>
// Define a structure named 'Person'
struct Person {
    char name[50];
    int age;
};
```

```
// Use typedef to create an alias 'PersonType' for 'struct Person'

typedef struct Person PersonType;

int main() {

    // Declare a variable of type 'PersonType'

    PersonType person1;

    // Input data for the person

    printf("Enter name: ");

    fgets(person1.name, sizeof(person1.name), stdin);

    printf("Enter age: ");

    scanf("%d", &person1.age);

    // Display information about the person

    printf("\nPerson Information:\n");

    printf("Name: %s", person1.name);

    printf("Age: %d\n", person1.age);

    return 0;

}
```

3.6.2 C मध्ये गणन (किंवा enum).

गणन (किंवा enum) हा C मध्ये वापरकर्ता परिभाषित डेटा प्रकार आहे. त्याचा उपयोग मुख्यत्वे अविभाज्य स्थिरांकांना नावे नियुक्त करण्यासाठी केला जातो, नावांमुळे प्रोग्राम वाचणे आणि देखरेख करणे सोपे होते. यात स्थिर पूर्णांक किंवा पूर्णांक असतात ज्यांना वापरकर्त्याद्वारे नावे दिली जातात. पूर्णांक मूल्यांना नाव देण्यासाठी C मधील enum चा वापर संपूर्ण प्रोग्रामला समान किंवा अगदी भिन्न प्रोग्रामरद्वारे शिकणे, समजणे आणि देखरेख करणे सोपे करते.

सिंटॅक्स: enum State {Working = 1, Failed = 0};

उदाहरण: enum flag{constant1, constant2, constant3, ... };

Program:

```
#include <stdio.h>

// Define an enumeration named 'Month'

enum Month {

    JANUARY,    // 0

    FEBRUARY,   // 1

    MARCH,      // 2

    APRIL,      // 3

    MAY,        // 4

    JUNE,       // 5

    JULY,       // 6

    AUGUST,     // 7

    SEPTEMBER,  // 8

    OCTOBER,    // 9

    NOVEMBER,   // 10

    DECEMBER    // 11

};



int main() {

    // Declare a variable of type 'enum Month'
```

```
enum Month currentMonth;

// Assign a value to the 'currentMonth' variable

currentMonth = JANUARY;

// Display the value of 'currentMonth'

printf("Current month: %d\n", currentMonth);

// Switch statement using enum

switch (currentMonth) {

    case JANUARY:

    case FEBRUARY:

    case DECEMBER:

        printf("It's winter.\n");

        break;

    case MARCH:

    case APRIL:

    case MAY:

        printf("It's spring.\n");

        break;

    case JUNE:

    case JULY:

    case AUGUST:

        printf("It's summer.\n");

        break;
```

```
        case SEPTEMBER:
        case OCTOBER:
        case NOVEMBER:
            printf("It's autumn.\n");
            break;
        default:
            printf("Invalid month.\n");
    }
    return 0;
}
```

संदर्भ:

१. https://www.javatpoint.com/array-of-structures-in-c

२. https://www.geeksforgeeks.org/difference-between-structure-and-array-in-c

३. https://www.scaler.com/topics/c/array-of-structure-in-c

४. https://www.tutorialspoint.com/difference-between-array-and-structure

५. https://www.w3schools.com/c/c_arrays.php

घटक ४

फंक्शन्स

(Functions)

गुण(14)

विषय निष्पत्ती (Course Outcome):युसर डिफाइन फंक्शन वापरून C प्रोग्राम विकसित करा.

घटक निष्पत्ती (Unit Outcome):

४.१ C प्रोग्राममधील फंक्शन्सची गरज स्पष्ट करा.

४.२ C लायब्ररी फंक्शन्सचा समावेश असलेला C प्रोग्राम लिहा.

४.३ C प्रोग्राममध्ये दिलेल्या समस्येसाठी युसर डिफाइन फंक्शन लिहा.

४.४ फंक्शन कॉल करण्यासाठी कॉल बाय व्हॅल्यू आणि कॉल बाय रेफरन्स वापरूनC प्रोग्राम लिहा.

४.५ C प्रोग्राममध्ये रिकर्सिव फंक्शन्स वापरून प्रोग्राम लिहा.

४.१ फंक्शन्सची संकल्पना आणि गरज(Concept and need of functions)

फंक्शन हा कोड किंवा सब प्रोग्राम्सचा एक स्वयंपूर्ण ब्लॉक असतो ज्यात विधानांचा संच असतो जेव्हा ते कॉल केले जाते तेव्हा काही विशिष्ट कार्य किंवा सुसंगत कार्य करते .

कोणत्याही 'सी' प्रोग्राममध्ये किमान एक फंक्शन असते म्हणजे main ()

मुळात फंक्शनचे दोन प्रकार आहेत

1. लायब्ररी फंक्शन(Library Functions)

सर्व प्रोग्रामिंग भाषांमध्ये फंक्शन्स असतात. C मधील लायब्ररी फंक्शन्स देखील C भाषेत इनबिल्ट फंक्शन्स आहेत. ही इनबिल्ट फंक्शन्स काही सामान्य ठिकाणी स्थित आहेत आणि ती लायब्ररी म्हणून ओळखली जाते. सर्व फंक्शन्स विशिष्ट ऑपरेशन कार्यान्वित करण्यासाठी वापरली जातात. या लायब्ररी फंक्शन्सना सामान्यतः पूर्वनिर्धारित आउटपुट प्राप्त करण्यासाठी प्राधान्य दिले जाते. तसेच, या फंक्शन्सच्या क्रिया त्यांच्या शीर्षलेख फायलींमध्ये उपस्थित असतात.

2. वापरकर्ता परिभाषित फंक्शन (User Defined Functions)

वापरकर्ता परिभाषित फंक्शन्स वापरकर्त्याने त्याच्या गरजेनुसार परिभाषित केलेली असते. प्रणाली

परिभाषित फंक्शन सुधारित केले जाऊ शकत नाही; ते फक्त वाचू शकते आणि वापरले जाऊ शकते. हे फंक्शन प्रत्येक C कंपाइलरसह पुरवले जाते या लायब्ररी फंक्शनचे स्त्रोत पूर्व आहेत लिंकरद्वारे कोडला लिंक करून वापरकर्त्याद्वारे पालन केलेला आणि फक्त ऑब्जेक्ट कोड वापरला जातो.

सी प्रोग्रामिंग भाषेसह प्रोग्रामिंगमध्ये फंक्शन्स ही एक आवश्यक संकल्पना आहे. ते कोड संरचनेचा मार्ग प्रदान करतात, पुन्हा वापरण्यायोग्यतेला प्रोत्साहन देतात आणि प्रोग्राम अधिक मॉड्यूलर बनवतात. C मधील फंक्शन्सशी संबंधित काही प्रमुख संकल्पना आणि गरजा येथे आहेत:

1. मॉड्यूलरिटी(Modularity**):**

फंक्शन्स तुम्हाला प्रोग्रामला लहान, आटोपशीर तुकड्यांमध्ये मोडण्याची परवानगी देतात. प्रत्येक फंक्शन एक विशिष्ट कार्य करू शकते, कोड अधिक मॉड्यूलर आणि समजण्यास सोपे बनवते.

2. पुन: उपयोगिता(Reusability**):**

एकदा फंक्शन परिभाषित केल्यावर, ते प्रोग्रामच्या वेगवेगळ्या भागांमध्ये किंवा इतर प्रोग्राममध्ये देखील वापरले जाऊ शकते. हे कोडच्या पुनर्वापराला प्रोत्साहन देते, रिडंडंसी कमी करते आणि देखभाल अधिक सोपी करते.

3. अमूर्तता(Abstraction**):**

फंक्शन्स तुम्हाला विशिष्ट कार्याच्या अंमलबजावणीचे तपशील काढून टाकण्याची परवानगी देतात. फंक्शनच्या वापरकर्त्यांना ते (इंटरफेस) कसे वापरायचे हे माहित असणे आवश्यक आहे, ते कसे लागू केले जाते हे आवश्यक नाही. हे एकूण कार्यक्रम रचना सुलभ करते.

4. वाचनीयता(Readability**):**

प्रोग्रामला लहान फंक्शन्समध्ये विभाजित केल्याने कोड वाचनीयता वाढते. कोडच्या मोठ्या ब्लॉकशी व्यवहार करण्याऐवजी, तुम्ही ज्या विशिष्ट फंक्शनवर काम करत आहात त्यावर लक्ष केंद्रित करू शकता, ज्यामुळे ते समजणे आणि डीबग करणे सोपे होईल.

5. पॅरामीटर पासिंग(Parameter Passing**):**

फंक्शन्स पॅरामीटर्स घेऊ शकतात, त्यांना कॉलिंग कोडमधून डेटा प्राप्त करण्यास अनुमती देतात. हे अधिक लवचिक आणि डायनॅमिक कार्ये तयार करणे शक्य करते जे भिन्न इनपुटसह कार्य करू शकतात.

6. परतावा मूल्ये(Return Values**):**

फंक्शन्स कॉलिंग कोडची मूल्ये देखील परत करू शकतात. फंक्शनमध्ये केलेल्या गणनेचे अभिप्राय किंवा परिणाम प्रदान करण्यासाठी हे उपयुक्त आहे.

7. एन्कॅप्सुलेशन(Encapsulation**):**

फंक्शन्स ऑपरेशन्सचा एक संच एन्कॅप्स्युलेट करतात, उर्वरित प्रोग्राममधून अंतर्गत तपशील लपवतात. हे चांगल्या कोड संस्थेत योगदान देते आणि प्रोग्रामच्या विविध भागांमधील अनपेक्षित परस्परसंवादाची शक्यता कमी करते.

8. कोड ऑर्गनायझेशन(Code Organization)**:**

फंक्शन्स तार्किकरित्या कोड व्यवस्थापित करण्यात मदत करतात. मुख्य प्रोग्राम विविध फंक्शन्स कॉल करू शकतो, प्रत्येक प्रोग्रामच्या कार्यक्षमतेच्या विशिष्ट पैलूसाठी जबाबदार आहे, ज्यामुळे अधिक संरचित आणि देखभाल करण्यायोग्य कोडबेस बनतो.

9. डीबगिंगची सुलभता(Ease of Debugging)**:**

फंक्शन्ससह, डीबगिंग अधिक आटोपशीर बनते. एखादी त्रुटी आढळल्यास, तुम्ही विशिष्ट कार्यावर लक्ष केंद्रित करू शकता, ज्यामुळे समस्या ओळखणे आणि त्यांचे निराकरण करणे सोपे होईल.

उदाहरण:

```
#include <stdio.h>

// Function declaration
int add(int a, int b);

int main() {
    // Function call
    int result = add(5, 3);

    // Output the result
    printf("The sum is: %d\n", result);

    return 0;
}

// Function definition
int add(int a, int b) {
    return a + b;
```

}

या उदाहरणात, add फंक्शन दोन पॅरामीटर्स घेते, जोडते आणि परिणाम देते. मुख्य प्रोग्राम अधिक वाचनीय आहे आणि त्याच्या विशिष्ट कार्यावर लक्ष केंद्रित करतो, तर add फंक्शन अतिरिक्त कार्यक्षमता हाताळते.

४.२ लायब्ररी फंक्शन्स: मॅथ फंक्शन्स, स्ट्रिंग हँडलिंग फंक्शन्स, इतर विविध फंक्शन्स जसे की getchar(), putchar(), malloc(), calloc()(Library functions: Math functions, String handling functions, other miscellaneous functions)

सी प्रोग्रामिंगमध्ये, अनेक मानक लायब्ररी आहेत जी वेगवेगळ्या उद्देशांसाठी विविध कार्ये प्रदान करतात. सी मधील काही सामान्य लायब्ररी फंक्शन्स येथे आहेत:

गणित कार्ये Math Functions (<math.h>**):**

- सामान्य गणितीय क्रिया
- उदाहरणे: sqrt(), pow(), sin(), cos(), tan(), log(), exp(), etc.

C मधील काही सामान्य गणित कार्यांचे येथे थोडक्यात स्पष्टीकरण दिले आहे:

1. sqrt(x):

- दिलेल्या संख्येचे वर्गमूळ `x` काढते.
- उदाहरण: `sqrt(16.0)` `4.0` मिळवते.

2. pow(x, y):

- `x` ला `y` च्या घातावर वाढवते.
- उदाहरण: `pow(2.0, 3.0)` `8.0` मिळवते.

3. sin(x), cos(x), tan(x):

- कोन `x` (रेडियनमध्ये) च्या साइन, कोसाइन आणि स्पर्शिकेची गणना करते.
- उदाहरण: `sin(0.0)` `0.0` मिळवते.

4. log(x):

- दिलेल्या संख्येच्या `x` च्या नैसर्गिक लॉगरिदमची गणना करते.
- उदाहरण: `log(2.71828)` `1.0` (अंदाजे) मिळवते.

5. exp(x):

- घातांकीय कार्याची गणना करते, `e` (युलरची संख्या) `x` च्या घातापर्यंत वाढवते.
- उदाहरण: `exp(1.0)` `2.71828` (अंदाजे) मिळवते.

ही फंक्शन्स C मधील `<math.h>` लायब्ररीचा भाग आहेत आणि सामान्यतः वैज्ञानिक आणि अभियांत्रिकी अनुप्रयोगांमध्ये गणितीय गणनांसाठी वापरली जातात. ही फंक्शन्स वापरण्यासाठी तुमच्या C प्रोग्रामच्या सुरुवातीला `<math.h>` हेडर समाविष्ट केल्याची खात्री करा.

```
#include <math.h>
#include <stdio.h>

int main() {
   double num = 16.0;
   double squareRoot = sqrt(num);
   printf("Square root of %f is %f\n", num, squareRoot);
   return 0;
}
```

2. स्ट्रिंग हँडलिंग फंक्शन्स (String Handling Functions (**<string.h>**)):

- स्ट्रिंग्सवरील ऑपरेशन्स (Operations on strings)
- उदाहरण: strlen(), strcpy(), strcat(), strcmp(), strstr(), etc.

1. strlen(str): **स्ट्रिंगच्या लांबीची गणना करते.**

उदाहरण:

```
#include <string.h>
#include <stdio.h>

int main() {
   char str[] = "Hello, World!";
   int length = strlen(str);
   printf("Length of the string: %d\n", length);
   return 0;
}
```

2. strcpy(dest, src): स्रोत स्ट्रिंग src ची सामग्री गंतव्य स्ट्रिंग डेस्टवर कॉपी करते.

उदाहरण:

```
#include <string.h>
#include <stdio.h>

int main() {
   char src[] = "Hello, World!";
   char dest[20];
   strcpy(dest, src);
   printf("Copied string: %s\n", dest);
   return 0;
}
```

3. strcat(dest, src):

स्रोत स्ट्रिंग src ची सामग्री गंतव्य स्ट्रिंग डेस्टशी जोडते (जोडते).

उदाहरण:

```
#include <string.h>
#include <stdio.h>

int main() {
    char str1[] = "Hello";
    char str2[] = " World!";

    // Concatenate str2 to str1
    strcat(str1, str2);

    printf("Concatenated string: %s\n", str1);
    return 0;
}
```

4. strcmp(str1, str2):

दोन स्ट्रिंग्सची तुलना शब्दकोषानुसार करते.

स्ट्रिंग समान असल्यास 0 मिळवते, str1 मोठे असल्यास सकारात्मक मूल्य आणि str2 मोठे असल्यास ऋण मूल्य.

उदाहरण:

```
#include <string.h>
#include <stdio.h>

int main() {
    char str1[] = "apple";
    char str2[] = "banana";
    int result = strcmp(str1, str2);
```

```
    printf("Comparison result: %d\n", result);
    return 0;
}
```

5. strstr(str, substr):

स्ट्रिंग स्ट्रिंगमध्ये सबस्ट्रिंग सबस्ट्रिंगची पहिली घटना शोधते.
पहिल्या घटनेसाठी पॉइंटर परत करते, किंवा न आढळल्यास NULL.
उदाहरण:

```
#include <string.h>
#include <stdio.h>

int main() {
    char str[] = "Hello, World!";
    char substr[] = "World";
    char *result = strstr(str, substr);
    if (result != NULL) {
        printf("Substring found at index: %ld\n", result - str);
    } else {
        printf("Substring not found\n");
    }
    return 0;
}
```

ही स्ट्रिंग हँडलिंग फंक्शन्स C मधील <string.h> लायब्ररीचा भाग आहेत आणि C प्रोग्राम्समधील स्ट्रिंग्समध्ये हाताळणी आणि कार्य करण्यासाठी आवश्यक आहेत.

3. इनपुट/आउटपुट कार्ये Input/Output Functions (<stdio.h>):

इनपुट/आउटपुट ऑपरेशन्स.

उदाहरणे: printf(), scanf(), getchar(), putchar(), इ.

C मधील काही सामान्य इनपुट/आउटपुट फंक्शन्ससाठी येथे संक्षिप्त स्पष्टीकरण आणि उदाहरणे आहेत:

1. printf(स्वरूप, ...): मानक आउटपुट (सामान्यतः कन्सोल) वर स्वरूपित आउटपुट मुद्रित करते.

उदाहरण:

```
#include <stdio.h>
    int main() {
    int num = 42;
    printf("The value of num is %d\n", num);
    return 0;
}
```

2. scanf(format, ...):निर्दिष्ट फॉरमॅटवर आधारित मानक इनपुट (सामान्यतः कन्सोल) मधून इनपुट वाचते.

उदाहरण:

```
#include <stdio.h>

int main() {
    int num;
    printf("Enter an integer: ");
    scanf("%d", &num);
    printf("You entered: %d\n", num);
    return 0;
}
```

3. getchar():मानक इनपुटमधून एकल वर्ण वाचतो.

उदाहरण:

```
#include <stdio.h>

int main() {
    char ch;
    printf("Enter a character: ");
    ch = getchar();
    printf("You entered: ");
    putchar(ch);
    return 0;
}
```

4. putchar(ch):

मानक आउटपुटवर एकल वर्ण लिहितो.

उदाहरण:

```
#include <stdio.h>

int main() {
    char ch = 'A';
    putchar(ch);
    return 0;
}
```

5. gets(str):

- मानक इनपुटमधून मजकूराची एक ओळ वाचते आणि स्ट्रिंगमध्ये संग्रहित करते.
- सुरक्षा धोक्यांमुळे अलीकडील C मानकांमध्ये वगळण्यात आले आहे. त्याऐवजी `fgets` वापरण्याची शिफारस

केली जाते.

6. puts(str):

- स्टँडर्ड आउटपुटवर एक स्ट्रिंग लिहिते आणि त्यानंतर नवीन ओळ वर्ण.

- उदाहरण:

```
#include <stdio.h>

int main() {
    char str[] = "Hello, World!";
    puts(str);
    return 0;
}
```

7. fgets(str, size, stdin):

- मानक इनपुटमधून मजकूराची एक ओळ वाचते आणि स्ट्रिंगमध्ये संग्रहित करते. हे तुम्हाला वाचण्यासाठी जास्तीत जास्त अक्षरांची संख्या निर्दिष्ट करण्यास अनुमती देते.

- उदाहरण:

```
#include <stdio.h>

int main() {
    char str[50];
    printf("Enter a sentence: ");
    fgets(str, sizeof(str), stdin);
    printf("You entered: %s", str);
    return 0;
}
```

ही इनपुट/आउटपुट फंक्शन्स C मधील `<stdio.h>` लायब्ररीचा भाग आहेत आणि वापरकर्त्यांशी संवाद साधण्यासाठी आणि C प्रोग्राममध्ये माहिती प्रदर्शित करण्यासाठी महत्त्वपूर्ण आहेत. लक्षात ठेवा की उदाहरणे उदाहरणासाठी सोपी केली आहेत.

4. मेमरी वाटप कार्ये Memory allocation Functions(<stdlib.h>): डायनॅमिक मेमरी वाटप.
उदाहरणे: malloc(), calloc(), realloc(), free(), इ.

C मधील काही सामान्य मेमरी वाटप फंक्शन्ससाठी येथे स्पष्टीकरण आणि उदाहरणे आहेत:

1. malloc(size):

बाइट्समध्ये निर्दिष्ट आकाराच्या मेमरी ब्लॉकचे वाटप करते.
उदाहरण:

```
#include <stdlib.h>
#include <stdio.h>

int main() {
    int *arr;
    int size = 5;

    // Allocate memory for an integer array
    arr = (int*)malloc(size * sizeof(int));

    // Check if allocation was successful
    if (arr != NULL) {
        // Use the allocated memory

        // Don't forget to free the allocated memory when done
```

```
        free(arr);
    } else {
        printf("Memory allocation failed\n");
    }

    return 0;
}
```

2. calloc(num, size):संख्या घटकांच्या अरेसाठी, प्रत्येक आकाराच्या आकाराच्या बाइट्ससाठी मेमरी ब्लॉक वाटप करते.

मेमरी शून्यावर सुरू केली आहे.

उदाहरण:

```
#include <stdlib.h>
#include <stdio.h>

int main() {
    int *arr;
    int size = 5;

    // Allocate memory for an integer array and initialize to zero
    arr = (int*)calloc(size, sizeof(int));

    // Check if allocation was successful
    if (arr != NULL) {
        // Use the allocated memory

        // Don't forget to free the allocated memory when done
        free(arr);
```

```
    } else {
        printf("Memory allocation failed\n");
    }

    return 0;
}
```

3. realloc(ptr, newSize):ptr द्वारे निर्देशित केलेल्या मेमरीच्या आधी वाटप केलेल्या ब्लॉकचा आकार निर्दिष्ट नवीन आकारात बदलतो.

उदाहरण:

```
#include <stdlib.h>
#include <stdio.h>

int main() {
    int *arr;
    int size = 5;

    // Allocate memory for an integer array
    arr = (int*)malloc(size * sizeof(int));

    // ... use the allocated memory ...

    // Reallocate memory for a larger array
    int newSize = 10;
    arr = (int*)realloc(arr, newSize * sizeof(int));

    // Check if reallocation was successful
    if (arr != NULL) {
```

```
        // Use the reallocated memory

        // Don't forget to free the allocated memory when done
        free(arr);
    } else {
        printf("Memory reallocation failed\n");
    }

    return 0;
}
```

4. free(ptr):malloc, calloc, **किंवा** realloc **द्वारे पूर्वी वाटप केलेल्या मेमरी ब्लॉकचे वाटप रद्द करते**.

उदाहरण:

```
#include <stdlib.h>

int main() {
    int *arr;
    int size = 5;

    // Allocate memory for an integer array
    arr = (int*)malloc(size * sizeof(int));

    // ... use the allocated memory ...

    // Free the allocated memory when done
    free(arr);

    return 0;
```

```
}
```

ही मेमरी ऍलोकेशन फंक्शन्स C मधील <stdlib.h> लायब्ररीचा भाग आहेत आणि डायनॅमिक मेमरी व्यवस्थापनासाठी वापरली जातात. C प्रोग्राममधील मेमरी लीक आणि अपरिभाषित वर्तन टाळण्यासाठी मेमरी योग्यरित्या व्यवस्थापित करणे महत्वाचे आहे.

४.३ वापरकर्ता परिभाषित फंक्शन्स लिहिणे - फंक्शन डेफिनिशन, फंक्शन्स डिक्लेरेशन, फंक्शन कॉल, व्हेरिएबल्सची स्कोप - लोकल व्हेरिएबल्स, ग्लोबल व्हेरिएबल्स (Writing User defined functions - function definition, functions declaration, function call, scope of variables - local variables, global variables)

C मध्ये वापरकर्ता-परिभाषित फंक्शन्स लिहिण्यामध्ये तीन मुख्य पायऱ्यांचा समावेश होतो: फंक्शन डेफिनिशन, फंक्शन डिक्लेरेशन आणि फंक्शन कॉल. याव्यतिरिक्त, स्थानिक आणि जागतिक चलांसह व्हेरिएबल्सची व्याप्ती समजून घेणे महत्वाचे आहे. चला उदाहरणांसह तपशीलवार जाऊ:

1. कार्य व्याख्या(Function Defination):

फंक्शनची कार्यक्षमता बनवणाऱ्या विधानांसह फंक्शनचा मुख्य भाग इथेच परिभाषित करतो.

```
// Function Definition
int add(int a, int b) {
    int sum = a + b;
    return sum;
}
```

या उदाहरणात, आम्ही add नावाचे फंक्शन परिभाषित करतो जे दोन पॅरामीटर्स (a आणि b) घेते, त्यांना एकत्र जोडते, परिणाम स्थानिक व्हेरिएबल बेरीजमध्ये संग्रहित करते आणि नंतर परिणाम देते.

2. कार्य घोषणा(Function Declaration):

कंपाइलरला फंक्शनच्या प्रोटोटाइपबद्दल (त्याचे नाव, रिटर्न प्रकार आणि पॅरामीटर प्रकार) वास्तविक फंक्शनच्या व्याख्यापूर्वी माहिती देण्याचा हा एक मार्ग आहे. हे फंक्शन कॉल करण्यापूर्वी हेडर फाइलमध्ये किंवा थेट कोडमध्ये केले जाते.

```
// Function Declaration
int add(int a, int b);
```

ही घोषणा कंपायलरला सांगते की add नावाचे फंक्शन अस्तित्वात आहे जे दोन int पॅरामीटर्स घेते आणि एक int परत करते. हे कंपाइलरला फंक्शनला योग्यरित्या कसे कॉल करायचे हे समजण्यास मदत करते.

3. फंक्शन कॉल(Function Call):

या ठिकाणी तुम्ही तुमच्या कोडमधील फंक्शनला आवश्यकतेनुसार वितर्क प्रदान करता आणि फंक्शनमध्ये रिटर्न प्रकार असल्यास ते मूल्य मिळवते.

```
// Function Call
int result = add(5, 3);
```

येथे, आम्ही 5 आणि 3 युक्तिवादांसह add फंक्शन म्हणतो. जोडणीचा परिणाम व्हेरिएबल परिणामामध्ये संग्रहित केला जातो.

4. चलांची व्याप्ती (Scope of Variables):

स्थानिक चल (Local Variables):

फंक्शनमध्ये घोषित व्हेरिएबल्स त्या फंक्शनसाठी स्थानिक असतात.

ज्या फंक्शनमध्ये ते घोषित केले जातात त्यामध्येच ते प्रवेशयोग्य असतात.

```
int multiply(int x, int y) {
   // Local variables x, y, and product
   int product = x * y;
   return product;
}
```

ग्लोबल व्हेरिएबल्स(Global Variables)**:**

कोणत्याही फंक्शनच्या बाहेर घोषित व्हेरिएबल्स हे ग्लोबल व्हेरिएबल्स असतात.

ते संपूर्ण कार्यक्रमात प्रवेश करण्यायोग्य आहेत.

उदाहरण:

```
// Global variable
```

```
int globalVar = 10;

int main() {
    // Accessing globalVar in main function
    printf("Global variable: %d\n", globalVar);
    return 0;
}
```

हे सर्व एकत्र ठेवणे:

येथे एक संपूर्ण उदाहरण आहे ज्यात कार्य व्याख्या, घोषणा आणि कॉल समाविष्ट आहे:

```
// Function Declaration
int add(int a, int b);

int main() {
    // Function Call
    int result = add(5, 3);

    // Displaying the result
    printf("The sum is: %d\n", result);

    return 0;
}

// Function Definition
int add(int a, int b) {
    // Local variable sum
    int sum = a + b;
    return sum;
}
```

या उदाहरणात, मुख्य फंक्शनच्या आधी फंक्शन ऍड घोषित केले जाते आणि वास्तविक व्याख्या कोडमध्ये नंतर दिली जाते. ऍड फंक्शन दोन संख्या जोडते आणि परिणाम देते. मुख्य फंक्शन याला ऍड फंक्शन म्हणतो आणि परिणाम प्रदर्शित करतो. बेरीज व्हेरिएबल ऍड फंक्शनसाठी स्थानिक आहे, तर परिणाम मुख्य कार्यासाठी स्थानिक आहे.

४.४ फंक्शन पॅरामीटर्स: पॅरामीटर पासिंग- मूल्यानुसार कॉलFunction parameters: (Parameter passing- call by value & call by reference, function return values, function return types ,declaring function return types, The 'return' statement)

1. फंक्शन पॅरामीटर्स:

पॅरामीटर्स हे कॉलिंग कोडमधून मूल्ये प्राप्त करण्यासाठी फंक्शनमध्ये वापरलेले चल आहेत. ते फंक्शनला अपेक्षित असलेले इनपुट परिभाषित करतात.

```
#include <stdio.h>

// Function declaration
void printNumbers(int a, int b);

int main() {
   int x = 5, y = 10;

   // Function call with arguments x and y
   printNumbers(x, y);

   return 0;
}
```

```
// Function definition
void printNumbers(int a, int b) {
    printf("Numbers: %d and %d\n", a, b);
}
```

या उदाहरणात, printNumbers हे एक फंक्शन आहे जे दोन पॅरामीटर्स a आणि b घेते आणि त्यांना प्रिंट करते.

2. पॅरामीटर पासिंग(Parameter Passing):

मूल्यानुसार कॉल करा(Call by Value):

वास्तविक पॅरामीटर्सची मूल्ये औपचारिक पॅरामीटर्समध्ये कॉपी केली जातात. फंक्शनमधील औपचारिक पॅरामीटर्समध्ये केलेले बदल वास्तविक पॅरामीटर्सवर परिणाम करत नाहीत.

```
#include <stdio.h>

void addTen(int num) {
    num = num + 10;
    printf("Inside function: %d\n", num);
}

int main() {
    int num = 5;

    // Function call with argument num
    addTen(num);

    // num remains unchanged outside the function
    printf("Outside function: %d\n", num);

    return 0;
}
```

संदर्भानुसार कॉल करा(Call by Reference**):**

वास्तविक पॅरामीटर्सचा पत्ता औपचारिक पॅरामीटर्सकडे पाठविला जातो.

फंक्शनमधील औपचारिक पॅरामीटर्समध्ये केलेले बदल वास्तविक पॅरामीटर्सवर परिणाम करतात.

```
#include <stdio.h>

void addTenByReference(int *num) {
    *num = *num + 10;
    printf("Inside function: %d\n", *num);
}

int main() {
    int num = 5;

    // Function call with the address of num
    addTenByReference(&num);

    // num is changed outside the function
    printf("Outside function: %d\n", num);

    return 0;
}
```

3. फंक्शन रिटर्न व्हॅल्यू (Function Return Values)**:**

फंक्शन्स रिटर्न स्टेटमेंट वापरून कॉलिंग कोडची व्हॅल्यू परत करू शकतात.

```
#include <stdio.h>


// Function declaration with return type
int square(int num);
```

```
int main() {
    int x = 5;

    // Function call and assignment of the result
    int result = square(x);

    // Display the result
    printf("Square of %d is: %d\n", x, result);

    return 0;
}

// Function definition with return type
int square(int num) {
    return num * num;
}
```

4.फंक्शन रिटर्न प्रकार (Function Return Types):

C मधील फंक्शन्समध्ये रिटर्नचे प्रकार असतात, जे फंक्शन कोणत्या प्रकारची व्हॅल्यू देईल ते निर्दिष्ट करतात. फंक्शन डिक्लेरेशन आणि व्याख्येमध्ये फंक्शनच्या नावापूर्वी ते घोषित केले जाते.

```
#include <stdio.h>

// Function declaration with return type
float divide(float a, float b);

int main() {
    float x = 10.0, y = 2.0;
```

```
    // Function call and assignment of the result
    float result = divide(x, y);

    // Display the result
    printf("Result of division: %.2f\n", result);

    return 0;
}

// Function definition with return type
float divide(float a, float b) {
    if (b != 0) {
        return a / b;
    } else {
        printf("Error: Division by zero\n");
        return 0;
    }
}
```

5. फंक्शन रिटर्न प्रकार घोषित करणे (Declaring Function Return Types):

```
#include <stdio.h>

// Function declaration with return type
int findMax(int num1, int num2);

int main() {
    int a = 5, b = 8;

    // Function call
    int max = findMax(a, b);
```

```
    // Display the result
    printf("Maximum between %d and %d is: %d\n", a, b, max);

    return 0;
}

// Function definition with return type
int findMax(int num1, int num2) {
    return (num1 > num2) ? num1 : num2;
}
```

6. 'रिटर्न' विधान(The 'return' Statement):

रिटर्न स्टेटमेंटचा वापर फंक्शनची अंमलबजावणी समाप्त करण्यासाठी आणि कॉलिंग कोडला मूल्य परत करण्यासाठी केला जातो.

```
#include <stdio.h>

// Function declaration with return type
int multiply(int a, int b);

int main() {
    int x = 3, y = 4;

    // Function call and assignment of the result
    int result = multiply(x, y);

    // Display the result
    printf("Product: %d\n", result);
```

```
    return 0;
}

// Function definition with return type
int multiply(int a, int b) {
    int product = a * b;

    // Return the product to the calling code
    return product;
}
```

या उदाहरणात, गुणाकार फंक्शन दोन संख्यांच्या गुणाकाराची गणना करते आणि रिटर्न स्टेटमेंट वापरून परिणाम मिळवते.

या संकल्पना समजून घेतल्याने तुम्ही C मधील फंक्शन्स वापरून अधिक मॉड्यूलर आणि पुन्हा वापरता येण्याजोगा कोड तयार करू शकाल.

४.५ आवर्ती कार्ये(Recursive functions)

रिकर्सिव फंक्शन हे एक फंक्शन आहे जे त्याच्या अंमलबजावणी दरम्यान स्वतःला कॉल करते. रिकर्सिव्ह फंक्शन्स उपयोगी असतात जेव्हा एखादी समस्या मूळ समस्येप्रमाणेच लहान उपप्रॉब्लेम्समध्ये मोडली जाऊ शकते. रिकर्सिव्ह फंक्शन्समध्ये बेस केस आणि रिकर्सिव्ह केस असतात. बेस केस अशी स्थिती परिभाषित करते ज्या अंतर्गत फंक्शन स्वतःला कॉल करणे थांबवते, अनंत पुनरावृत्ती रोखते.

येथे उदाहरणासह तपशीलवार स्पष्टीकरण आहे:

उदाहरण: फॅक्टोरियल गणना

नॉन-ऋणात्मक पूर्णांक n च्या फॅक्टोरियलची गणना करण्याच्या उदाहरणाचा विचार करूया. n चा फॅक्टोरियल, n! म्हणून दर्शविला जातो, हा n पेक्षा कमी किंवा समान असलेल्या सर्व धन पूर्णांकांचा गुणाकार आहे.

```
#include <stdio.h>

// Recursive function to calculate factorial
int factorial(int n) {
```

```
    // Base case: factorial of 0 is 1
    if (n == 0) {
        return 1;
    } else {
        // Recursive case: n! = n * (n-1)!
        return n * factorial(n - 1);
    }
}

int main() {
    int num = 5;

    // Function call
    int result = factorial(num);

    // Display the result
    printf("Factorial of %d is: %d\n", num, result);

    return 0;
}
```

स्पष्टीकरण:

बेस केस:

बेस केस ही अशी स्थिती आहे जिथे फंक्शन स्वतःला कॉल करत नाही आणि ज्ञात परिणाम देते. या प्रकरणात, जर n 0 असेल, तर फॅक्टोरियल 1 असेल.

आवर्ती केस:

रिकर्सिव्ह केस म्हणजे फंक्शन स्वतःला समस्येच्या लहान किंवा सोप्या आवृत्तीसह कॉल करते. या उदाहरणात, n चे गुणनूज n * गुणन्य (n - 1) म्हणून मोजले जाते.

अंमलबजावणी:

जेव्हा फॅक्टोरियल फंक्शनला num = 5 ने कॉल केले जाते, तेव्हा ते प्रथम num 0 (बेस केस) आहे का ते तपासते. ते नसल्यामुळे, ते 5 * फॅक्टोरियल (4) ची गणना करून, रिकर्सिव केसकडे जाते. बेस केस (फॅक्टोरियल(0)) पर्यंत पोहोचेपर्यंत ही प्रक्रिया चालू राहते आणि कॉल स्टॅकचा बॅकअप परिणाम प्रसारित केला जातो.

आउटपुट:

नंतर परिणाम मुख्य फंक्शनमध्ये प्रदर्शित केला जातो, 5 चे फॅक्टोरियल दर्शवितो.

आवर्ती कार्यांसाठी महत्वाचे विचार:

बेस केस:

अनंत पुनरावृत्ती टाळण्यासाठी बेस केस असल्याची नेहमी खात्री करा.

बेस केसमध्ये अभिसरण:

अभिसरण हमी देण्यासाठी रिकर्सिव कॉल्स बेस केसकडे नेले पाहिजेत.

कार्यक्षमता:

आवर्ती उपाय नेहमी सर्वात कार्यक्षम असू शकत नाहीत. काही समस्यांसाठी, पुनरावृत्तीचा दृष्टीकोन अधिक योग्य असू शकतो.

मेमरी वापर:

रिकर्सिव्ह फंक्शन्स कॉल स्टॅक वापरतात आणि जास्त रिकर्शनमुळे स्टॅक ओव्हरफ्लो होऊ शकतो. टेल रिकर्सन ऑप्टिमायझेशन हे एक तंत्र आहे जे काही कंपाइलर टेल-रिकर्सिव्ह फंक्शन्स ऑप्टिमाइझ करण्यासाठी वापरतात.

आवर्ती कार्ये मोहक आहेत आणि विशिष्ट समस्यांचे संक्षिप्त निराकरण देऊ शकतात, विशेषत: नैसर्गिकरित्या पुनरावर्ती संरचना असलेल्या. बेस केस आणि रिकर्सिव्ह केस समजून घेणे योग्य आणि कार्यक्षम रिकर्सिव्ह फंक्शन्स तयार करण्यासाठी महत्वाचे आहे

संदर्भ:

1. https://www.geeksforgeeks.org/c-library-functions/

2. https://www.programiz.com/c-programming/library-function

3. https://www.ibm.com/docs/en/i/7.3?topic=extensions-standard-c-library-functions-table-by-name

घटक ५
पॉइंटर्स (Pointers)

गुण (१२)

विषय निष्पत्ती (Course Outcome):

पॉइंटर वापरून C प्रोग्राम लिहा

घटक निष्पत्ती (Unit Outcome):

१. पॉइंटर व्हेरिएबल घोषित व आणि परिभाषित करा.

२. पॉइंटर व्हेरिएबल्स चा पत्ता व मूल्य प्रिंट करण्यासाठी C प्रोग्राम लिहा.

३. पॉइंटर वापरून अंकगणित क्रिया करण्यासाठी C प्रोग्राम लिहा.

४. पॉइंटर्स वापरून अॅरेवर ऑपरेशन्स करण्यासाठी C प्रोग्राम लिहा.

५. पॉइंटर वापरून स्ट्रिंग संबंधित ऑपरेशन्स स्पष्ट करा.

६. स्ट्रक्चरचे व्हेरिएबल पॉइंटर चा वापर करून ऍक्सेस करा

५.१.पॉइंटरचा परिचय : व्याख्या, वापर, '*' आणि '&' ऑपरेटर, पॉइंटर घोषित करणे, इनिशलाइज करणे,पॉइंटर्स चा वापर करणे.

पॉइंटर हे एक व्हेरिएबल आहे जे दुसऱ्या व्हेरिएबलचा मेमरी ॲड्रेस स्टोर करते . पॉइंटर च्या मदतीने मेमरी लोकेशन द्वारे variable च्या value मध्ये बदल करता येतात. पॉईंटर हे C प्रोग्रामिंग साठी एक महत्वाचे आणि शक्तिशाली वैशिट्य आहे .पॉईंटर डायनॅमिक मेमरी अलोकेशन ला मान्यता देते.काही C प्रोग्रामिंग कार्ये पॉइंटर्ससह अधिक सहजतेने पार पाडली जातात आणि इतर कार्ये, जसे की डायनॅमिक मेमरी वाटप, पॉइंटर्स वापरल्याशिवाय करता येत नाही. त्यामुळे परिपूर्ण C प्रोग्रामर होण्यासाठी पॉइंटर्स शिकणे आवश्यक आहे.

५.१.१ पॉइंटर व्याख्या: C पॉइंटर हा एक व्हेरिएबल आहे ज्यामध्ये मेमरी चा ऍड्रेस किंवा ठिकाण साठविले जाते. हा मेमरी पत्ता समान डेटा प्रकाराच्या दुसऱ्या व्हेरिएबलचा मेमरी ऍड्रेस असतो.सोप्या शब्दात, जर एका व्हेरिएबलने दुसऱ्या व्हेरिएबलचा पत्ता संग्रहित केला असेल तर पहिला व्हेरिएबल दुसऱ्या व्हेरिएबलकडे निर्देश करेल (pointed to second variable) असे म्हणता येईल.

५.१.२ '&' ऑपरेटर आणि * ऑपरेटर('&' Operator and '*' Operator)

& ऑपरेटरला संदर्भ (referencing) ऑपरेटर म्हणून संबोधले जाते.पॉइंटर काय आहेत आणि ते काय करू शकतात हे समजून घेण्यापूर्वी, "मेमरी स्थानाचा पत्ता"(Address memory location) म्हणजे काय हे समजून घेणे आवश्यक आहे. जेव्हा जेव्हा व्हेरिएबल परिभाषित(declare) केले जातात तेव्हा त्याच्यासाठी मेमरी स्थान नियुक्त केले जाते. आणि त्या स्थानावर व्हेरिएबल ची value स्टोर केली जाते '&' चिन्ह वापरून आपण हा मेमरी पत्ता तपासू शकतो. जर var हे व्हेरिएबलचे नाव असेल, तर &var त्याचा पत्ता(address) देईल.

C पॉइंटरला * ऑपरेटर वापरून घोषित केले जाते * या ऑपरेटर ला इनडायरेक्शन किंवा डीरेफेरन्सिंग ऑपरेटर असेही म्हणतात. * ऑपरेटर च्या मदतीने पॉइंटर ने पॉईंट केलेली value मिळवता येते.

खालील C प्रोग्राम मध्ये var , &var हे क्रमश: var या व्हेरिएबल ची value व ऍड्रेस दर्शवितात ते प्रोग्राम च्या आउटपुट मध्ये स्पष्ठ होते .

```
#include <stdio.h>
  int main()
{
  int var = 10;
  printf("Value of variable var=%d\n",var);       //printing value
  printf("Address of variable var=%u", &var);   //printing memory address
  printf("Value of variable var=%d", *&var);
   return 0;
}
```

Output:

Value of variable var=10

Address of variable var=3428699500

Value of variable var=10

५.१.३ पॉइंटर घोषित व इनिशिलाईज करणे (declaring pointer & Initilizing pointer)

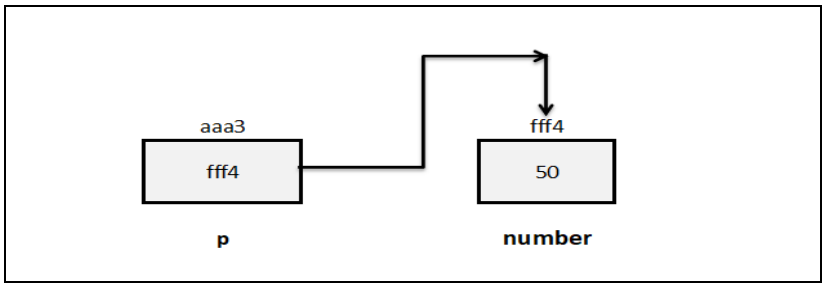
Syntax

Data type *ptrvariable;

Ptrvariable=&variable;

येथे data type पॉईंटर पॉईंट करत असलेल्या variable चा type असेल. पॉईंटर घोषित (declare) केल्यानंतर पॉईंटर मध्ये variable चा ऍड्रेस स्टोर करून पॉईंटर ला ऍड्रेस value देता येते. त्यासाठी ऍड्रेस ऑफ (&) ऑपरेटर variable च्या नावापूर्वी वापरावा लागतो.

C पॉइंटर डिक्लेर किंवा घोषित करण्यासाठी * सिंबॉल/ऑपरेटर वापरला जातो. पॉइंटरला डीरेफरन्स (dereference) करण्यासाठी हा ऑपरेटर गरजेचा आहे. म्हणजेच पॉईंटर ज्या व्हेरिएबल ला पॉईंट करतो तिथे साठविलेली variable ची value ऍक्सेस करण्यासाठी * ऑपरेटर चा वापर केला जातो.



आकृती ५.१.३ पॉइंटर व्हेरिएबल

वरील आकृतीमध्ये दर्शविल्याप्रमाणे number व्हेरिएबल आहे जो मेमरी ऍड्रेस fff4 वर साठविलेला आहे आणि पॉइंटर व्हेरिएबल p चे मूल्य fff4 आहे तसेच पॉईंटर p हा aaa3 या लोकेशन वर स्टोर केलेला आहे

उदाहरणार्थ:

```
#include <stdio.h>
int main() {
  int number = 50;
  int* p;
   p = &number;
  printf("number=%d\n", number);
  printf("Value of pointer p=%u",p );
  printf("value of *p=\n", *p);
 return 0;
}
```

Output:

number=50

Value of pointer p =fff4

Value of *p=10

५.२ पॉईंटर अरीथमेटिक ऑपरेशन्स (Pointer Arithmetic Operations):

पॉइंटर अंकगणित हा वैध अंकगणित ऑपरेशन्सचा संच आहे जो पॉइंटरवर करता येतो. पॉइंटर व्हेरिएबल्स दुसऱ्या व्हेरिएबलचा मेमरी पत्ता संग्रहित करतात. म्हणून, फक्त काही ऑपरेशन्स आहेत ज्यांना C भाषेत पॉइंटर्सवर करण्याची परवानगी आहे.

खालीलप्रमाणे पॉइंटर अरीथमेटिक ऑपरेशन्स आहे.

५.२.१ Increment(++)/Decrement(--) of a Pointer

जेव्हा पॉइंटर वाढविला /(increment) केला जातो, तेव्हा तो प्रत्यक्षात डेटा प्रकाराच्या आकाराच्या समान संख्येने वाढतो ज्यासाठी तो पॉइंटर आहे.

उदाहरणार्थ:

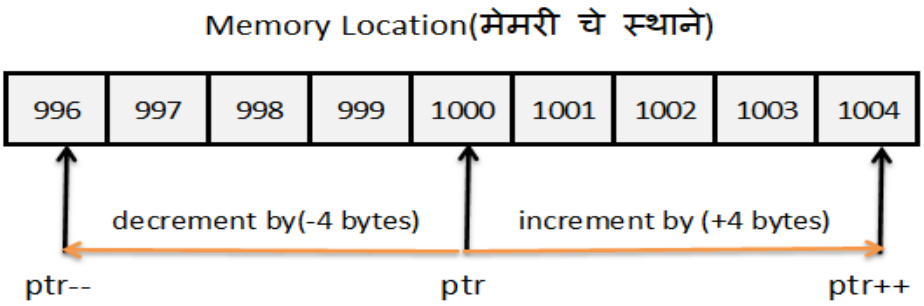
जर अड्रेस १००० साठवणारा integer पॉइंटर वाढवला असेल तर तो ४ ने वाढेल (int size), आणि नवीन अड्रेस १००४ ला पॉइंट करेल. जर double टाइप पॉइंटर वाढवला असेल तर तो ८ ने वाढेल (double) आणि नवीन अड्रेस १००८ ला पॉइंट करेल,. म्हणून पॉईंटर चा उपयोग अरे (array elements) ऍक्सेस करण्यासाठी महत्वाचा ठरतो .

५.२.१ Decrement of a Pointer(--)

जेव्हा पॉइंटर कमी केला जातो, तेव्हा तो प्रत्यक्षात डेटा प्रकाराच्या आकाराच्या समान संख्येने कमी होतो ज्यासाठी तो पॉइंटर आहे.

उदाहरणार्थ:

जर अड्रेस १००० साठवणारा integer पॉइंटर कमी केला असेल,तर तो ४ byte ने (int size) ने कमी होईल आणि नवीन पत्ता ९९८ वर निर्देशित करेल. जर double प्रकार पॉइंटर कमी केला असेल तर तो ८ byte (double size) ने कमी होईल. आणि नवीन पत्ता ९९२ असेल.



आकृती **५.२.१** increment/decrement operator

टीप: येथे असे गृहीत धरले आहे की आर्किटेक्चर ६४-बिट आहे आणि सर्व डेटा प्रकार त्यानुसार आकारले आहेत. उदाहरणार्थ, int type ४ बाइट्सचा आहे.

Program:

खालील प्रोग्राम ++ आणि -- ऑपरेटरचा पॉइंटर व्हेरिएबलसह वापर दर्शवतो

```
#include <stdio.h>
 int main() {
  int a;
  int *pt;
  printf("Pointer Example Program : Increment and Decrement Integer\n");
  a = 10;
  pt = &a;
  (*pt)++;                                    //Post Increment
  printf("\n[a]:Increment Value of A = %d", a);
  ++(*pt);                                    //Pre Increment
  printf("\n[a]:Increment Value of A = %d", a);
  (*pt)--;                                    //Post Decrement
  printf("\n[a]:Decrement Value of A = %d", a);
  --(*pt);                                    //Pre Decrement
  printf("\n[a]:Decrement Value of A = %d", a);
  return 0;
 }
```

Output

Pointer Example Program : Increment and Decrement Integer

[a]:Increment Value of A = 11

[a]:Increment Value of A = 12

[a]:Decrement Value of A = 11

[a]:Decrement Value of A = 10

खालील प्रोग्राम ++ आणि -- ऑपरेटरचा पॉइंटर char ,int ,float पॉईंटर टाईप साठी वापरला आहे.

```
#include <stdio.h>
int main(){
 int a = 22;
 int *p = &a;
 printf("p = %u\n", p); // p = 6422288
 p++;
 printf("p++ = %u\n", p); //p++ = 6422292          +4   // 4 bytes
 p--;
 printf("p-- = %u\n", p); //p-- = 6422288       -4   // restored to original value
 float b = 22.22;
 float *q = &b;
 printf("q = %u\n", q);  //q = 6422284
 q++;
 printf("q++ = %u\n", q); //q++ = 6422288        +4   // 4 bytes
 q--;
 printf("q-- = %u\n", q);   //q-- = 6422284    -4  // restored to original value
 char c = 'a';
 char *r = &c;
 printf("r = %u\n", r);                                  //r = 6422283
 r++;
 printf("r++ = %u\n", r);                          //r++ = 6422284     +1   // 1 byte
 r--;
 printf("r-- = %u\n", r);   //r-- = 6422283     -1  // restored to original value
 return 0;
```

```
}
```

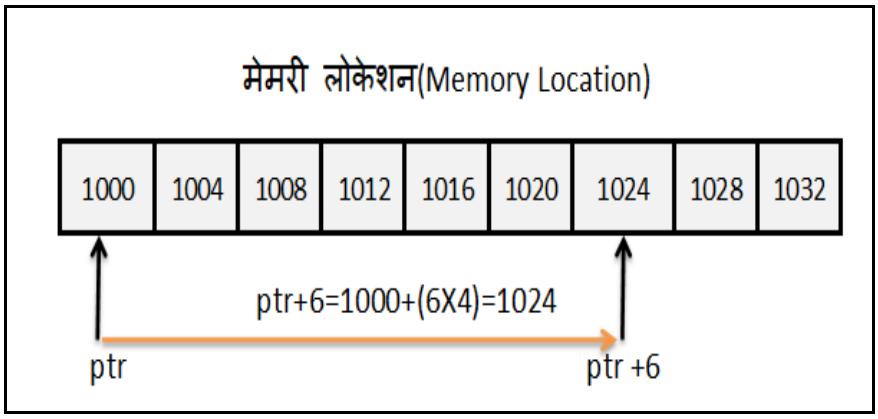
Output

```
p = 1441900792

p++ = 1441900796

p-- = 1441900792

q = 1441900796

q++ = 1441900800

q-- = 1441900796

r = 1441900791

r++ = 1441900792

r-- = 1441900791
```

टीप: पॉइंटर %p वापरून आउटपुट मिळते पण बहुतेकदा ते हेक्साडेसिमल फॉर्ममध्ये दर्शविले जाते . समजून घेण्यासाठी व अधिक सोपे करण्यासाठी unsigned int फॉर्ममध्ये मूल्य मिळविण्यासाठी %u चा वापर करता येतो .वरील प्रोग्रॅम मध्ये %u चा वापर केलेला असल्यामुळे आउटपुट समजणे येथे सोपे जाते

५.२.२ पॉइंटर मध्ये पूर्णांकाची बेरीज (Addition of integer to a pointer)

जेव्हा पॉइंटर मध्ये int नंबर add केला जातो, तेव्हा पॉईंटर (४ X नंबर) या मेमरी च्या लोकेशन ला पॉईंटर निर्देश (पॉईंट) करतो. येथे कोणती संख्या गुणाकार केली जाईल हे डेटा प्रकारावर अवलंबून असते. येथे कोणती संख्या गुणाकार केली जाईल हे डेटा प्रकारावर अवलंबून असते उदाहरणार्थ char १, int ४, float ४, double ८. पॉईंटर मध्ये अपूर्णांक संख्या ऍड (add) करता येत नाही कारण पॉईंटर केवळ मेमरी ऍड्रेस पूर्णांक संख्येत साठवितो.



आकृती ५.२.२ पॉइंटर मध्ये पूर्णांकाची बेरीज

```
#include <stdio.h>
 int main()
{
   // Integer variable
   int N = 4;
  int *ptr1, *ptr2;
   ptr1 = &N;                          // Pointer stores the address of N variable
   ptr2 = &N;
   printf("Pointer ptr2 before Addition: ");
   printf("%u \n", ptr2);
   ptr2 = ptr2 + 3;                          // Addition of 3 to ptr2
   printf("Pointer ptr2 after Addition: ");
   printf("%u \n", ptr2);
   return 0;
}
```

Output

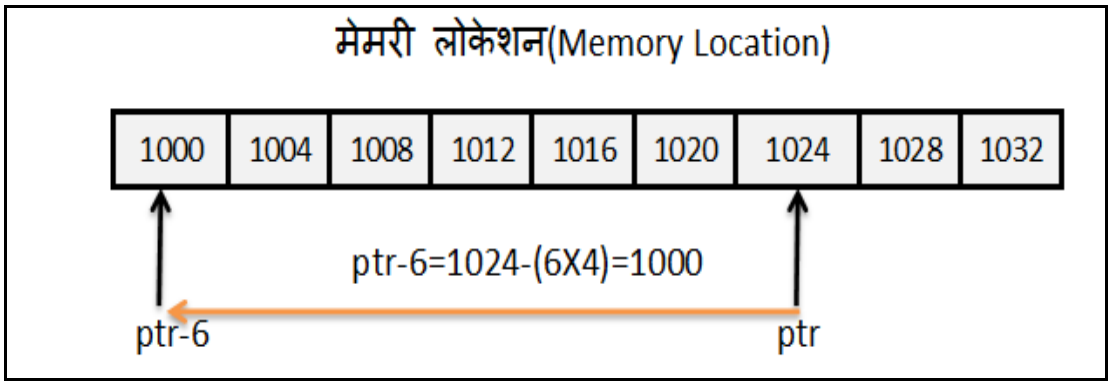
```
Pointer ptr2 before Addition: 6487564
Pointer ptr2 after Addition: 6487576
```

५.२.३ पॉइंटरची पूर्णांकातील वजाबाकी *(*Subtraction of integer to a pointer)

जेव्हा पॉइंटर मधून ठरावीक पूर्णांक वजा केला जातो , तेव्हा डेटा प्रकाराच्या साईज नुसार गुणाकार करून आलेली value पॉइंटरमधून वजा केले जाते .पॉईंटर वजाबाकी हे पॉईंटर मध्ये संख्या मिळविण्याच्या विरुद्ध आहे येथे पॉईंटर मागच्या लोकेशन ला सरकतो.

उदाहरणार्थ:

वरील प्रमाणेच उदाहरण विचारात घ्या जिथे ptr हा एक पूर्णांक पॉइंटर आहे जो 1024 ला पत्ता म्हणून संग्रहित करतो. जर आपण ptr = ptr – 6 या चा वापर करून पूर्णांक 5 वजा केला तर, ptr मध्ये संग्रहित केलेला अंतिम पत्ता ptr = 1000 – sizeof(int) * 6 = 1000 असेल.



आकृती ५.२.३ पॉइंटरची पूर्णांकातील वजाबाकी

५.२.४ एकाच प्रकारचे दोन पॉइंटर वजा करणे(Subtracting two pointers of the same type)

दोन पॉइंटर्सची वजाबाकी तेव्हाच शक्य आहे जेव्हा त्यांचा डेटा प्रकार समान असेल. दोन पॉइंटरच्या पत्त्यांमधील फरक मोजून आणि पॉइंटर डेटा प्रकारानुसार डेटाचे किती बिट आहेत याची गणना करून परिणाम तयार केला जातो. दोन पॉइंटर्सची वजाबाकी दोन पॉइंटरमधील अंतर देते.

समजा दोन पूर्णांक पॉइंटर ptr1(ऍड्रेस :1000) आणि ptr2 (ऍड्रेस :1004) वजा केले तर त्यांमधील फरक 4 बाइट्स आहे. int चा आकार 4 बाइट असल्याने, ptr1 आणि ptr2 मधील वाढ (4/4) = 1 ने दिली आहे.

एकाच प्रकारच्या दोन पॉइंटरची वजाबाकी दाखविणारा खालील प्रोग्राम द्वारे आपण ही संकल्पना समजून घेऊ.

```
#include <stdio.h>
int main()
{
   int x = 6;                          // Integer variable declaration
   int N = 4;
   int *ptr1, *ptr2;                   // Pointer declaration
```

```
    ptr1 = &N;                          // stores address of N
    ptr2 = &x;                          // stores address of x
     printf(" ptr1 = %u, ptr2 = %u\n", ptr1, ptr2);
     x = ptr1 - ptr2;        // Subtraction of ptr2 and ptr1
     printf("Subtraction of ptr1 "  "& ptr2 is %d\n",  x);        // printing x value
    return 0;
}
```

Output:

ptr1 = 6487560, ptr2 = 6487564

Subtraction of ptr1 & ptr2 is -1

५.२.५ कॅम्पेरिजन ऑफ पॉइंटर्स (Comparison of pointers)

C मध्ये, पॉइंटरसह कॉम्पेरिजन ऑपरेटर वापरले जाऊ शकतात , पॉइंटर्स ने स्टोर केलेली value तसेच ऍड्रेस यामध्ये तुलना करण्यासाठी !=,==,>,<,>=,<=,!<,!> हे ऑपरेटर वापरता येतात

५.२.५.१ Greater Than (>), Less Than (<), Greater Than or Equal To (>=), Less Than or Equal To (<=)

हे ऑपरेटर दोन पॉइंटर्सच्या मेमरी पत्त्यांची तुलना करण्यासाठी वापरले जाऊ शकतात. तथापि, हे ऑपरेटर वापरून पॉइंटरची तुलना करणे अर्थपूर्ण असू शकत नाही जोपर्यंत ते समान ॲरेमधील घटकांकडे किंवा सलग मेमरी स्थानांकडे निर्देश करत नाहीत. म्हणून पॉइंटर्स बरोबर कॉम्पॅरिजन ऑपरेटर ॲरे वगळता इतरत्र वापरात आणत नाहीत.

५.२.५.२ समानता(Equality) operator(==)

दोन पॉइंटर एकाच मेमरी पत्त्याकडे निर्देश करतात का ते तपासते.

```
    if (ptr1 == ptr2)
  {
      // Pointers are equal
   }
```

५.२.५.३ असमानता(Inequality) Operator (!=) :

दोन पॉइंटर एकाच मेमरी पत्त्याकडे निर्देश करत नाहीत का ते तपासते.

```
if (ptr1 != ptr2)
{
    // Pointers are not equal
 }
```

५.२.५.४ Greater than Operator (>)

```
if (ptr1 > ptr2)
{
   // ptr1 points to a higher memory address than ptr2
}
```

५.२.५. ५ Less than Operator(<)

```
if (ptr1 < ptr2)
{
   // ptr points to a lower  memory address than ptr2
}
```

५.२.५.६ NULL शी तुलना (Comparison to NULL)

ज्या पॉईंटर ला NULL value देऊन घोषित केले जाते अश्या पॉईंटर ला NULL पॉईंटर असे म्हणतात.

C पॉईंटर ला NULL value असाइन किंवा देता येते तसेच दिलेला पॉईंटर null आहे किंवा नाही हे हि कॅम्परिजन ऑपरेटर (==) वापरून तपासता येते.

खालील C प्रोग्रॅम मध्ये NULL पॉईंटर चेक केला आहे

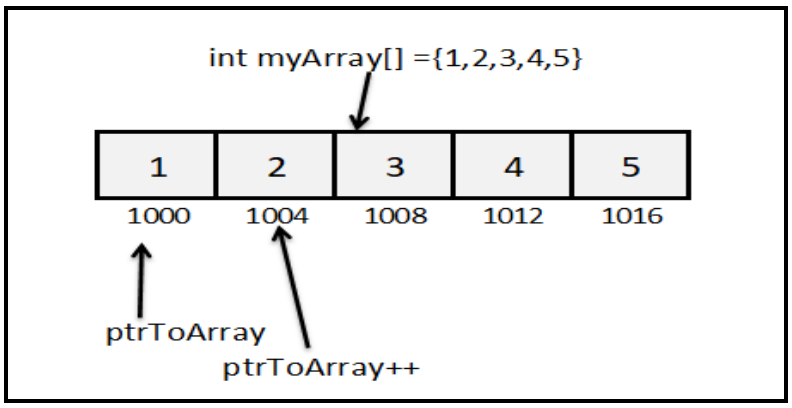
```
#include <stdio.h>
int main()
{
   int* ptr = NULL;
   if (ptr == NULL) {
      printf("The pointer is NULL");
   }
   else {
      printf("The pointer is not NULL");
   }
```

```
    return 0;
}
```

५.३ Pointer to array(अॅरे साठी पॉइंटर)

C मध्ये, अॅरेचा पॉइंटर अॅरेचे इलेमेंट्स (elements) हाताळण्यासाठी उपयुक्त ठरतो . जेव्हा तुम्ही अॅरे पॉइंटर घोषित करता, तेव्हा तुम्ही मूलत: एक व्हेरिएबल तयार करता जे अॅरेच्या पहिल्या घटकाचा मेमरी पत्ता संचयित करू शकते. तुम्ही अॅरेला पॉइंटर कसे घोषित आणि वापरू शकता ते खालील उदाहरणात स्पष्ट केले आहे

```
#include <stdio.h>
int main() {
    int myArray[] = {1, 2, 3, 4, 5};
    int *ptrToArray;
    // Assign the address of the first element of the array to the pointer
    ptrToArray = myArray;
    // Access elements using the pointer
    printf("First element: %d\n", *ptrToArray);  // Output: 1
    // Move the pointer to the next element
    ptrToArray++;
    printf("Second element: %d\n", *ptrToArray);  // Output: 2
    return 0;
}
Output:
First element: 1
Second element: 2
```



आकृती ५.३ अॅरे पॉइंटर

या उदाहरणात, ptrToArray हा पूर्णांक (int*) साठी एक पॉइंटर आहे आणि त्याला myArray अॅरेच्या पहिल्या घटकाचा ऍड्रेस संग्रहित केला आहे. हा पॉइंटर वाढवून किंवा कमी करून,अॅरेमधून नेव्हिगेट करून अॅरे इलेमेंट्स ऍक्सेस करता येतात. हे लक्षात घेणे महत्त्वाचे आहे की पॉइंटरचा प्रकार अॅरेमधील घटकांच्या प्रकाराशी जुळला पाहिजे. जर तुमच्याकडे पूर्णांकांची अॅरे असेल, तर पॉइंटर int* प्रकारचा असावा; जर तुमच्याकडे char अॅरे असेल, तर पॉइंटर char*, प्रकारचा असावा.

Example

Write a C program to find sum of array elements using Pointer to array.

(पॉइंटर टू अॅरे वापरून अॅरे घटकांची बेरीज शोधण्यासाठी C प्रोग्राम लिहा.)

Ans:-

```
#include <stdio.h>
int main() {
    int myArray[] = {1, 2, 3, 4, 5};
    int *ptrToArray = myArray;
    int arraySize = sizeof(myArray) / sizeof(myArray[0]);
    int sum = 0;
    // Use a loop to iterate through the array and add elements
    for (int i = 0; i < arraySize; i++) {
        sum += *ptrToArray; // Add the value pointed to by the pointer
        ptrToArray++;       // Move the pointer to the next element
    }
    printf("Sum of array elements: %d\n", sum);
```

```
    return 0;
}
```

Output

Sum of array elements: 15

५.४. पॉईंटर व टेक्स्ट स्ट्रिंग (Pointer and Text string)

C मध्ये, टेक्स्ट स्ट्रिंग सामान्यत: char अॅरे म्हणून स्टोर केले जाते . या स्ट्रिंग्सची कुशलतेने हाताळणी आणि प्रक्रिया करण्यासाठी पॉइंटर्सचा वापर सामान्यतः केला जातो. C मध्ये पॉइंटर आणि टेक्स्ट स्ट्रिंग एकत्र कसे कार्य करतात याविषयी चे स्पष्टीकरण खालीलप्रमाणे.

C मध्ये, डेटा प्रकार (char) निर्दिष्ट करून कॅरेक्टर अॅरे घोषित करतात , त्यानंतर अॅरेचे नाव आणि चौरस कंसात अॅरे ची size दयावी लागते . येथे काही उदाहरणे दिलेली आहेत:

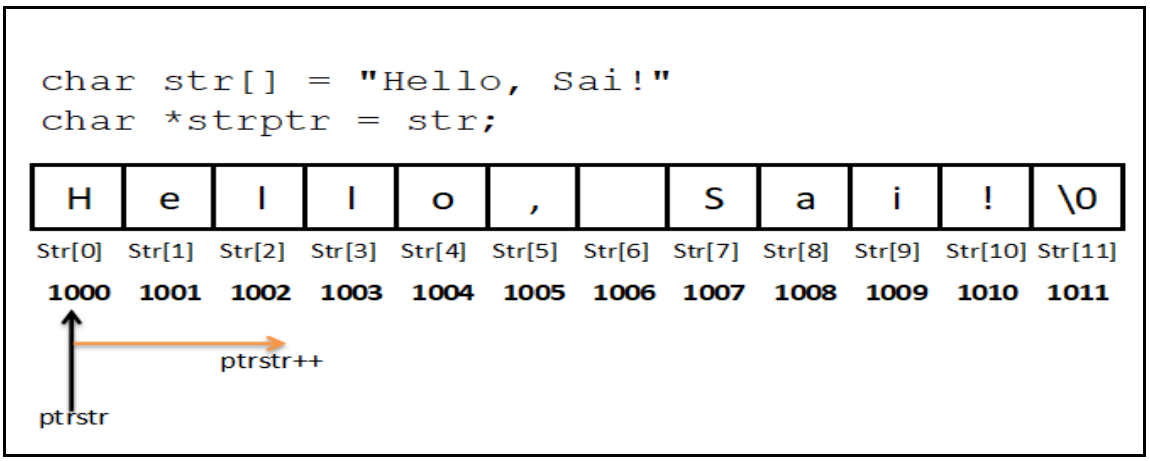
```
char myCharArray[ ] = "Hello, World!";
```

तसेच

```
char myCharArray[14] = "Hello, World!";
```

येथे दोन्हीही प्रकारचे अॅरे डिक्लेरेशन ला C मान्यता देते. ज्यावेळेस आपण size ला रिकामी ठेवत असू त्यावेळी त्या अॅरे ला डिक्लेरेशन वेळी इनीशिलाईज (initialize) करणे पण गरजेचे असते.

char पॉईंटर अॅरे च्या पहिल्या घटकाला निर्देश (point) करतो ,आणि त्यानंतर ++ ऑपरेटर च्या मार्फत इतर char अॅरे चे घटक ऍक्सेस करता येतात .



आकृती ५.४. पॉईंटर व टेक्स्ट स्ट्रिंग

```
#include <stdio.h>
int main()
 {
    char str[] = "Hello, Sai!";
    char *ptrstr = myString;            // पॉईंटर टु फर्स्ट स्ट्रिंग char
    printf("String using pointer: %s\n", ptrToString);
    while (*ptrToString != '\0')        //सर्व स्ट्रिंग इलेमेंट्स ऍक्सेस करणे
   {
        printf("%c ", *ptrToString);
        ptrToString++;
    }

    return 0;
}
```

Write a simple program in C to calculate the length of a string.

(स्ट्रिंगची लांबी मोजण्यासाठी C वापरून प्रोग्राम लिहा)

स्ट्रिंग ची लेंग्थ माहिती करून घेण्यासाठी जोपर्यंत null ('\0') value येत नाही तोपर्यंत आपल्याला स्ट्रिंग मध्ये असणारे सर्व char क्रमश: मोजावे लागतील. null हे स्ट्रिंग चा शेवट म्हणून संबोधले जाते .

```
#include <stdio.h>
int main() {
    char myString[] = "Hello, World!";
    int length = 0;                  // length variable ची सुरुवातीची value 0 ठेवा
    while (myString[length] != '\0')
    {
       length++;
    }
    printf("String: %s\n", myString);
```

```
    printf("Length of String: %d\n",length);

}
```

Output

```
String: Hello, World!
Length of String: 13
```

५.५ पॉइंटर्स द्वारे फंक्शन चा वापर करणे (Function handling using pointers)

जसे आपल्याला माहित आहे की आपण int, char, float अशा कोणत्याही डेटा प्रकाराचा पॉइंटर तयार करू शकतो, तसेच आपण फंक्शनकडे निर्देश (point) करणारा पॉइंटर देखील तयार करू शकतो.फंक्शनचा कोड नेहमी मेमरीमध्ये राहतो, याचा अर्थ फंक्शनचा काही ऍड्रेस असतो.फंक्शन पॉइंटर वापरून आपण मेमरीचा ऍड्रेस मिळवू शकतो.

उदाहरण.

```
#include <stdio.h>
int main()
{
   printf("Address of main() function is %p",main);
   return 0;
}
```

Output:

```
Address of main() function is 0000000000401530
```

वरील कोड main() फंक्शनचा ऍड्रेस प्रिंट करते.म्हणून आपण असा निष्कर्ष काढू की प्रत्येक फंक्शनला काही ऍड्रेस असतो.

५.५.१ फंक्शन पॉइंटरची घोषणा (Declaration of a function pointer)

C मध्ये, फंक्शनचा रिटर्न प्रकार,त्यानंतर पॉइंटरचे नाव आणि नंतर कंसात पॅरामीटर प्रकार निर्दिष्ट करून फंक्शन पॉइंटर घोषित करतात. सिंटॅक्स खालीलप्रमाणे

```
return_type (*pointer_name) (parameter_types);
```

येथे:

return_type: फंक्शनचा रिटर्न प्रकार(int ,char, float ,double) यापैकी एक असू शकेल ज्याकडे पॉइंटर निर्देशित(point) करेल.

(*pointer_name): (*) सूचित करते की तो एक पॉइंटर आहे आणि pointer_name हे फंक्शन पॉइंटरचे नाव आहे.

(parameter_types): फंक्शनचे पॅरामीटर प्रकार (int, char, float, double) यापैकी एक असू शकेल. ज्याकडे पॉइंटर निर्देशित (point) करेल

```
#include <stdio.h>
int add(int a, int b)        // add फंक्शन ची घोषणा
 {
   return a + b;
}

int main() {

   int (*funcPointer)(int, int);   //// फंक्शन पॉइंटरची घोषणा
    funcPointer = add;          // पॉइंटरला फंक्शनचा पत्ता नियुक्त करणे
   int result = funcPointer(3, 4);  //फंक्शन कॉल करण्यासाठी फंक्शन पॉइंटर
   printf("Result: %d\n", result);
   return 0;
}
```

int (*funcPointer)(int, int); funcPointer नावाचे फंक्शन पॉइंटर घोषित करते जे दोन पूर्णांकांना पॅरामीटर्स म्हणून घेऊन आणि पूर्णांक परत करणाऱ्या फंक्शनकडे पॉईंट करते

funcPointer = add ; फंक्शन पॉइंटरला add फंक्शनचा ऍड्रेस पुरविते.

int result = funcPointer(3, 4); 3 आणि 4 पॅरामीटर्स सह add फंक्शन कॉल करण्यासाठी फंक्शन पॉइंटर वापरला जातो.

अशाप्रकारे आउटपुट मध्ये आपल्याला 7 हि value प्रिन्ट होते.

Passing Pointers to functions:

C मध्ये,फंक्शन्समध्ये पॉइंटर पॅरामीटर्स म्हणून पास करता येतात त्याला कॉल बाय रेफेरेन्स (call by reference) असेही म्हणतात,ज्यामुळे पॉइंटर्स पॉईंट करत असलेल्या मेमरी लोकेशन्सवरील व्हॅल्यूवर क्रिया करता येतात.खालील प्रोग्राम दोन पूर्णांकांच्या मूल्यांची अदलाबदल करण्यासाठी पॉइंटर्स चा वापर करून फंक्शन ला पॅरामीटर्स कसे पास करायचे हे स्पष्ठ करतो.

Program:

```
#include <stdio.h>
void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}
int main() {
    int num1 = 5, num2 = 10;
    printf("Before swapping: num1 = %d, num2 = %d\n", num1, num2);
    swap(&num1, &num2);
    printf("After swapping: num1 = %d, num2 = %d\n", num1, num2);
    return 0;
}
```

आउटपुट

Before swapping: num1 = 5, num2 = 10

After swapping: num1 = 10, num2 = 5

या प्रोग्रॅम मध्ये :

swap function पॅरामीटर्स म्हणून दोन पूर्णांक पॉइंटर्स (int *a ,int *b) घेते.फंक्शनच्या आत, ते a आणि b द्वारे निर्देशित केलेल्या मेमरी स्थानांवर मूल्ये बदलते.मुख्य फंक्शनमध्ये, num1 आणि num2 हे दोन पूर्णांक व्हेरिएबल्स घोषित केले जातात.

num1 व num2 चे ऍड्रेस & ऑपरेटर वापरून swap function ला पाठविले जातात.

swap function num1 आणि num2 च्या मूल्यांची अदलाबदल करून, त्या मेमरी स्थानावरील मूल्यांमध्ये बदल करते.प्रोग्राम स्वॅपिंगच्या आधी आणि नंतर व्हॅल्यू प्रिंट करतो.

५.६ पॉईंटर टू स्ट्रक्चर (Pointers to structure)

C मध्ये, पॉईंटर चा वापर करून structure चे इलेमेंट्स ऍक्सेस करता येतात तसेच structure मधील डेटा मेंबर्स वर विविध गणितीय ऑपरेशन्स करून structure मधील डेटा कार्यक्षमतेने हाताळण्याचा पॉइंटर महत्वाची भूमिका पार पाडतो.

५.६.१ वाक्यरचना पॉईंटर टु स्ट्रक्चर ते घोषणा

(Syntax to declaration structure pointer):

```
struct structVarName  *strPtrName;
```

येथे structVarName हे स्ट्रक्चरचे नाव(name of structure) आहे आणि *strPtrName हे स्ट्रक्चर टू पॉइंटर व्हेरिएबलचे नाव आहे.

५.६.२.पॉइंटरला स्ट्रक्चर व्हेरिएबलचा ऍड्रेस देणे (Assigning Address to structure pointer)

Syntax

```
strPtrName=& structVarName
```

येथे strPtrName हे स्ट्रक्चर टू पॉइंटर व्हेरिएबलचे नाव आहे structVarName हे स्ट्रक्चर चे नाव(name of structure) आहे

५.६. ३ पॉईंटर द्वारे स्ट्रक्चर डेटा मेंबर्स ऍक्सेस करणे (Accessing Structure data members using Pointer)

स्ट्रक्चर डेटा मेंबर्स ला पॉईंटर द्वारे ऍक्सेस करण्यासाठी -> ऑपरेटर चा वापर केला जातो.

Syntax:

strPtrName->datamembername=value;

येथे strPtrName हे स्ट्रक्चर टू पॉइंटर व्हेरिएबलचे नाव आहे. Datamembername हे स्ट्रक्चर च्या डेटा मेंबर्स चे नाव आहे.

खालील प्रोग्रॅमच्या मदतीने आपण पॉईंटर टु स्ट्रक्चर चा वापर कसा केला जातो ते समजून घेऊ .

```
struct Point {
    int x;
    int y;
};
int main()
{
    struct Point myPoint = {3, 7};                  // Declare a structure variable
    struct Point *ptrToMyPoint;                  // Declare a pointer to a structure
    // Assign the address of the structure variable to the pointer
    ptrToMyPoint = &myPoint;
    printf("Original Point: x = %d, y = %d\n", myPoint.x, myPoint.y);
    ptrToMyPoint->x = 10;            //Using the pointer to modify structure members
    ptrToMyPoint->y = 5;
    // Accessing structure members using the pointer
    printf("Modified Point: x = %d, y = %d\n", ptrToMyPoint->x, ptrToMyPoint->y);
    return 0;
}
```

Output

```
Original Point: x = 3, y = 7
Modified Point: x = 10, y = 5
```

संदर्भ

१. https://www.javatpoint.com/function-pointer-as-argument-in-c

२. https://www.geeksforgeeks.org/c-pointers/?ref=lbp

३. https://www.javatpoint.com/c-pointer-to-pointer

४. Let us C by Yashwant Kanetkar

५. https://www.studytonight.com/c/pointer-with-function-in-c.php

६. https://www.w3schools.com/c/c_pointers.php