7

പ്രധാന ആശയങ്ങൾ

- തീരുമാനം എടുക്കുന്നതിനുള്ള പ്രസ്താവനകൾ
  - `if` പ്രസ്താവന
  - `if .. else` പ്രസ്താവന
  - നെസ്റ്റഡ് `if`
  - `else if` ലാഡർ
  - `switch` പ്രസ്താവന
  - കൺഡീഷണൽ ഓപ്പറേറ്റർ
- ആവർത്തന പ്രസ്താവനകൾ
  - `while` പ്രസ്താവന
  - `for` പ്രസ്താവന
  - `do .. while` പ്രസ്താവന
  - ലൂപ്പുകളുടെ നെസ്റ്റിംഗ്
- ജമ്പ് പ്രസ്താവന
  - `go to`
  - `breack`
  - `continue`

H2D7S6

നിയന്ത്രണ പ്രസ്താവനകൾ

ഇൻപുട്ട്, ഔട്ട്പുട്ട്, വില നൽകൽ എന്നിവ ചെയ്യുന്നതതിനുള്ള C++-ലെ നിർവഹണ പ്രസ്താവനകൾ കഴിഞ്ഞ അധ്യായങ്ങളിൽ നാം ചർച്ച ചെയ്തു. ഇതുപയോഗിച്ച് ലളിതമായ പ്രോഗ്രാമുകൾ എങ്ങനെ എഴുതുവാൻ കഴിയുമെന്ന് നമുക്കറിയാം. ഈ പ്രോഗ്രാമുകളുടെ നിർവഹണം അനുക്രമമാണ്. അതായത്, പ്രോഗ്രാമിലെ ഓരോ നിർദ്ദേശവും ഒന്നിന് പുറകെ ഒന്നായി പ്രവർത്തിക്കുന്നു. ഈ അധ്യായത്തിൽ C++-ലെ പ്രോഗ്രാമിന്റെ തനത് പ്രവർത്തനക്രമത്തിന് മാറ്റം വരുത്തുന്ന പ്രസ്താവനകളെ കുറിച്ചാണ് നാം ചർച്ച ചെയ്യുന്നത്. അധ്യായം 3 ൽ നാം ചർച്ച ചെയ്ത തെരഞ്ഞെടുക്കൽ, ആവർത്തിക്കൽ, നീക്കംചെയ്യൽ എന്നീ പ്രസ്താവനകൾ പ്രശ്നങ്ങൾ നിർദ്ധാരണം ചെയ്യാൻ ആവശ്യമായേക്കാം. സാധാരണയായി ഇത്തരം തീരുമാനങ്ങൾ കൈക്കൊള്ളുന്നത് ചില നിബന്ധനകളെ അടിസ്ഥാനമാക്കിയാണ്. C++ ഈ ആവശ്യം നിറവേറ്റുന്നത് നിയന്ത്രണ പ്രസ്താവനകളുടെ സഹായത്തോടെയാണ്. ഈ പ്രസ്താവനകൾ പ്രോഗ്രാം നിർവഹണത്തിലെ സാധാരണ രീതിക്ക് മാറ്റം വരുത്തുന്നതിനായി ഉപയോഗിക്കുന്നു. നിയന്ത്രണ പ്രസ്താവനകളെ രണ്ടായി തരംതിരിക്കാം. (1) തീരുമാനമെടുക്കൽ/തിരഞ്ഞെടുക്കൽ (Decision making/Selection statement) (2) ആവർത്തന പ്രസ്താവനകൾ (Iteration statement). ഈ പ്രസ്താവനകളും ഇവയുടെ വാക്യഘടനയും നിർവഹണ രീതികളും നമുക്ക് ചർച്ച ചെയ്യാം.

7.1 തീരുമാനങ്ങൾ എടുക്കുന്നതിനുള്ള പ്രസ്താവനകൾ (Decision making statements)

പ്രശ്നങ്ങൾ നിർദ്ധാരണം ചെയ്യുമ്പോൾ കമ്പ്യൂട്ടറുകളിൽ എല്ലാ പ്രസ്താവനകളും എല്ലാ സന്ദർഭങ്ങളിലും ഒരുപോലെ പ്രവർത്തിക്കണമെന്നില്ല. ചില പ്രസ്താവനകൾ ഒരു സന്ദർഭത്തിൽ പ്രവർത്തിക്കുമെങ്കിലും മറ്റു ചില സന്ദർഭങ്ങളിൽ പ്രവർത്തിക്കണമെന്നില്ല. ഇത്തരം സന്ദർഭ

ങ്ങളിൽ കമ്പ്യൂട്ടറിന് ആവശ്യമായ തീരുമാനങ്ങൾ എടുക്കേണ്ടതുണ്ട്. ഇതിനായി നാം ഇവിടെ അനുയോജ്യമായ നിബന്ധനകൾ നൽകുകയും അവയെ കമ്പ്യൂട്ടർ വിലയിരുത്തുകയും വേണം. ഈ ഫലത്തിന്റെ അടിസ്ഥാനത്തിൽ അത് ഒരു തീരുമാനം എടുക്കുന്നു. ഈ തീരുമാനങ്ങൾ ഒന്നുകിൽ ഒരു പ്രത്യേക പ്രസ്താവനയെ പ്രവർത്തിപ്പിക്കുന്നതിനായി തിരഞ്ഞെടുക്കുന്നതിനോ അല്ലെങ്കിൽ ചില പ്രസ്താവനകളെ പ്രവർത്തിപ്പിക്കുന്നതിൽ നിന്നും ഒഴിവാക്കുന്നതിനോ ആയിരിക്കും. ഇപ്രകാരം ചില പ്രസ്താവനകൾ മാത്രം നിർവഹണം നടത്തുന്നതിനായി C++ ൽ തീരുമാനമെടുക്കൽ പ്രസ്താവനകൾ അല്ലെങ്കിൽ തെരഞ്ഞെടുക്കൽ പ്രസ്താവനകൾ ഉപയോഗിക്കുന്നു. `if`, `switch` എന്നിവയാണ് C++ ലെ രണ്ടുതരം തിരഞ്ഞെടുക്കൽ പ്രസ്താവനകൾ.

7.1.1 if പ്രസ്താവന (if statement)

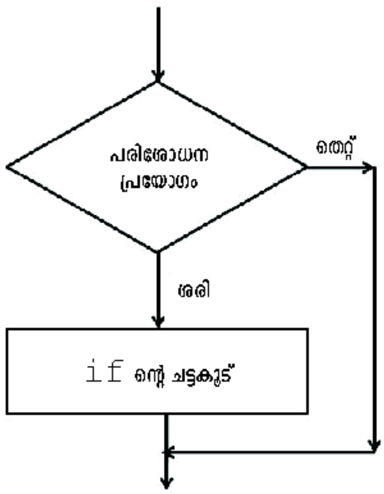
ഒരു നിബന്ധനയുടെ (condition) അടിസ്ഥാനത്തിൽ ഒരു കൂട്ടം പ്രസ്താവനകളെ പ്രവർത്തിപ്പിക്കുന്നതിനായി `if` പ്രസ്താവന ഉപയോഗിക്കുന്നു. C++ ൽ നിബന്ധനകൾ (പരിശോധനാ പ്രയോഗങ്ങൾ എന്നും അറിയപ്പെടുന്നു) നൽകുന്നത് റിലേഷണൽ അല്ലെങ്കിൽ ലോജിക്കൽ പ്രയോഗങ്ങൾ ഉപയോഗിച്ചാണ്. `if` പ്രസ്താവനയുടെ വാക്യഘടന (`Syntax`) താഴെ കൊടുത്തിരിക്കുന്നു.

```
if (പരിശോധന പ്രയോഗം)
{
    പ്രസ്താവനകൾ;
}
```

`if` ചട്ടക്കൂടിലെ നിബന്ധനകൾ ശരിയാവുകയാണെങ്കിൽ പ്രവർത്തിക്കേണ്ട പ്രസ്താവന കൂട്ടം;

```
if (test expression)
{
    statement block;
}
```

ഇവിടെ പരിശോധനാ പ്രയോഗം എന്നത് ഒന്നുകിൽ റിലേഷണൽ പ്രയോഗം അല്ലെങ്കിൽ ലോജിക്കൽ പ്രയോഗമായ ഒരു നിബന്ധനയെയാണ് സൂചിപ്പിക്കുന്നത്. പരിശോധനാ പ്രയോഗം ശരിയാണെങ്കിൽ (True-പൂജ്യം അല്ലാത്ത വില) `if` -നോടു ചേർന്നുള്ള പ്രസ്താവനയോ അല്ലെങ്കിൽ ഒരു കൂട്ടം പ്രസ്താവനകളോ പ്രവർത്തിക്കും. അല്ലെങ്കിൽ `if` -നു ശേഷമുള്ള പ്രസ്താവനയിലേക്ക് നിയന്ത്രണം കൈമാറുന്നു. ചിത്രം 7.1 `if` പ്രസ്താവനയുടെ പ്രവർത്തന രീതി കാണിക്കുന്നു. `if` ഉപയോഗിക്കുമ്പോൾ താഴെ പറയുന്ന ചില കാര്യങ്ങൾ ഓർത്തിരിക്കേണ്ടതുണ്ട്.



ചിത്രം 7.1: if പ്രസ്താവനയുടെ പ്രവർത്തനം

- പരിശോധനാ പ്രയോഗം എപ്പോഴും ആവരണ ചിഹ്നത്തിന് അകത്തായിരിക്കണം.
- നിബന്ധനയിലുള്ള പ്രയോഗം റിലേഷണൽ പ്രയോഗങ്ങൾ ഉപയോഗിച്ചുള്ള ലളിതമായ പ്രയോഗങ്ങളോ, ലോജിക്കൽ പ്രയോഗങ്ങൾ ഉപയോഗിച്ചുള്ള സംയുക്ത പ്രയോഗങ്ങളോ ആകാം.
- `if` പ്രസ്താവനയോടുകൂടി ഒന്നോ അതിലധികമോ

പ്രസ്താവനകൾ ഉണ്ടാകാം. ഒരു പ്രസ്താവന മാത്രമാണെങ്കിൽ {,} എന്നീ ബ്രാക്കറ്റുകൾ നിർബന്ധമില്ല. ഒന്നിൽ കൂടുതൽ പ്രസ്താവനകൾ ഉണ്ടെങ്കിൽ ഈ ബ്രാക്കറ്റുകൾ നിർബന്ധമാണ്.

പ്രോഗ്രാം 7.1 ഒരു വിദ്യാർഥിയുടെ സ്കോർ സ്വീകരിക്കുകയും അത് 18 ഓ അതിലധികമോ ആണെങ്കിൽ "You have Passed" എന്ന് പ്രദർശിപ്പിക്കുകയും ചെയ്യുന്നു. (പാസാവാൻ മിനിമം 18 സ്കോർ വേണമെന്ന് വിചാരിക്കുക)

പ്രോഗ്രാം 7.1: സ്കോർ 18 ഓ അതിലധികമോ ആണെങ്കിൽ "You have Passed" എന്ന പ്രദർശിപ്പിക്കുന്നതിന്

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

{
    int score ;
    cout << "Enter your score: ";
    cin >> score;
    if (score >= 18)
         cout << "You have passed";
    return 0;
}
```

if ന്റെ ചട്ടക്കൂട്

പ്രോഗ്രാം 7.1 - ന്റെ മാതൃകാ ഔട്ട്പുട്ട് താഴെകൊടുത്തിരിക്കുന്നു.

```
Enter your score: 25
You have passed
```

പ്രോഗ്രാം 7.1 -ൽ ഒരു വിദ്യാർത്ഥിയുടെ സ്കോർ നൽകുകയും അത് `score` എന്ന വേരിയബിളിൽ സംഭരിക്കുകയും ചെയ്യുന്നു. പരിശോധനാപ്രയോഗം സ്കോർ എന്ന വേരിയബിളിലെ വില 18-ഓ അതിൽ അധികമോ ആണോ എന്നു നോക്കുന്നു. പരിശോധനാപ്രയോഗം ശരിയാണെങ്കിൽ `if` -ന്റെ ഭാഗം പ്രവർത്തിക്കുന്നു. അതായത് സ്കോർ 18-ഓ അതിലധികമോ ആണെങ്കിൽ `"You have Passed"` എന്ന സന്ദേശം സ്ക്രീനിൽ പ്രദർശിപ്പിക്കപ്പെടുന്നു. അല്ലാത്തപക്ഷം ഒരു ഔട്ട്പുട്ടും ലഭിക്കുന്നില്ല.

`if`-നോടു കൂടിയുള്ള ഭാഗം ഒരു ടാബ് ദൂരത്തിന് ശേഷമാണ് എഴുതിയിട്ടുള്ളത് എന്നത് ശ്രദ്ധിക്കുക. നാം അതിനെ ഇൻഡന്റേഷൻ എന്നു വിളിക്കുന്നു. ഇത് പ്രോഗ്രാമിന്റെ വായന ക്ഷമത വർദ്ധിപ്പിക്കുകയും തെറ്റുകൾ കണ്ടുപിടിക്കാൻ സഹായിക്കുകയും ചെയ്യുന്നു. എന്നാൽ ഇവ പ്രോഗ്രാമിന്റെ പ്രവർത്തനത്തിൽ ഒരു സ്വാധീനവും ചെലുത്തുന്നില്ല.

താഴെ കൊടുത്തിരിക്കുന്ന C++ പ്രോഗ്രാം ശകലം ശ്രദ്ധിക്കുക. തന്നിരിക്കുന്ന ഇൻപുട്ട് ഒരു അക്ഷരമാണോ അല്ലെങ്കിൽ ഒരു അക്കമാണോ എന്ന് ഇത് പരിശോധിക്കുന്നു.

```
char ch;
cin >> ch;
if (ch >= 'a' && ch <= 'z')
```

ലോജിക്കൽ പ്രസ്താവന നിർദ്ധാരണം ചെയ്തിരിക്കുന്നു.

```
        cout << "You entered an alphabet";
    if (ch >= '0' && ch <= '9')
    {
        cout << "You entered a digit\n";
        cout << "It is a decimal number ";
    }
```

ഒരു പ്രസ്താവന മാത്രമേയുള്ളൂ അതുകൊണ്ട് { , } ആവശ്യമില്ല

ഒന്നിൽകൂടുതൽ പ്രസ്താവനകൾ ഉള്ളതു കൊണ്ട് അവ നിർബന്ധമായും {,} ന് അകത്തായിരിക്കണം

7.1.2 if... else പ്രസ്താവന (if... else statement)

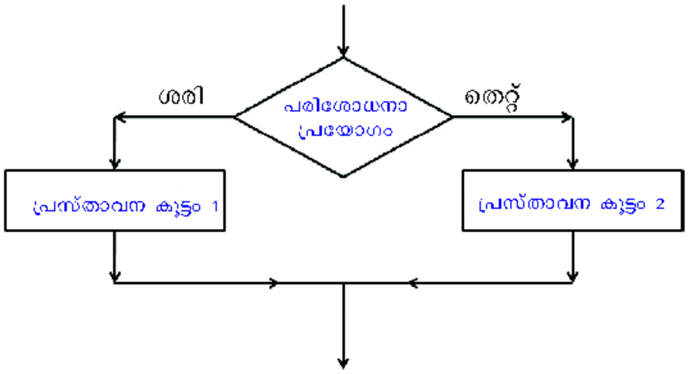
പ്രോഗ്രാം 7.1-ലെ if പ്രസ്താവന പരിഗണിക്കുക.

```
if (score >= 18)
    cout << "You have passed";
```

ഇവിടെ സ്കോർ പതിനെട്ടോ അതിൽ കൂടുതലോ ആണെങ്കിൽ മാത്രമേ ഔട്ട്പുട്ട് ലഭിക്കുന്നുള്ളൂ. നൽകിയ സ്കോർ 18-ൽ കുറവാണെങ്കിൽ എന്ത് സംഭവിക്കും? ഒരു ഔട്ട്പുട്ടും ലഭിക്കില്ല എന്ന് വ്യക്തമാണ്. പരിശോധന പ്രയോഗം വിലയിരുത്തുമ്പോൾ തെറ്റ് (false) ലഭിക്കുകയാണെങ്കിൽ മറ്റൊരു കൂട്ടം പ്രസ്താവനകൾ തിരഞ്ഞെടുക്കുന്നതിനുള്ള അവസരം നമുക്ക് ലഭിക്കാതെ വരുന്നു. നിബന്ധന തെറ്റാവുന്ന അവസരത്തിൽ ചില പ്രവർത്തനങ്ങൾ ചെയ്യണമെങ്കിൽ if പ്രസ്താവനയുടെ മറ്റൊരു രൂപമായ **if.. else** നമുക്ക് ഉപയോഗിക്കാം. ഇതിന്റെ വാക്യഘടന താഴെ കൊടുക്കുന്നു.

```
    if    (പരിശോധനാ പ്രയോഗം)
    {
          പ്രസ്താവനകൾ 1 ;
    }
    else
    {
          പ്രസ്താവനകൾ 2;
    }
    if    (test expression)
    {
          statement block 1;
    }
    else
    {
          statement
block 2;
    }
```

പരിശോധനാപ്രയോഗം ശരിയാണെങ്കിൽ പ്രസ്താവനകൾ 1 ഉം തെറ്റാണെങ്കിൽ പ്രസ്താവനകൾ 2 ഉം പ്രവർത്തിക്കുന്നു. if...else പ്രസ്താവനയുടെ പ്രവർത്തനം ചിത്രം 7.2-ൽ കാണിച്ചിരിക്കുന്നു.



ചിത്രം 7.2 if....else പ്രസ്താവനയുടെ ഫ്ളോചാർട്ട്

താഴെ കൊടുത്തിരിക്കുന്ന കോഡ് ശകലം if...else പ്രസ്താവനയുടെ പ്രവർത്തനം വിവരിക്കുന്നു.

```
if (score >= 18)
    cout << "Passed";
else
    cout << "Failed";
```

18 ഓ അതിലധികമോ ആണെങ്കിൽ മാത്രം ഈ പ്രസ്താവന പ്രവർത്തിക്കുന്നു. (അതായത് പരിശോധനാ പ്രയോഗം ശരിയാകുമ്പോൾ)

18 ൽ കുറവാണെങ്കിൽ ഈ പ്രസ്താവന പ്രവർത്തിക്കുന്നു. (അതായത് പരിശോധനാ പ്രയോഗം തെറ്റാകുമ്പോൾ.)

രണ്ട് കുട്ടികളുടെ ഉയരം ഇൻപുട്ടായി സ്വീകരിച്ച് അവരിൽ ഉയരമുള്ള കുട്ടിയെ കണ്ടുപിടിക്കുന്നതിനുള്ള ഒരു പ്രോഗ്രാം നമുക്കെഴുതാം.

പ്രോഗ്രാം 7.2: വിദ്യാർത്ഥികളുടെ ഉയരം താരതമ്യം ചെയ്ത് അവരിൽ ഉയരം കൂടുതലുള്ള ആളെ കണ്ടുപിടിക്കുന്നതിന്.

```
#include <iostream>
using namespace std;
int main()
{
    int ht1, ht2;
    cout << "Enter heights of the two students: ";
    cin >> ht1 >> ht2;
    if (ht1 > ht2)    //decision making based on condition
        cout<<"Student with height "<<ht1<<" is taller";
    else
        cout<<"Student with height "<<ht2<<" is taller";
    return 0;
}
```

പ്രോഗ്രാം 7.2 പ്രവർത്തിക്കുമ്പോൾ ht1>ht2 എന്ന റിലേഷണൽ പ്രയോഗത്തിന്റെ ഫലത്തെ ആശ്രയിച്ച് ഏതെങ്കിലും ഒരു ഔട്ട്പുട്ട് പ്രദർശിപ്പിക്കപ്പെടും. മാതൃകാ ഔട്ട്പുട്ടുകൾ താഴെ കൊടുത്തിരിക്കുന്നു.

ഔട്ട്പുട്ട് 1:
```
Enter the height of two students: 170   165
Student with height 170 is taller
```

ഔട്ട്പുട്ട് 2:
```
Enter the height of two students: 160   171
Student with height 171 is taller
```

ഔട്ട്പുട്ട് 1-ൽ ht1 -ന് 170-ഉം ht2 -ന് 165-ഉം ഇൻപുട്ടായി നൽകിയിരിക്കുന്നു. അതുകൊണ്ട് (ht1>ht2) എന്ന പരിശോധനാപ്രയോഗം ശരിയാവുകയും തൽഫലമായി if ബ്ലോക്ക് പ്രവർത്തിക്കുകയും ചെയ്യുന്നു. ഔട്ട്പുട്ട് 2-ൽ ht1 -ന് 160-ഉം ht2-ന് 171-ഉം വിലകൾ നൽകുമ്പോൾ ht1>ht2 എന്ന പരിശോധനാ പ്രയോഗം തെറ്റാവുകയും തൽഫലമായി else ബ്ലോക്ക് പ്രവർത്തിക്കുകയും ചെയ്യുന്നു. if .. else പ്രസ്താവനയിൽ ഒന്നുകിൽ if നോട് അനുബന്ധിച്ചുള്ള കോഡും

(പ്രസ്താവനകൾ 1) അല്ലെങ്കിൽ `else` നോട് അനുബന്ധിച്ചുള്ള കോഡും (പ്രസ്താവനകൾ 2) പ്രവർത്തിക്കുന്നു.

പരിശോധനാ പ്രയോഗത്തിൽ ഒരു ഓപറൻഡ് ആയി അരിത്തമെറ്റിക് പ്രയോഗം ഉപയോഗിച്ചിരിക്കുന്ന മറ്റൊരു പ്രോഗ്രാം നമുക്ക് നോക്കാം. പ്രോഗ്രാം 7.3 ഈ ആശയം ഉപയോഗിച്ച് ഒരു സംഖ്യ ഒറ്റസംഖ്യയാണോ ഇരട്ടസംഖ്യയാണോ എന്ന് പരിശോധിക്കുന്നു.

പ്രോഗ്രാം 7.3: തന്നിരിക്കുന്ന സംഖ്യ ഒറ്റസംഖ്യയാണോ ഇരട്ടസംഖ്യയാണോ എന്ന് പരിശോധിക്കുന്നതിന്.

```
#include <iostream>
using namespace std;
int main()
{
    int num;
    cout << "Enter the number: ";
    cin >> num;
    if (num%2 == 0)
        cout << "The given number is Even";
    else
        cout << "The given number is Odd";
    return 0;
}
```

പ്രോഗ്രാം 7.3 ന്റെ ചില മാതൃക ഔട്ട്പുട്ടുകൾ താഴെ കാണിച്ചിരിക്കുന്നു

ഔട്ട്പുട്ട് 1:

```
Enter the number: 7
The given number is Odd
```

ഔട്ട്പുട്ട് 2:

```
Enter the number: 10
The given number is Even
```

ഈ പ്രോഗ്രാമിൽ `(num%2)` എന്ന പ്രയോഗം `num` ലെ വിലയെ 2 കൊണ്ട് ഹരിച്ച് കിട്ടുന്ന ശിഷ്ടത്തെ പൂജ്യമായി താരതമ്യം ചെയ്യുന്നു. അത് തുല്യമാണെങ്കിൽ `if` പ്രസ്താവനകൾ പ്രവർത്തിക്കും അല്ലെങ്കിൽ `else` പ്രസ്താവനകൾ പ്രവർത്തിക്കും.

നമുക്ക് ചെയ്യാം

1. തന്നിരിക്കുന്ന സംഖ്യ പോസിറ്റീവ് ആണോ നെഗറ്റീവ് ആണോ എന്ന് പരിശോധിക്കാനുള്ള പ്രോഗ്രാം എഴുതുക. (പൂജ്യം ഒഴികെയുള്ള സംഖ്യ മാത്രമേ ഇൻപുട്ട് ചെയ്യുന്നുള്ളൂ.)
2. ഒരു അക്ഷരം സ്വീകരിച്ച് ‘M’ ആണെങ്കിൽ Male എന്നും ‘F’ ആണെങ്കിൽ Female എന്നും ഔട്ട്പുട്ട് കാണിക്കുന്നതിനുള്ള പ്രോഗ്രാം എഴുതുക.
3. നിങ്ങളുടെ പ്രായം ഇൻപുട്ടായി നൽകി അത് 18 വയസ്സിനു മുകളിലാണെങ്കിൽ വോട്ടു ചെയ്യാൻ യോഗ്യതയുണ്ടെന്നും അല്ലെങ്കിൽ യോഗ്യതയില്ലെന്നും പ്രദർശിപ്പിക്കാനുള്ള പ്രോഗ്രാം എഴുതുക.

7.1.3 നെസ്റ്റഡ് if (Nested if)

ചില സാഹചര്യങ്ങളിൽ `if` പ്രസ്താവനയുടെ അകത്തുനിന്നുകൊണ്ട് തീരുമാനങ്ങൾ എടുക്കേണ്ട ആവശ്യം വരും. ഒരു `if` ബ്ലോക്കിനകത്ത് മറ്റൊരു `if` ബ്ലോക്ക് എഴുതുന്നതിനെ നെസ്റ്റിങ്ങ് എന്നു പറയുന്നു. നെസ്റ്റഡ് എന്നാൽ ഒന്നിനകത്ത് മറ്റൊന്ന് എന്നാണ് അർഥം. താഴെ കൊടുത്തിരിക്കുന്ന പ്രോഗ്രാം ശകലം പരിഗണിക്കുക.

```
if (score >= 60)
{
    if(age >= 18)
        cout<<"You are selected for the course!";
}
```

(പുറമെയുള്ള if → `if (score >= 60)`; അകത്തുള്ള if → `if(age >= 18)`)

ഈ കോഡ് ശകലത്തിൽ Scoreന്റെ വില 60 ഓ അതിൽ കൂടുതലോ ആണെങ്കിൽ പ്രോഗ്രാമിന്റെ നിയന്ത്രണം പുറത്തെ **`if`** ബ്ലോക്കിനുള്ളിലേക്ക് പ്രവേശിക്കുന്നു. അതിനുശേഷം അകത്തുള്ള **`if`** ന്റെ പരിശോധനാ പ്രസ്താവന വിലയിരുത്തുന്നു. (അതായത് age ന്റെ വില പതിനെട്ടോ അതിൽ കൂടുതലോ എന്ന്). ഇത് ശരിയാണെങ്കിൽ "You are selected for the course!" എന്ന സന്ദേശം പ്രദർശിപ്പിക്കും. തുടർന്ന് പുറത്തെ **`if`** പ്രസ്താവനകൾക്ക് ശേഷമുള്ള പ്രസ്താവനകൾ പ്രവർത്തിക്കുന്നു.

ഒരു `if` പ്രസ്താവനക്കുള്ളിലെ മറ്റൊരു `if` പ്രസ്താവനയെ നെസ്റ്റഡ് `if` (nested **`if`**) എന്നു വിളിക്കുന്നു. നെസ്റ്റഡ് `if`-ന്റെ വിപുലീകരിച്ച വാക്യഘടന താഴെ കൊടുത്തിരിക്കുന്നു.

```
if  (test expression1)
{
    if (test expression 2)
        statement 1;
    else
        statement 2;
}
else
{
    body of else;
}
```

- statement 1: പരിശോധനാ പ്രയോഗം രണ്ടും ശരിയാകുമ്പോൾ മാത്രം പ്രവർത്തിക്കുന്നു.
- statement 2: പരിശോധനാ പ്രയോഗം 1 ശരിയാവുകയും പരിശോധനാ പ്രയോഗം 2 തെറ്റാവുകയും ചെയ്യുമ്പോൾ പ്രവർത്തിക്കുന്നു.
- body of else: പരിശോധനാ പ്രയോഗം 1 തെറ്റാകുമ്പോൾ പ്രവർത്തിക്കും. പരിശോധനാ പ്രയോഗം 2 നിർദ്ധാരണം ചെയ്യുന്നില്ല.

നെസ്റ്റഡ് `if` മായി ബന്ധപ്പെട്ട് ശ്രദ്ധിക്കേണ്ട പ്രധാന കാര്യം ഒരു `else` എപ്പോഴും അതേ ബ്ലോക്കിൽ തന്നെയുള്ള ഏറ്റവും അടുത്ത if മായി ബന്ധപ്പെട്ടിരിക്കുന്നു എന്നതാണ്. ഒരു ഉദാഹരണത്തിലൂടെ നമുക്ക് ഇത് ചർച്ച ചെയ്യാം. താഴെ പറയുന്ന പ്രോഗ്രാം ശകലം പരിഗണിക്കുക.

```
cout<<"Enter your score in Computer Science exam: ";
 cin>>score;
 if (score >= 18)
      cout<<"You have passed";
      if(score >= 54)
           cout<<"with A+ grade !";
 else
      cout<<"\nYou have failed";
```

Score ന് 45 എന്ന വില നൽകുകയാണെങ്കിൽ ഔട്ട്പുട്ട് താഴെ ഉള്ളതുപോലെയായിരിക്കും.

```
You have passed
You have failed
```

യുക്തിപരമായി ഇത് ശരിയല്ല എന്ന് നമുക്ക് അറിയാം. കോഡിന്റെ ഇൻഡന്റേഷൻ ശരിയാണെങ്കിലും പ്രവർത്തനത്തിൽ പ്രതിഫലിച്ചിട്ടില്ല. ഇതിൽ രണ്ടാമത്തെ `if` പ്രസ്താവന നെസ്റ്റഡ് `if` ആയി പരിഗണിച്ചിട്ടില്ല. പകരം അതിനെ `else` ബ്ലോക്കോടു കൂടിയ സ്വതന്ത്രമായ ഒരു `if` ആയിട്ടാണ് കണക്കാക്കിയിട്ടുള്ളത്. അതുകൊണ്ട് ആദ്യത്തെ `if` പ്രസ്താവനയിലെ പരിശോധനാ പ്രയോഗം ശരിയായതിനാൽ അതിലെ `if` ബ്ലോക്ക് പ്രവർത്തനത്തിനായി തിരഞ്ഞെടുക്കുന്നു. ഇത് ഔട്ട്പുട്ടിലെ ഒന്നാമത്തെ വരിക്ക് കാരണമാകുന്നു. അതിനുശേഷം രണ്ടാമത്തെ `if` പ്രസ്താവനയിലെ പരിശോധനാ പ്രയോഗം തെറ്റായതിനാൽ ഔട്ട്പുട്ടിലെ രണ്ടാമത്തെ വരി ലഭിക്കുന്നു. അതുകൊണ്ട് ശരിയായ ഔട്ട്പുട്ട് ലഭിക്കുന്നതിനായി കോഡ് താഴെയുള്ളതുപോലെ പരിഷ്കരിക്കണം.

```
cout<<"Enter your score in Computer Science exam: ";
cin>>score;
if (score >= 18)
{
     cout<<"You have passed";
     if(score >= 54)
          cout<<" with A+ grade !";
}
else
     cout<<"\nYou have failed";
```

ഒരു ജോഡി ബ്രാക്കറ്റുകളുടെ സഹായത്തോടെ നെസ്റ്റിങ്ങ് നടപ്പിലാക്കിയിരിക്കുന്നു.

else പ്രസ്താവന ഇപ്പോൾ പുറത്തെ if നോട് ചേർന്നിരിക്കുന്നു.

മുൻ ഉദാഹരണത്തിലെ ഇൻപുട്ട് ആയ 45 ഇവിടെയും നൽകിയാൽ താഴെ പറയുന്ന ഔട്ട്പുട്ട് ലഭിക്കും.

```
You have passed
```

തന്നിരിക്കുന്ന മൂന്നു സംഖ്യകളിൽ വലുത് കണ്ടെത്തുന്നതിനായി പ്രോഗ്രാം 7.4 ൽ നെസ്റ്റഡ് `if` ഉപയോഗിച്ചിരിക്കുന്നു. ഈ പ്രോഗ്രാമിൽ `if` പ്രസ്താവന `if` ബ്ലോക്കിനകത്തും `else` ബ്ലോക്കിനകത്തും ഉപയോഗിച്ചിരിക്കുന്നു.

പ്രോഗ്രാം 7.4: മൂന്ന് സംഖ്യകളിൽ നിന്നും വലുത് കണ്ടുപിടിക്കുന്നതിന്

```
#include <iostream>
using namespace std;
int main()
{
    int x, y, z;
    cout << "Enter three different numbers: ";
    cin >> x >> y >> z ;
    if (x > y)
    {
        if (x > z)
            cout << "The largest number is: " << x;
        else
            cout << "The largest number is: " << z;
    }
    else
    {
        if (y > z)
            cout << "The largest number is: " << y;
        else
            cout << "The largest number is: " << z;
    }
    return 0;
}
```

പ്രോഗ്രാം 7.4-ന്റെ ഒരു മാതൃകാ ഔട്ട്പുട്ട് താഴെ കൊടുത്തിരിക്കുന്നു.

```
Enter three different numbers: 6   2   7
The largest number is: 7
```

ഇൻപുട്ട് നൽകിയ പ്രകാരം പുറമെയുള്ള `if` ലെ പരിശോധനാപ്രയോഗം `(x>y)` ശരിയായതിനാൽ അതിലെ അകത്തെ `if` ലേക്ക് പ്രവേശിക്കുന്നു. ഇവിടെ `(x>z)` എന്ന പരിശോധനാപ്രയോഗം തെറ്റായതിനാൽ അതിന്റെ `else` ബ്ലോക്ക് പ്രവർത്തിക്കുന്നു. അതുകൊണ്ട് z-ന്റെ വില ഔട്ട്പുട്ടായി പ്രദർശിപ്പിക്കുന്നു.

സ്വയം പരിശോധിക്കാം

1. ഒരു പൂർണ്ണ സംഖ്യ ഇൻപുട്ടായി സ്വീകരിച്ച്, അത് പോസറ്റീവാണോ നെഗറ്റീവാണോ പൂജ്യം ആണോ എന്ന് പരിശോധിക്കുന്നതിനുള്ള ഒരു പ്രോഗാം എഴുതുക
2. മൂന്ന് സംഖ്യകളെ ഇൻപുട്ടായി സ്വീകരിച്ച് അതിലെ ചെറുത് പ്രിന്റ് ചെയ്യാനുള്ള ഒരു പ്രോഗാം എഴുതുക.

7.1.4 else if ലാഡർ (The else if ladder)

ഒരു else ബ്ലോക്കിനുള്ളിൽ ഒരു if പ്രസ്താവന ഉപയോഗിക്കേണ്ട സാഹചര്യം ഉണ്ടായേക്കാം. അനേകം നിബന്ധനകൾ (condition) ആവശ്യമുള്ള പ്രോഗ്രാമുകളിൽ അത് ഉപയോഗിക്കുന്നു. പ്രവർത്തനത്തിനായി ഏത് പ്രസ്താവന തിരഞ്ഞെടുക്കണമെന്ന് അതാത് നിബന്ധന തീരുമാനിക്കുന്നു. if പ്രസ്താവനയെ അടിസ്ഥാനമാക്കിയുള്ള ഒരു സാധാരണ പ്രോഗ്രാമിംഗ് രൂപകൽപനയാണ് else if ലാഡർ. അതിന്റെ രൂപഘടനയിലുള്ള പ്രത്യേകത കാരണം അതിനെ else if സ്റ്റെയർ കെയ്സ് എന്നും പറയുന്നു. ഇത് if..else if പ്രസ്താവന എന്നും അറിയപ്പെടുന്നു. else if ലാഡറിന്റെ വാക്യഘടന താഴെ കൊടുക്കുന്നു.

```
if (പരിശോധനാ പ്രയോഗം 1)
      പ്രസ്താവനകൾ 1;
         else if (പരിശോധനാ പ്രയോഗം 2)
            പ്രസ്താവനകൾ 2;
               else if (പരിശോധനാ പ്രയോഗം 3)
                  പ്രസ്താവനകൾ 3;
                     ...............
                        else
                           പ്രസ്താവനകൾ n;
```

```
if (test expression 1)
      statement block 1;
         else if (test expression 2)
            statement block 2;
               else if (test expression 3)
                  statement block 3;
                     ...............
                        else
                           statement block n;
```

ആദ്യം പരിശോധനാ പ്രയോഗം 1 വിലയിരുത്തുമ്പോൾ അത് ശരിയാണെങ്കിൽ പ്രസ്താവനകൾ 1 പ്രവർത്തിച്ചതിനുശേഷം ലാഡറിൽ നിന്ന് പുറത്തേക്ക് വരുന്നു. അതായത് ലാഡറിന്റെ ബാക്കി ഭാഗം ഒഴിവാക്കപ്പെടുന്നു. പരിശോധനാപ്രയോഗം 1 വിലയിരുത്തുമ്പോൾ അത് തെറ്റാണെങ്കിൽ പരിശോധനാ പ്രയോഗം 2 വിലയിരുത്തുന്നു. ഈ പ്രക്രിയ അങ്ങിനെ തുടരുന്നു. ഏതെങ്കിലും ഒരു പരിശോധനാപ്രയോഗം ശരിയാണെങ്കിൽ അതിനനുസൃതമായ പ്രസ്താവനകൾ പ്രവർത്തിച്ചതിനുശേഷം നിയന്ത്രണം ലാഡറിന്റെ പുറത്തേക്ക് വരുന്നു. എല്ലാ പരിശോധനാ പ്രയോഗങ്ങളും വിലയിരുത്തുമ്പോൾ തെറ്റാണെങ്കിൽ അവസാന else-നുശേഷമുള്ള പ്രസ്താവനകൾ n പ്രവർത്തിക്കുന്നു. വാക്യഘടനയിൽ കൊടുത്തിരിക്കുന്ന ഇൻഡന്റേഷൻ നിരീക്ഷിക്കുകയും else if ലാഡർ ഉപയോഗിക്കുന്നതിന് ഈ രീതി പിന്തുടരുകയും ചെയ്യുക.

ഒരു വിദ്യാർത്ഥിക്ക് ഒരു വിഷയത്തിൽ 100 ൽ ലഭ്യമായ സ്കോറിന്റെ അടിസ്ഥാനത്തിൽ ഗ്രേഡ് കണ്ടുപിടിക്കാനുള്ള പ്രോഗ്രാം else if ലാഡർ ഉപയോഗിച്ച് നമുക്ക് വിവരിക്കാം.

താഴെയുള്ള പട്ടികയിൽ കൊടുത്തിരിക്കുന്ന മാനദണ്ഡമനുസരിച്ചാണ് ഗ്രേഡ് കണ്ടുപിടിക്കേണ്ടത്.

സ്കോർ	ഗ്രേഡ്
80 ഓ അതിൽ കൂടുതലോ	A
60 മുതൽ 79 വരെ	B
40 മുതൽ 59 വരെ	C
30 മുതൽ 39 വരെ	D
30 ൽ താഴെ	E

പ്രോഗ്രാം 7.5: തന്നിരിക്കുന്ന സ്കോറിന്റെ അടിസ്ഥാനത്തിൽ വിദ്യാർത്ഥിയുടെ ഗ്രേഡ് കണ്ടുപിടിക്കുന്നതിന്

```
#include <iostream>
using namespace std;
int main()
{
    int score;
    cout << "Enter your score: ";
    cin >> score;
    if (score >= 80)
        cout << "A Grade";
      else if (score >= 60)
          cout << "B Grade ";
        else if (score >= 40)
            cout << "C grade";
          else if (score >= 30)
              cout << "D grade";
            else
                cout << "E Grade";
    return 0;
}
```

പ്രോഗ്രാം 7.5 ന്റെ മാതൃക ഔട്ട്പുട്ടുകളാണ് താഴെയുള്ളത്.

ഔട്ട്പുട്ട് 1:

```
Enter your score: 73
B Grade
```

ഔട്ട്പുട്ട് 2:

```
Enter your score: 25
E Grade
```

പ്രോഗ്രാം 7.5 ൽ ആദ്യം പരിശോധനാ പ്രയോഗം `score>=80` വിലയിരുത്തുന്നു. ഔട്ട്പുട്ട് 1-ൽ ഇൻപുട്ട് ചെയ്ത വില 73 ആയതിനാൽ പരിശോധനാ പ്രയോഗം തെറ്റ് ആകുകയും `score>=60`

എന്ന അടുത്ത പരിശോധനാ പ്രയോഗം വിലയിരുത്തുകയും ചെയ്യുന്നു. ഇവിടെ ഇത് ശരിയായതിനാൽ "B Grade" എന്ന് പ്രദർശിപ്പിക്കുകയും else if ലാഡറിന്റെ ബാക്കി ഭാഗം ഒഴിവാക്കുകയും ചെയ്യുന്നു. എന്നാൽ ഔട്ട്പുട്ട് 2 - ൽ എല്ലാ പരിശോധനപ്രയോഗങ്ങളും തെറ്റാണെന്ന് വിലയിരുത്തിയതിനാൽ അവസാനത്തെ else ബ്ലോക്ക് പ്രസ്താവന പ്രവർത്തിക്കുകയും "E grade" എന്ന ഔട്ട്പുട്ട് ലഭിക്കുകയും ചെയ്യുന്നു.

തന്നിരിക്കുന്ന വർഷം അധിവർഷം (leap year) ആണോ അല്ലയോ എന്ന് പരിശോധിക്കുന്നതിനുള്ള ഒരു പ്രോഗ്രാം നമുക്ക് എഴുതാം. ഇൻപുട്ട് സംഖ്യ ശതാബ്ദമാണോ എന്ന് പരിശോധിക്കേണ്ടതുണ്ട് (നൂറുകൊണ്ട് ഹരിക്കാൻ സാധിക്കുന്ന വർഷമാണോ എന്ന്). അത് ഒരു ശതാബ്ദ വർഷമാണെങ്കിൽ അതിനെ 400 കൊണ്ട് കൂടി ഹരിക്കാമെങ്കിലേ അത് അധിവർഷമാകുന്നുള്ളൂ. ഇൻപുട്ട് സംഖ്യ ശതാബ്ദ വർഷമല്ലെങ്കിൽ അതിനെ 4 കൊണ്ട് ഹരിക്കുവാൻ സാധിക്കുമോ എന്ന് നാം പരിശോധിക്കേണ്ടതുണ്ട്. അതിനെ ഹരിക്കാൻ സാധിക്കുമെങ്കിൽ തന്നിരിക്കുന്ന വർഷം അധിവർഷം ആണ്, അല്ലെങ്കിൽ അത് ഒരു അധിവർഷമല്ല.

പ്രോഗ്രാം 7.6 തന്നിരിക്കുന്ന വർഷം അധിവർഷമാണോ അല്ലയോ എന്ന് പരിശോധിക്കുന്നതിന്.

```
#include <iostream>
using namespace std;
void main()
{
  int year ;
  cout << "Enter the year (in 4-digits): ";
  cin >> year;
  if (year%100 == 0)// Checks for century year
  {
        if (year%400 == 0)
            cout << "Leap year\n";
        else
            cout<< "Not a leap year\n";
  }
  else if (year%4 == 0)
        cout << "Leap year\n";
  else
        cout<< "Not a leap year\n";
  return 0;
}
```

ശതാബ്ദ വർഷമല്ലാത്തവ അധിവർഷം ആകണമെങ്കിൽ അവയെ 4കൊണ്ട് ഹരിക്കുവാൻ കഴിയണം.

പ്രോഗാം 7.6 ന്റെ ചില മാതൃക ഔട്ട്പുട്ടുകൾ നമുക്ക് നോക്കാം.

ഔട്ട്പുട്ട് 1:

```
Enter the year (in 4-digits): 2000
Leap year
```

ഔട്ട്പുട്ട് 2:

```
Enter the year (in 4-digits): 2014
Not a leap year
```

ഔട്ട്പുട്ട് 3:

```
Enter the year (in 4-digits): 2100
Not a leap year
```

ഔട്ട്പുട്ട് 4:

```
Enter the year (in 4-digits): 2004
Leap year
```

`else if` ലാഡറിന്റെ ഉപയോഗം വിവരിക്കുന്നതിനുള്ള ഒരു പ്രോഗ്രാം കൂടി നമുക്ക് എഴുതാം. പ്രോഗ്രാം 7.7-ൽ ആഴ്ചയിലെ ദിവസത്തെ സൂചിപ്പിക്കുന്നതിനായി 1 മുതൽ 7 വരെയുള്ള സംഖ്യ ഇൻപുട്ടു ചെയ്യുന്നതിന് അനുവദിക്കുകയും അതിനനുസൃതമായ ദിവസത്തിന്റെ പേര് പ്രദർശിപ്പിക്കുകയും ചെയ്യുന്നു. ഇൻപുട്ട് 1 ആണെങ്കിൽ "Sunday" എന്നും 2 ആണെങ്കിൽ "Monday" എന്നും ഔട്ട്പുട്ടുകൾ പ്രദർശിപ്പിക്കപ്പെടുന്നു. ഇതുപോലെ മറ്റു ദിവസങ്ങളും. 1 മുതൽ 7 വരെയുള്ള പരിധിക്ക് പുറത്താണ് ഇൻപുട്ട് എങ്കിൽ "Wrong input" എന്നായിരിക്കും ഔട്ട്പുട്ട്.

പ്രോഗ്രാം 7.7: തന്നിരിക്കുന്ന ദിവസത്തെ സൂചിപ്പിക്കുന്ന സംഖ്യയ്ക്ക് അനുസൃതമായ ദിവസത്തിന്റെ പേര് പ്രദർശിപ്പിക്കുന്നതിന്

```
#include <iostream>
using namespace std;
int main()
{
    int day;
    cout << "Enter the day number (1-7): ";
    cin >> day;
    if (day == 1)
      cout << "Sunday";
        else if (day == 2)
          cout << "Monday";
            else if (day == 3)
              cout << "Tuesday";
                else if (day == 4)
                  cout << "Wednesday";
                    else if (day == 5)
                      cout << "Thursday";
                        else if (day == 6)
                          cout << "Friday";
                            else if (day == 7)
                              cout << "Saturday";
                                else
                                  cout << "Wrong input";
    return 0;
}
```

പ്രോഗ്രാം 7.7 ന്റെ ചില മാതൃക ഔട്ട്പുട്ടുകളാണ് താഴെയുള്ളത്.

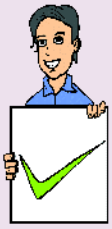
ഔട്ട്പുട്ട് 1:

```
Enter the day number (1-7): 5
Thursday
```

ഔട്ട്പുട്ട് 2:

```
Enter day number (1-7): 9
Wrong input
```

സ്വയം പരിശോധിക്കാം



1. ഒരു പൂർണ്ണ സംഖ്യ ഇൻപുട്ടായി സ്വീകരിച്ച് അത് പോസറ്റീവാണോ നെഗറ്റീവാണോ പൂജ്യമാണോ എന്ന് പരിശോധിക്കുവാനുള്ള പ്രോഗ്രാം if else if പ്രസ്താവന ഉപയോഗിച്ച് എഴുതുക.
2. ഒരു അക്ഷരം (a, b, c അല്ലെങ്കിൽ d) ഇൻപുട്ട് ചെയ്യുന്നതിനും താഴെപറയുന്ന രീതിയിൽ ഔട്ട് പ്രദർശിപ്പിക്കുന്നതിനുമുള്ള ഒരു പ്രോഗ്രാം എഴുതുക.
   "a - abacus", "b - boolean", "c - computer", "d - debugging"
3. ഒരു അക്ഷരം ഇൻപുട്ട് ചെയ്യുന്നതിനും അത് ആൽഫബറ്റാണോ, സംഖ്യയാണോ അതോ മറ്റേതെങ്കിലും ക്യാരക്ടർ ആണോ എന്ന് പ്രിന്റ് ചെയ്യുന്നതിനുള്ള ഒരു പ്രോഗ്രാം എഴുതുക.

7.1.5. switch പ്രസ്താവന (switch statement)

else if ലാഡറിന്റെ സഹായത്തോടെ ബഹുശാഖീകരണം (Multiple branching) എന്ന ആശയം നാം കണ്ടു കഴിഞ്ഞു. C++-ലെ മറ്റൊരു രൂപകൽപ്പനയായ switch പ്രസ്താവന ഉപയോഗിച്ച് ഇവയിൽ ചില പ്രോഗ്രാമുകൾ എഴുതുവാൻ സാധിക്കും. ഈ തിരഞ്ഞെടുക്കൽ പ്രസ്താവന ഒരു വേരിയബിളിന്റേയോ ഒരു പ്രയോഗത്തിന്റേയോ (expression) വിലയെ ഒരു കൂട്ടം പൂർണ സംഖ്യകളുമായോ അക്ഷര സ്ഥിരാങ്കങ്ങളുമായോ തുടർച്ചയായി പരിശോധിക്കുന്നു. switch പ്രസ്താവനയുടെ വാക്യഘടന ചുവടെ ചേർത്തിരിക്കുന്നു.

```
switch (പ്രയോഗം)
        case സ്ഥിരാംഗം_1     : പ്രസ്താവനകൾ 1;
                               break;
        case സ്ഥിരാംഗം_2     : പ്രസ്താവനകൾ 2;
                               break;
        case സ്ഥിരാംഗം_3     : പ്രസ്താവനകൾ 3;
                               break;
                             :
                             :
        case സ്ഥിരാംഗം_n-1   : പ്രസ്താവനകൾ n-1;
                               break;
        default              : പ്രസ്താവനകൾ n;
    }
```

```
switch (expression)
        'case' constant_1   : statement block 1;
                              break;
        'case' constant_2   : statement block 2;
                              break;
        'case' constant_3   : statement block 3;
                              break;
                            :
                            :
        'case' constant_n-1  :  statement block n-1;
                              break;
        default             : statement block n;
    }
```

വാക്യഘടനയിൽ switch, case, break, default എന്നിവ കീ വേർഡുകളാണ് (Keyword). ഒരു പൂർണ്ണസംഖ്യയോ ഒരു ക്യാരക്ടർ കോൺസ്റ്റന്റോ കിട്ടാവുന്ന രീതിയിൽ പ്രയോഗത്തെ വിലയിരുത്തുകയും അത് case പ്രസ്താവനകളിൽ കൊടുത്തിരിക്കുന്ന സ്ഥിരാംഗങ്ങൾക്ക് തുല്യമാണോ എന്ന് നോക്കുകയും ചെയ്യുന്നു. ഒരു തുല്യത കണ്ടെത്തിയാൽ ആ case-നോട് അനുബന്ധിച്ചുള്ള പ്രസ്താവനകൾ പ്രവർത്തിക്കും (break പ്രസ്താവന വരെയോ അല്ലെങ്കിൽ switch പ്രസ്താവനയുടെ അവസാനം വരെയോ). തുല്യത കണ്ടെത്തിയില്ലെങ്കിൽ default ബ്ലോക്കിലെ പ്രസ്താവന കൂട്ടം പ്രവർത്തിക്കും. default പ്രസ്താവന നിർബന്ധമല്ല. അത് ഉപയോഗിച്ചിട്ടില്ലെങ്കിൽ തുല്യത കണ്ടെത്താനാവാത്ത സന്ദർഭങ്ങളിൽ മറ്റൊന്നും പ്രവർത്തിക്കുകയില്ല.

switch-നകത്ത് ഉപയോഗിച്ചിരിക്കുന്ന break പ്രസ്താവന C++-ലെ ഒരു ജമ്പ് പ്രസ്താവനയാണ്. break പ്രസ്താവനയിൽ എത്തുമ്പോൾ പ്രോഗ്രാം നിയന്ത്രണം switch പ്രസ്താവനയ്ക്ക് ശേഷമുള്ള പ്രസ്താവനകളിലേക്ക് പോകുന്നു.

ഭാഗം 7.3.2 ൽ break സ്റ്റേറ്റ്മെന്റിനെക്കുറിച്ച് വിശദമായി നമുക്ക് ചർച്ച ചെയ്യാം. പ്രോഗ്രാം 7.7നെ switch പ്രസ്താവന ഉപയോഗിച്ച് എഴുതാവുന്നതാണ്. ഇത് കോഡിന്റെ വായനാ സുഖവും ഫലപ്രാപ്തിയും വർദ്ധിപ്പിക്കുന്നു. പ്രോഗ്രാം 7.8 ൽ വരുത്തിയ ഭേദഗതികൾ ശ്രദ്ധിക്കുക.

പ്രോഗ്രാം 7.8: switch പ്രസ്താവന ഉപയോഗിച്ച് ആഴ്ചയിലെ ദിവസം പ്രദർശിപ്പിക്കുന്നതിന്.

```
#include <iostream>
using namespace std;
int main()
{     int day ;
      cout << "Enter a number between 1 and 7: ";
      cin >> day ;
      switch (day)
      {
         case 1: cout << "Sunday";
                 break;
```

```
        case 2: cout << "Monday";
                break;
        case 3: cout << "Tuesday";
                break;
        case 4: cout << "Wednesday";
                break;
        case 5: cout << "Thursday";
                break;
        case 6: cout << "Friday";
                break;
        case 7: cout << "Saturday";
                break;
        default: cout << "Wrong input";
     }
     return 0;
}
```

പ്രോഗ്രാം 7.7-ന്റെ ഔട്ട്പുട്ട് തന്നെയായിരിക്കും പ്രോഗ്രാം 7.8-നും. ചില മാതൃകകൾ താഴെ കൊടുത്തിരിക്കുന്നു.

ഔട്ട്പുട്ട് 1:

```
     Enter a number between 1 and 7: 5
     Thursday
```

ഔട്ട്പുട്ട് 2:

```
     Enter a number between 1 and 7: 8
     Wrong input
```

പ്രോഗ്രാം 7.8-ൽ day വേരിയബിന്റെ വില case പ്രസ്താവനയിലെ സ്ഥിരാങ്കങ്ങളുമായി താരതമ്യം ചെയ്തിരിക്കുന്നു. ഒരു തുല്യത കണ്ടെത്തുമ്പോൾ ആ case-നോട് അനുബന്ധിച്ചുള്ള ഔട്ട്പുട്ട് പ്രസ്താവന പ്രവർത്തിക്കുന്നു. വേരിയബിൾ day യ്ക്ക് 5 എന്ന വില നാം ഇൻപുട്ടായി കൊടുത്താൽ അഞ്ചാമത്തെ case പ്രസ്താവന തുല്യമാവുകയും cout << "Thursday"; എന്ന പ്രസ്താവന പ്രവർത്തിക്കുകയും ചെയ്യുന്നു. ഇൻപുട്ട് 8 ആണെങ്കിൽ തുല്യത കണ്ടെത്തുവാൻ ആകില്ല. ആയതിനാൽ default ബ്ലോക്ക് പ്രവർത്തിക്കും.

എല്ലാ break പ്രസ്താവനകളെയും ഒഴിവാക്കുകയാണെങ്കിൽ പ്രോഗ്രാം 7.8-ന്റെ ഔട്ട്പുട്ട് നിങ്ങൾക്ക് പ്രവചിക്കാമോ? case സ്ഥിരാങ്കങ്ങളുമായി day യുടെ വില താരതമ്യം ചെയ്യുന്നു. ആദ്യത്തെ തുല്യത കണ്ടെത്തുമ്പോൾ അനുബന്ധ പ്രസ്താവനകളും തുടർന്നുള്ള പ്രസ്താവനകളും പ്രവർത്തിക്കുന്നു (അവശേഷിക്കുന്ന സ്ഥിരാങ്കങ്ങൾ പരിശോധിക്കാതെ). ചില സാഹചര്യങ്ങളിൽ break പ്രസ്താവന മന:പൂർവ്വം ഒഴിവാക്കാറുണ്ട്. ഒരു switch ലെ തന്നിട്ടുള്ള എല്ലാ case നോടും അനുബന്ധിച്ചുള്ള പ്രസ്താവനകൾ ഒരുപോലെയാണെങ്കിൽ അത്തരം പ്രസ്താവനകൾ അവസാനത്തെ case ൽ നാം എഴുതിയാൽ മതിയാകും. പ്രോഗ്രാം 7.9 ഈ ആശയം വിവരിക്കുന്നു.

പ്രോഗ്രാം 7.9: തന്നിരിക്കുന്ന അക്ഷരം സ്വരാക്ഷരം ആണോ അല്ലയോ എന്ന് പരിശോധിക്കുന്നതിന്.

```
#include <iostream>
using namespace std;
int main()
{
    char ch;
    cout<<"Enter the character to check: ";
    cin>>ch;
    switch(ch)
    {
        case 'A' :
        case 'a' :
        case 'E' :
        case 'e' :
        case 'I' :
        case 'i' :
        case 'O' :
        case 'o' :
        case 'U' :
        case 'u' : cout<<"The given character is a vowel";
                   break;
        default  : cout<<"The given character is not a vowel";
    }
    return 0;
}
```

പ്രോഗ്രാം. 7.9 നൽകുന്ന ചില ഔട്ട്പുട്ടുകൾ താഴെ കാണിച്ചിരിക്കുന്നു.

ഔട്ട്പുട്ട് 1:

```
Enter a character to check: E
The given character is a vowel
```

ഔട്ട്പുട്ട് 2:

```
Enter a character to check: k
The given character is not a vowel
```

Switch ഉപയോഗിക്കുന്നതിന്റെ അനുയോജ്യതയും ആവശ്യകതയും

Switch പ്രസ്താവന else if ലാഡറിന്റെ അനേകം ശാഖകളുള്ള പ്രസ്താവനകൾക്ക് പകരമായാണ് ഉപയോഗിക്കുന്നതെങ്കിലും ഇവ രണ്ടും ഒരേ രീതിയിലല്ല പ്രവർത്തിക്കുന്നത്. C++-ൽ എല്ലാ Switch പ്രസ്താവനകളേയും else if ലാഡർ ഉപയോഗിച്ച് മാറ്റിയെഴുതാം. എന്നാൽ എല്ലാ else if ലാഡറുകളും switch ഉപയോഗിച്ച് മാറ്റിയെഴുതാൻ സാധിക്കില്ല. switch പ്രസ്താവന ഉപയോഗിച്ച് അനേകം ശാഖകൾ നടപ്പിൽ വരുത്തുന്നതിന് താഴെ പറയുന്നവ ആവശ്യമാണ്.

- നിബന്ധനകളിൽ തുല്യത പരിശോധന മാത്രമെ ഉള്ളൂ. മറ്റ് അവസരങ്ങളിൽ അതിനെ തുല്യത പ്രയോഗങ്ങളാക്കി മാറ്റിയെഴുതണം.
- എല്ലാ തുല്യതാ പ്രയോഗങ്ങളിലേയും ആദ്യത്തെ ഓപ്പറൻഡ് (operand) ഒരേ വേരിയബിളോ പ്രയോഗമോ ആയിരിക്കണം.
- ഈ പ്രയോഗങ്ങളിലെ രണ്ടാമത്തെ ഓപ്പറൻഡ് (operand) പൂർണസംഖ്യ (integer) അല്ലെങ്കിൽ ക്യാരക്ടർ കോൺസ്റ്റന്റ് ആയിരിക്കണം.

ഈ അധ്യായത്തിൽ ഇതുവരെ ചർച്ച ചെയ്ത പ്രോഗ്രാം 7.3, പ്രോഗ്രാം 7.7 എന്നിവയിലെ ശാഖകൾ മാത്രമേ `switch` ഉപയോഗിച്ച് മാറ്റിയെഴുതുവാൻ സാധിക്കുകയുള്ളൂ. പ്രോഗ്രാം 7.5-ൽ പരിശോധനാ പ്രയോഗങ്ങൾ `score/10==10`, `score/10==9`, `score/10==8` എന്നിങ്ങനെ മാറ്റം വരുത്തിയാൽ `switch` ഉപയോഗിക്കാം. ഇതുപോലെ മറ്റു സ്കോറുകൾ മാറ്റി എഴുതുക. താഴെകൊടുത്തിരിക്കുന്ന പ്രോഗ്രാംശകലം `else if` ഗോവണിക്കു പകരമായി ഉപയോഗിക്കാം.

```
switch(score/10)
{
    case 10:
    case 9: case 8: cout<< "A Grade"; break;
    case 7: case 6: cout<< "B Grade"; break;
    case 5: case 4: cout<< "C Grade"; break;
    case 3:          cout<< "D Grade"; break;
    default: cout<< "E Grade";
}
```

സ്കോർ `int` ടൈപ്പ് ആയതിനാൽ പ്രയോഗം പൂർണ സംഖ്യകൾ മാത്രമേ നൽകുന്നുള്ളൂ.

`switch` ഉം `else if` ലാഡറും തമ്മിലുള്ള ഒരു താരതമ്യം പട്ടിക 7.1-ൽ കൊടുത്തിരിക്കുന്നു.

`switch` പ്രസ്താവന	`else if` ലാഡർ
1. അനേകം ശാഖകൾ (ബ്രാഞ്ച്) അനുവദിക്കുന്നു.	1. അനേകം ശാഖകൾ (ബ്രാഞ്ച്) അനുവദിക്കുന്നു.
2. തുല്യത (equality) ഓപ്പറേറ്റർ ഉള്ള നിബന്ധനകൾ മാത്രം വിലയിരുത്തുന്നു.	2. ഏതൊരു റിലേഷണൽ/ലോജിക്കൽ പ്രയോഗങ്ങളും വിലയിരുത്തുന്നു.
3. `case` സറിരാങ്കം എപ്പോഴും പൂർണ സംഖ്യയോ അക്ഷരമോ ആയിരിക്കണം.	3. ഫ്ളോട്ടിങ് പോയിന്റ് സറിരാങ്കങ്ങളോ ഒരു പരിധിയിലുള്ള വിലകളോ നിബന്ധനകളിലുൾപ്പെടുത്താം.
4. ഒരു തുല്യതയും ലഭിക്കാത്തപ്പോൾ default പ്രസ്താവന പ്രവർത്തിക്കുന്നു.	4. ഒരു പ്രയോഗവും ശരിയായില്ലെങ്കിൽ else ബ്ലോക്ക് പ്രവർത്തിക്കുന്നു.
5. `switch` പ്രസ്താവനയിൽ നിന്നും പുറത്തു കടക്കുന്നതിന് `break` പ്രസ്താവന ആവശ്യമാണ്.	5. ഒരു ബ്ലോക്ക് പൂർത്തീകരിച്ചതിനുശേഷം പ്രോഗ്രാമിന്റെ നിയന്ത്രണം സ്വയം ബ്ലോക്കിന് പുറത്തു പോകുന്നു.
6. ഒരേ വേരിയബിളോ പ്രയോഗമോ ഒരു കൂട്ടം വിലകളുമായി തുല്യത പരിശോധിക്കുന്നതിന് കൂടുതൽ ഫലപ്രദമാണ്.	6. `switch`-നെക്കാൾ വഴക്കമുള്ളതും എളുപ്പത്തിൽ ഉപയോഗിക്കുവാൻ സാധിക്കുന്നതുമാണ്.

പട്ടിക 7.1: `switch` ഉം `else if` ലാഡറും തമ്മിലുള്ള താരതമ്യം

7.1.6 കണ്ടിഷണൽ ഓപ്പറേറ്റർ (Conditional operator (?:))

അധ്യായം 6- ൽ സൂചിപ്പിച്ചതുപോലെ C++ ൽ ഒരു ടെർണറി ഓപ്പറേറ്റർ ഉണ്ട്. ? ഉം : ഉം എന്നീ ചിഹ്നങ്ങൾ (ചോദ്യചിഹ്നവും കോളനും) ഉൾപ്പെടുന്ന കണ്ടിഷണൽ ഓപ്പറേറ്റർ (Conditional operator) ആണ് അത്. ഇത് ഉപയോഗിക്കുന്നതിന് മൂന്ന് ഓപ്പറന്റുകൾ ആവശ്യമാണ്. `if...else` പ്രസ്താവനക്ക് പകരമായി ഇതിനെ ഉപയോഗിക്കാം. ഇതിന്റെ വാക്യഘടന താഴെ കൊടുത്തിരിക്കുന്നു.

പരിശോധനാ പ്രയോഗം ? പ്രയോഗം ശരിയാകുമ്പോൾ പ്രവർത്തിക്കുന്ന കോഡ് : പ്രയോഗം തെറ്റാകുമ്പോൾ പ്രവർത്തിക്കുന്ന കോഡ് ;

```
Test expression ? True_case code : False_case code;
```

പരിശോധനാ പ്രയോഗം ഏതെങ്കിലും റിലേഷണലോ ലോജിക്കലോ ആയ പ്രയോഗം ആകാം. പ്രയോഗം ശരിയാകുമ്പോൾ പ്രവർത്തിക്കുന്ന കോഡ്, പ്രയോഗം തെറ്റാകുമ്പോൾ പ്രവർത്തിക്കുന്ന കോഡ് എന്നിവ സ്ഥിരവിലയോ, വേരിയബിളോ, പ്രയോഗമോ അല്ലെങ്കിൽ പ്രസ്താവനയോ ആകാം. ഇതിന്റെ പ്രവർത്തനം `if else` പ്രസ്താവനയുടെ സഹായത്തോടു കൂടി താഴെ കാണിച്ചിരിക്കുന്നു.

```
if Test experssion (പരിശോധനാ പ്രയോഗം)
    {
    True_case code: (പ്രയോഗം ശരിയാകുമ്പോൾ പ്രവർത്തിക്കുന്ന കോഡ്:)
    }
    else
    {
    False_case code: (പ്രയോഗം തെറ്റാകുമ്പോൾ പ്രവർത്തിക്കുന്ന കോഡ്;)
    }
if  (Test expression)
    {
        True_case code;
    }
    else
    {
        False_case code;
    }
```

`if...else` പ്രവർത്തിക്കുന്നതുപോലെയാണ് കണ്ടിഷണൽ ഓപ്പറേറ്ററും പ്രവർത്തിക്കുന്നത്. പരിശോധനാ പ്രയോഗം വിലയിരുത്തി അത് ശരിയാണെങ്കിൽ 'പ്രയോഗം ശരിയാകുമ്പോൾ പ്രവർത്തിക്കുന്ന കോഡും' (true_case code) തെറ്റാണെങ്കിൽ 'പ്രയോഗം തെറ്റാകുമ്പോൾ പ്രവർത്തിക്കുന്ന കോഡും' (False_case code) തിരഞ്ഞെടുക്കുന്നു. കണ്ടീഷണൽ ഓപ്പറേറ്ററിന്റെ പ്രവർത്തനം പ്രോഗ്രാം 7.10 ൽ വിവരിച്ചിരിക്കുന്നു.

പ്രോഗ്രാം 7.10: കണ്ടീഷണൽ ഓപ്പറേറ്റർ ഉപയോഗിച്ച് ഏറ്റവും വലിയ സംഖ്യ കണ്ടുപിടിക്കുന്നതിന്.

```
#include <iostream>
using namespace std;
int main()
{
int num1, num2;
cout << "Enter two numbers: ";
cin>> num1 >> num2 ;
(num1>num2)? cout<<num1<<" is larger" : cout<<num2<<" is larger";
return 0;
}
```

ഈ പ്രോഗ്രാമിലെ `return 0` പ്രസ്താവനയ്ക്ക് മുമ്പുള്ള പ്രസ്താവനയിൽ കണ്ടീഷണൽ ഓപ്പറേറ്റർ ഉപയോഗിക്കുന്നതുകൊണ്ട് അതിനെ കണ്ടീഷണൽ പ്രസ്താവന എന്നു വിളിക്കുന്നു. ഈ പ്രസ്താവനയെ താഴെയുള്ള കോഡ് ശകലം ഉപയോഗിച്ച് മാറ്റി എഴുതാവുന്നതാണ്.

```
int big = (num1>num2)? num1 : num2;
cout<< big << "is larger";
```

പരിശോധനാ പ്രയോഗം ശരിയാണെങ്കിൽ `num1`-ന്റെ വിലയും തെറ്റാണെങ്കിൽ `num2` ന്റെ വിലയും ആയിരിക്കും `big` ലേക്ക് ശേഖരിക്കുക. ഇവിടെ കണ്ടീഷണൽ ഓപ്പറേറ്റർ ഉപയോഗിച്ചാണ് ഒരു കണ്ടീഷണൽ പ്രയോഗം ഉണ്ടാക്കിയിരിക്കുന്നത്. ഈ പ്രയോഗത്തിൽ നിന്നും ലഭിക്കുന്ന വില `big` ലേക്ക് ശേഖരിക്കുന്നു.

കണ്ടീഷണൽ പ്രയോഗത്തിന്റെ ഒരു സങ്കീർണ രൂപം താഴെ കൊടുത്തിരിക്കുന്നു. ഇത് മൂന്ന് സംഖ്യകളിൽ ഏറ്റവും വലുത് നൽകുന്നു. `n1`, `n2`, `n3`, `big` എന്നിവ പൂർണ സംഖ്യ വേരിയബിളുകളാണെങ്കിൽ,

```
big = (n1>n2) ? ( (n1>n3)?n1:n3 ) : ( (n2>n3)?n2:n3);
```

പ്രോഗ്രാം 7.4 പരിശോധിച്ച് മുകളിൽ കൊടുത്തിരിക്കുന്ന കണ്ടീഷണൽ പ്രയോഗം എങ്ങനെയാണ് നെസ്റ്റഡ് `if` നുപകരമായി ഉപയോഗിച്ചിരിക്കുന്നത് എന്ന് നോക്കുക.

സ്വയം പരിശോധിക്കാം

1. 1 മുതൽ 12 വരെയുള്ള സംഖ്യകൾ ഇൻപുട്ട് ചെയ്ത് അതിനനുസൃതമായ മാസത്തിന്റെ പേര് പ്രദർശിപ്പിക്കുന്നതിനുള്ള പ്രോഗ്രാം എഴുതുക. (1 ആണെങ്കിൽ January, 2 ആണെങ്കിൽ February എന്നിങ്ങനെ)
2. `switch` പ്രസ്താവന ഉപയോഗിച്ച് അരിത്തമറ്റിക് ഓപ്പറേഷനുകൾ ചെയ്യുവാനുള്ള ഒരു പ്രോഗ്രാം എഴുതുക. ഇതിനുവേണ്ടി 2 ഓപ്പറന്റുകളും ഒരു ഓപ്പറേറ്ററും ഇൻപുട്ടായി സ്വീകരിക്കുക.
3. `switch` പ്രസ്താവനയ്ക്കകത്തുള്ള `break` പ്രസ്താവനയുടെ പ്രാധാന്യം എന്താണ്?
4. 0 മുതൽ 9 വരെയുള്ള ഏതെങ്കിലുമൊരു സംഖ്യ ഇൻപുട്ട് ചെയ്ത് അതിനെ അക്ഷരത്തിലെഴുതുവാനുള്ള ഒരു പ്രോഗ്രാം. `switch` പ്രസ്താവന ഉപയോഗിച്ച് എഴുതുക.

5. ഒരു സംഖ്യ ഇൻപുട്ടായി സ്വീകരിച്ച് ആ സംഖ്യ 5 ന്റെ ഗുണിതമാണ് എന്ന് പരിശോധിക്കുന്നതിനുള്ള ഒരു പ്രോഗ്രാം തിരഞ്ഞെടുക്കൽ പ്രസ്താവനയും കണ്ടീഷണൽ ഓപ്പറേറ്ററും ഉപയോഗിച്ച് എഴുതുക.
6. താഴെ കൊടുത്തിരിക്കുന്ന പ്രസ്താവന `if..else` ഉപയോഗിച്ച് മാറ്റിയെഴുതുക.

```
result = (mark>30)? 'P' : 'F';
```

7.2. ആവർത്തന പ്രസ്താവനകൾ (Iteration statements)

അധ്യായം 4-ൽ നാം ചർച്ച ചെയ്ത ചില പ്രശ്നങ്ങളുടെ ഉത്തരങ്ങളിൽ ആവർത്തന സ്വഭാവമുള്ള പ്രവർത്തികൾ അടങ്ങിയിട്ടുണ്ട്. പ്രോഗ്രാമുകൾ എഴുതുമ്പോൾ ഒന്നോ അതിലധികമോ പ്രസ്താവനകളെ പല തവണ പ്രവർത്തിപ്പിക്കുന്നതിനായി ഭാഷയുടെ പ്രത്യേക രൂപകൽപ്പനകൾ നാം ഉപയോഗിക്കുന്നു. ഇത്തരം രൂപകൽപ്പനകളെ **ആവർത്തന പ്രസ്താവനകൾ** (Iteration statements) അല്ലെങ്കിൽ **ലൂപ്പിങ് പ്രസ്താവനകൾ** എന്നു വിളിക്കുന്നു. C++-ൽ മൂന്ന് തരം ആവർത്തന പ്രസ്താവനകൾ ഉണ്ട്. ഒരു നിബന്ധന ശരിയാകുമ്പോൾ ഒരു കൂട്ടം പ്രസ്താവനകൾ ആവർത്തിച്ച് പ്രവർത്തിപ്പിക്കുവാൻ ഇവ അനുവദിക്കുന്നു.

ലൂപ്പ് എന്ന ആശയം നിത്യജീവിതത്തിൽ നാം പ്രയോഗിക്കാറുണ്ട്. നമുക്ക് ഒരു സാഹചര്യം പരിഗണിക്കാം. പരീക്ഷയിൽ A+ ഗ്രേഡ് ലഭിക്കുന്ന എല്ലാ വിദ്യാർത്ഥികൾക്കും നിങ്ങളുടെ ക്ലാസ് ടീച്ചർ ഒരു സമ്മാനം തരുമെന്ന് പ്രഖ്യാപിച്ചു എന്ന് വിചാരിക്കുക. സമ്മാനം പൊതിയാനുള്ള ചുമതല നിങ്ങളെ ഏൽപ്പിക്കുന്നു. സമ്മാനം പൊതിയേണ്ടതെങ്ങനെയെന്ന് താഴെ കൊടുത്ത രീതിയിൽ ടീച്ചർ വിശദീകരിക്കുന്നു.

ഘട്ടം 1 : സമ്മാനം എടുക്കുക.

ഘട്ടം 2 : പൊതിയാനുള്ള പേപ്പർ മുറിക്കുക.

ഘട്ടം 3 : സമ്മാനം പൊതിയുക.

ഘട്ടം 4 : റിബൺ ഉപയോഗിച്ച് കവർ കെട്ടുക.

ഘട്ടം 5 : കാർഡിൽ പേരെഴുതി സമ്മാനത്തിന് മുകളിൽ ഒട്ടിക്കുക.

പരീക്ഷയിൽ 30 വിദ്യാർഥികൾക്ക് A+ ഗ്രേഡ് ഉണ്ടെങ്കിൽ ഇതേ പ്രവർത്തി 30 തവണ നിങ്ങൾ ആവർത്തിക്കേണ്ടതുണ്ട്. സമ്മാനം പൊതിയുന്ന ഈ പ്രവർത്തി 30 തവണ ആവർത്തിക്കുന്നതിന് താഴെ കൊടുത്ത രീതിയിൽ നിർദ്ദേശങ്ങൾ പുനഃക്രമീകരിക്കാം.

താഴെ കൊടുത്ത ഘട്ടങ്ങൾ 30 തവണ ആവർത്തിക്കുക

{

അടുത്ത സമ്മാനം എടുക്കുക.

പൊതിയാനുള്ള പേപ്പർ മുറിക്കുക.

സമ്മാനം പൊതിയുക.

റിബൺ ഉപയോഗിച്ച് കവർ കെട്ടുക.

കാർഡിൽ പേരെഴുതി സമ്മാനത്തിന് മുകളിൽ ഒട്ടിക്കുക.

}

ഇനി വേറൊരു ഉദാഹരണമെടുക്കാം. കമ്പ്യൂട്ടർ ആപ്ലിക്കേഷൻ വിഷയത്തിൽ ലഭിച്ച സ്കോറുകളുടെ ക്ലാസ് ശരാശരി നമുക്ക് കണ്ടുപിടിക്കണമെന്ന് കരുതുക. അതിനായി താഴെ പറയുന്ന ഘട്ടങ്ങളിലൂടെ കടന്നു പോകണം.

ആകെ-സ്കോറിന് പ്രാരംഭ വിലയായി പൂജ്യം കൊടുക്കുക.

താഴെ പറയുന്ന ഘട്ടങ്ങൾ ആദ്യത്തെ വിദ്യാർഥിക്ക് മുതൽ അവസാനത്തെ ആൾ വരെ ആവർത്തിക്കുക.

{

വിദ്യാർത്ഥിയുടെ സ്കോർ ആകെ-സ്കോറിനോട് കൂട്ടുക.

അടുത്ത വിദ്യാർത്ഥിയുടെ സ്കോർ എടുക്കുക.

}

ശരാശരി = ആകെ - സ്കോർ/ക്ലാസ്സിലെ ആകെ വിദ്യാർത്ഥികളുടെ എണ്ണം

ഈ രണ്ടു ഉദാഹരണങ്ങളിലും ചില ഘട്ടങ്ങൾ നാം പല തവണ ചെയ്യുന്നു. പ്രക്രിയ എത്ര തവണ ആവർത്തിച്ചു എന്നറിയുന്നതിന് നാം ഒരു കൗണ്ടർ (counter) ഉപയോഗിക്കുന്നു. പ്രവർത്തനം തുടരണമോ വേണ്ടയോ എന്ന് കൗണ്ടറിന്റെ വില തീരുമാനിക്കുന്നു. നിബന്ധനയ്ക്ക് വിധേയമായി ലൂപ്പുകൾ പ്രവർത്തിക്കുന്നതിനാൽ കൗണ്ടർ പോലുള്ള വേരിയബിൾ ലൂപ്പ് നിർമ്മിക്കുന്നതിന് ഉപയോഗിക്കുന്നു. ഈ വേരിയബിൾ പൊതുവെ ലൂപ്പ് നിയന്ത്രണവേരിയബിൾ (Loop control variable) എന്നറിയപ്പെടുന്നു. എന്തുകൊണ്ടെന്നാൽ യഥാർത്ഥത്തിൽ ഇതാണ് ലൂപ്പിന്റെ പ്രവർത്തനത്തെ നിയന്ത്രിക്കുന്നത്. അധ്യായം 4-ൽ ഒരു ലൂപ്പിന്റെ 4 ഘടകങ്ങളെക്കുറിച്ച് നാം ചർച്ച ചെയ്തു. നമുക്ക് അതൊന്ന് ഓർത്തെടുക്കാം.

1. **പ്രാരംഭ വില നൽകൽ (Initialisation)** : ലൂപ്പിലേക്ക് പ്രവേശിക്കുന്നതിനു മുമ്പ് അതിന്റെ നിയന്ത്രണ വേരിയബിളിന് പ്രാരംഭ വില നൽകണം. അങ്ങനെ ലൂപ്പ് നിയന്ത്ര വേരിയബിളിന് അതിന്റെ ആദ്യത്തെ വില ലഭിക്കും. പ്രാരംഭ വില നൽകുന്ന പ്രസ്താവന ലൂപ്പിന്റെ തുടക്കത്തിൽ മാത്രമേ പ്രവർത്തിക്കുന്നുള്ളൂ.

2. **പരിശോധനാ പ്രയോഗം (Test Expression)** : ഇത് ഒരു റിലേഷണൽ അല്ലെങ്കിൽ ലോജിക്കൽ പ്രയോഗമാണ്. ഇതിന്റെ വില ശരി അല്ലെങ്കിൽ തെറ്റ് ആയിരിക്കും. ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിക്കണോ വേണ്ടയോ എന്ന് ഇത് തീരുമാനിക്കുന്നു. പരിശോധനാ പ്രയോഗം ശരിയാണെങ്കിൽ ലൂപ്പ് പ്രവർത്തിക്കുന്നു. അല്ലെങ്കിൽ അത് പ്രവർത്തിക്കില്ല.

3. **പരിഷ്കരിക്കൽ പ്രസ്താവന (Updation Statement)** : പരിഷ്കരിക്കൽ പ്രസ്താവന ലൂപ്പ് നിയന്ത്രണ വേരിയബിളിന്റെ വിലയിൽ മാറ്റം വരുത്തുന്നു. ഈ പ്രസ്താവന അടുത്ത ആവർത്തനത്തിന് മുന്നേ പ്രവർത്തിക്കുന്നു.

4. **ലൂപ്പിന്റെ ചട്ടക്കൂട് (Body of loop)** : ആവർത്തിക്കപ്പെടേണ്ട പ്രസ്താവനകൾ ഉപയോഗിച്ച് ലൂപ്പിന്റെ ചട്ടക്കൂട് രൂപപ്പെടുത്തുന്നു. ഇതിൽ ഒന്നോ അതിലധികമോ പ്രസ്താവനകൾ ഉണ്ടായിരിക്കും. ലൂപ്പുകളെ പൊതുവെ ആഗമന നിയന്ത്രണ ലൂപ്പുകൾ (Entry controlled loop) എന്നും ബഹിർഗമന നിയന്ത്രണ ലൂപ്പുകൾ (Exit controlled loop) എന്നും തരംതിരിച്ചിരിക്കുന്നു എന്ന് അധ്യായം 4-ൽ നാം പഠിച്ചു. C++ ൽ മൂന്നുതരം ലൂപ്പ് പ്രസ്താവനകൾ ഉണ്ട്: `while loop`, `for loop`, `do-while loop`. ഓരോന്നിന്റേയും പ്രവർത്തനം വിശദമായി നമുക്ക് ചർച്ച ചെയ്യാം.

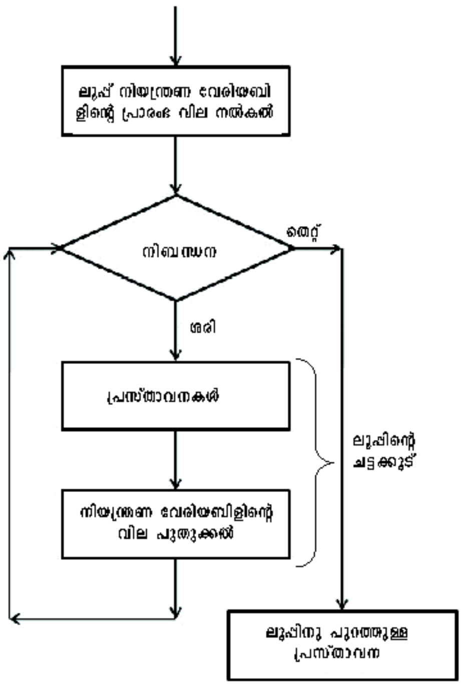
7.2.1 while പ്രസ്താവന (while statement)

while ലൂപ്പ് ഒരു ആഗമന നിയന്ത്രണ ലൂപ്പ് ആണ്. നിബന്ധന (Condition) ആദ്യം പരിശോധിക്കുകയും അത് ശരിയാണെങ്കിൽ ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിക്കുകയും ചെയ്യുന്നു. അതായത് നിബന്ധന ശരിയാകുന്നിടത്തോളം ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിക്കും. while ലൂപ്പിന്റെ വാക്യഘടന ഇതാണ്.

```
നിയന്ത്രണ വേരിയബിളിന്റെ പ്രാരംഭ വില നൽകൽ;
    while(പരിശോധനാ പ്രയോഗം)
    {
         ലൂപ്പിന്റെ ചട്ടക്കൂട്;
         ലൂപ്പ് നിയന്ത്രണ വേരിയബിളിനെ പുതുക്കൽ;
    }
intialisation of loop control variable;
while (test expression)
{
    body of the loop;
    updation of loop control variable;
}
```

ഇവിടെ പരിശോധനാ പ്രയോഗം നിബന്ധന നിർവചിക്കുകയും അത് ലൂപ്പിനെ നിയന്ത്രിക്കുകയും ചെയ്യുന്നു. ലൂപ്പിന്റെ ചട്ടക്കൂട് ഒരു പ്രസ്താവനയോ ഒന്നിലധികം പ്രസ്താവനകളോ അല്ലെങ്കിൽ പ്രസ്താവനകളില്ലാതെയോ ആകാം. ആവർത്തിച്ചു പ്രവർത്തിക്കുന്നതിനുള്ള ഒരു കൂട്ടം പ്രസ്താവനകളാണ് ലൂപ്പിന്റെ ചട്ടക്കൂട്. ലൂപ്പ് നിയന്ത്രണ വേരിയബിളിന്റെ വില വ്യത്യാസപ്പെടുത്തുന്ന പ്രസ്താവനയാണ് പരിഷ്കരിക്കൽ പ്രസ്താവന. ഒരു while ലൂപ്പിൽ ലൂപ്പ് നിയന്ത്രണ വേരിയബിളിന് ലൂപ്പ് തുടങ്ങുന്നതിനുമുമ്പ് പ്രാരംഭവില നൽകുകയും ലൂപ്പ് ചട്ടക്കൂടിനുള്ളിൽ വച്ച് അതു പുതുക്കുകയും ചെയ്യുന്നു. ഒരു while ലൂപ്പിന്റെ പ്രവർത്തന ചിത്രം 7.3-ലെ ഫ്ളോചാർട്ടിൽ വിവരിച്ചിരിക്കുന്നു.

നിയന്ത്രണ വേരിയബിളിന് പ്രാരംഭ വില നൽകുകയാണ് ആദ്യം ചെയ്യുന്നത്. പിന്നീട് പരിശോധനാ പ്രയോഗം വിലയിരുത്തുന്നു. അത് ശരിയാണ് എങ്കിൽ ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിക്കുന്നു. അതുകൊണ്ടാണ് while ലൂപ്പിനെ ആഗമന നിയന്ത്രണ ലൂപ്പ് എന്ന് വിളിക്കുന്നത്. ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിക്കുന്നതിനൊപ്പം ലൂപ്പ് നിയന്ത്രണ വേരിയബിളിന്റെ വിലയും പുതുക്കുന്നു. ലൂപ്പ് ചട്ടക്കൂട്ടിന്റെ പ്രവർത്തനം കഴിഞ്ഞതിനു ശേഷം പരിശോധനാപ്രയോഗം വീണ്ടും വിലയിരുത്തു



ചിത്രം 7.3. while ലൂപ്പിന്റെ പ്രവർത്തനം

ന്നു. നിബന്ധന ശരിയായിരിക്കുന്നിടത്തോളം ഈ പ്രക്രിയ തുടരുന്നു. `while` ലൂപ്പിന്റെ പ്രവർത്തനം വിവരിക്കുന്നതിനുള്ള ഒരു കോഡ് ശകലം നമുക്ക് ഇപ്പോൾ പരിഗണിക്കാം.

```
int k=1;
while(k<=3)
{
    cout << k << '\t';
    ++k;
}
cout << "\n Program Ends";
```

ലൂപ്പിന് മുമ്പേ പ്രാരംഭ വില നൽകൽ

പരിശോധനാ പ്രയോഗം

ലൂപ്പിന്റെ ചട്ടക്കൂട്

ലൂപ്പിന്റെ ചട്ടക്കൂടിനകത്തെ പുതുക്കൽ പ്രസ്താവന

ഈ കോഡ് ശകലത്തിൽ k എന്ന ലൂപ്പ് നിയന്ത്രണ വേരിയബിളിന് 1 എന്ന വില ആദ്യം നൽകിയിരിക്കുന്നു. പിന്നീട് k<=3 എന്ന പരിശോധനാ പ്രയോഗമായ വിലയിരുത്തുന്നു. ഇത് ശരിയായതു കൊണ്ട് ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിക്കുന്നു. അതായത് k-യുടെ വിലയായ 1 സ്ക്രീനിൽ പ്രദർശിപ്പിക്കുന്നു. അതിനുശേഷം പരിഷ്കരിക്കൽ പ്രസ്താവനയായ (update statement) ++k പ്രവർത്തിച്ച് k യുടെ വില 2 ആയി മാറുകയും ചെയ്യുന്നു. നിബന്ധന (k<=3) ഒന്നുകൂടി പരിശോധിച്ച് ശരിയാണെന്ന് കണ്ടെത്തുകയും ചെയ്തു. പ്രോഗ്രാമിന്റെ നിയന്ത്രണം ലൂപ്പിനകത്ത് പ്രവേശിച്ച് k യുടെ വില 2 എന്ന് സ്ക്രീനിൽ പ്രദർശിപ്പിക്കുന്നു. വീണ്ടും പരിഷ്കരിക്കൽ പ്രസ്താവന ആവർത്തിക്കുകയും k യുടെ വില 3 ആകുകയും ചെയ്യുന്നു. നിബന്ധന ഇപ്പോഴും ശരിയായതിനാൽ ലൂപ്പ് പ്രവർത്തിച്ച് 3 എന്ന് സ്ക്രീനിൽ പ്രദർശിപ്പിക്കുന്നു. k യുടെ വില വീണ്ടും പരിഷ്കരിച്ച 4 ആവുകയും ഇപ്പോൾ പരിശോധന പ്രയോഗത്തിനന്റെ ഫലം തെറ്റാവുകയും ചെയ്യുന്നു. നിയന്ത്രണം ലൂപ്പിന് പുറത്തേക്ക് വരുകയും `while` ലൂപ്പിന് പുറത്തുള്ള അടുത്ത പ്രസ്താവന പ്രവർത്തിക്കുകയും ചെയ്യുന്നു. ചുരുക്കത്തിൽ കോഡിന്റെ ഔട്ട്പുട്ട് താഴെ കൊടുത്തിരിക്കുന്നത് പോലെയായിരിക്കും.

```
1      2      3
Program ends
```

k-യുടെ പ്രാരംഭ വില 5 ആണെങ്കിൽ എന്ത് സംഭവിക്കുമെന്ന് സങ്കൽപ്പിക്കുക? ആദ്യം വിലയിരുത്തുമ്പോൾ തന്നെ പരിശോധന പ്രയോഗം തെറ്റായതിനാൽ ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിക്കുകയില്ല. ലൂപ്പിന്റെ ചട്ടക്കൂടിലെക്കുള്ള പ്രവേശനം `while loop` നിയന്ത്രിക്കുന്നുവെന്ന് ഇത് വ്യക്തമായി കാണിക്കുന്നു.

ആദ്യത്തെ 10 എണ്ണൽ സംഖ്യകളെ `while loop` ഉപയോഗിച്ച് പ്രിന്റ് ചെയ്യുന്നതിനുള്ള ഒരു പ്രോഗ്രാം നമുക്ക് നോക്കാം.

പ്രോഗ്രാം 7.11: ആദ്യത്തെ 10 എണ്ണൽ സംഖ്യകൾ പ്രിന്റ് ചെയ്യുന്നതിന്

```
#include<iostream>
using namespace std;
int main()
{
    int n = 1;
```

ലൂപ് നിയന്ത്രണ വേരിയബളിന് പ്രാരംഭ വില നൽകൽ.

```
    while(n <= 10)
    {
        cout<< n << "   ";
        ++n;
    }
    return 0;
}
```

പരിശോധനാ പ്രയോഗം

ലൂപ്പിന്റെ ചട്ടക്കൂട്

ലൂപ്പ് നിയന്ത്രണ വേരിബളിന്റെ വില പുതുക്കൽ

പ്രോഗ്രാം 7.11 ന്റെ ഔട്ട്പുട്ട് താഴെ കൊടുത്തിരിക്കുന്ന പോലെ ആയിരിക്കും.

```
1  2  3  4  5  6  7  8  9  10
```

20 വരെയുള്ള ഇരട്ട സംഖ്യകളുടെ തുക കണ്ട്പിടിക്കുന്നതിന് പ്രോഗ്രാം 7.12 `while` ലൂപ്പ് ഉപയോഗിക്കുന്നു. ലൂപ്പ് വേരിയബിളിന്റെ വില ഏത് ഓപ്പറേഷനുപയോഗിച്ചും പരിഷ്കരിക്കാമെന്ന് ഈ പ്രോഗ്രാം കാണിക്കുന്നു.

പ്രോഗ്രാം 7.12: 20 വരെയുള്ള ഇരട്ടസംഖ്യകളുടെ തുക കണ്ടുപിടിക്കുന്നതിന്

```
#include<iostream>
using namespace std;
int main()
{
    int i, sum = 0;
    i = 2;
    while( i<= 20)
    {
        sum = sum + i;
        i = i + 2;
    }
    cout<<"\nThe sum of even numbers up to 20 is: "<<sum;
    return 0;
}
```

നിലവിലുള്ള വിലയോട് രണ്ട് കൂട്ടിച്ചേർത്തു കൊണ്ട് ലൂപ്പ് നിയന്ത്രണ വേരിയബിന്റെ വില പുതുക്കുന്നു.

പ്രോഗ്രാം 7.12 ന്റെ ഔട്ട്പുട്ട് താഴെ കൊടുത്തിരിക്കുന്നു.

```
The sum of even numbers up to 20 is: 110
```

നമുക്ക് ചെയ്യാം

1. *100 - നും 200 - നും ഇടയിലുള്ള എല്ലാ ഒറ്റ സംഖ്യകളും പ്രദർശിപ്പിക്കുവാനായി പ്രോഗ്രാം 7.11 പരിഷ്കരിക്കുക.*
2. *പ്രോഗ്രാം 7.12 പരിഷ്കരിച്ച് ആദ്യത്തെ N എണ്ണൽ സംഖ്യകളുടെ ശരാശരി കണ്ടുപിടിക്കുക.*

while പ്രസ്താവനയിലെ പരിശോധനാ പ്രയോഗത്തിന് ശേഷം നാം ഒരു അർധവിരാമം (;) ഇട്ടാൽ വാക്യഘടനയിൽ തെറ്റൊന്നുമില്ല. എന്നാൽ അതിനുശേഷമുള്ള ബ്രാക്കറ്റുകളിലെ പ്രസ്താവനകളെ ലൂപ്പ് ചട്ടക്കൂടായി പരിഗണിക്കുന്നില്ല. പരിശോധനാ പ്രയോഗം ശരിയാണെങ്കിൽ while ലൂപ്പിനു ശേഷമുള്ള കോഡ് പ്രവർത്തിക്കുകയുമില്ല പ്രോഗ്രാം അവസാനിക്കുകയുമില്ല എന്നതാണ് ഏറ്റവും പരിതാപകരമായ അവസ്ഥ. ഇത് ഒരു അനന്തമായ ലൂപ്പിന് കാരണമാകുന്നു

7.2.2 for പ്രസ്താവന (for statement)

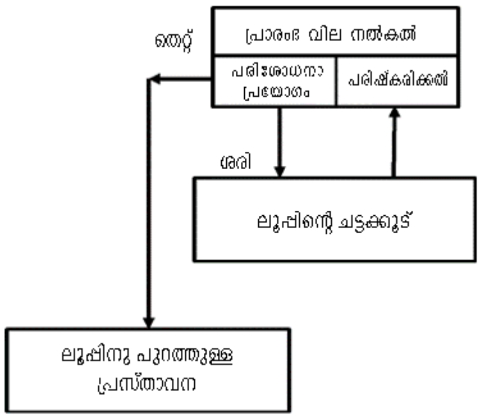
`for` ലൂപ്പും C++-ലെ ഒരു ആഗമന നിയന്ത്രണ ലൂപ്പ് ആണ്. ലൂപ്പിലെ ഘടകങ്ങളായ പ്രാരംഭ വില നൽകൽ, പരിശോധന പ്രയോഗം, പരിഷ്കരിക്കൽ പ്രസ്താവന എന്നിവ ഒരുമിച്ചാണ് `for` പ്രസ്താവനയിൽ നൽകിയിരിക്കുന്നത്. അതുകൊണ്ട് പ്രോഗ്രാം ഒതുക്കമുള്ളതായി തീരുന്നു. വാക്യ ഘടന ഇതാണ്:

```
for (പ്രാരംഭവില നൽകൽ; പരിശോധന പ്രയോഗം; പരിഷ്കരിക്കൽ പ്രസ്താവന)
{
    ലൂപ്പിന്റെ ചട്ടക്കൂട്;
}
for (initialisation; test expression; update statement)
{
    body-of-the-loop;
{
```

`for` ലൂപ്പിന്റെ പ്രവർത്തനം `while` ലൂപ്പിന്റേതുപോലെയാണ്. `while` ലൂപ്പിന്റെ ഫ്ളോചാർട്ട് `for` ലൂപ്പിന്റെ പ്രവർത്തനം വിശദമാക്കുന്നതിന് ഉപയോഗിക്കാവുന്നതാണ്.

`for` ലൂപ്പിൽ മൂന്നു ഘടകങ്ങളും ഒരുമിച്ചു വന്നതിനാൽ എണ്ണുന്ന (Counting) സാഹചര്യങ്ങളിൽ ഈ പ്രസ്താവന ഉപയോഗിക്കുന്നത് അഭികാമ്യമാണ്.

ചിത്രം 7.4 ൽ കൊടുത്തിരിക്കുന്ന ഫ്ളോചാർട്ട് സാധാരണയായി `for` പ്രസ്താവനയുടെ പ്രവർത്തനം കാണിക്കുന്നതിന് ഉപയോഗിക്കുന്നു.



ചിത്രം 7.4: ഫോർലൂപ്പിന്റെ പ്രവർത്തനം.

തുടക്കത്തിൽ, പ്രാരംഭ വില നൽകൽ നടക്കുന്നു, തുടർന്ന് പരിശോധനാ പ്രയോഗം വിലയിരുത്തുന്നു. ഇതിന്റെ ഫലം ശരിയാണെങ്കിൽ ലൂപ്പ് ചട്ടക്കൂട് പ്രവർത്തിക്കുന്നു. അല്ലെങ്കിൽ പ്രോഗ്രാം നിയന്ത്രണം ലൂപ്പിനു പുറത്തേക്കു പോകുന്നു. ലൂപ്പ് ചട്ടക്കൂടിന്റെ പ്രവർത്തനത്തിനുശേഷം പരിഷ്കരിക്കൽ പ്രയോഗം പ്രവർത്തിക്കുകയും പരിശോധനാ പ്രയോഗം വീണ്ടും വിലയിരുത്തുകയും ചെയ്യുന്നു. പരിശോധനാ പ്രയോഗം തെറ്റാവുന്നതു വരെ ഈ മൂന്നു ഘട്ടങ്ങളും (പരിശോധന, ചട്ടക്കൂട്, പരിഷ്കരിക്കൽ) തുടർന്നു കൊണ്ടേയിരിക്കും.

പ്രോഗ്രാം 7.11 ൽ ഉപയോഗിച്ചിരിക്കുന്ന ലൂപ്പ് ശകലത്തെ `for` ലൂപ്പ് ഉപയോഗിച്ച് താഴെ കാണും വിധം മാറ്റി എഴുതാം.

```
for (n=1; n<=10; ++n)
    cout << n << "   ";
```

`while` ലൂപ്പിലേതുപോലെ തന്നെ ഈ കോഡ് പ്രവർത്തിക്കുന്നു.

നമുക്ക് ചെയ്യാം

തൊട്ട് മുമ്പ് സൂചിപ്പിച്ച for ലൂപ്പിന്റെ പ്രവർത്തനക്രമത്തിലെ ഒന്നും രണ്ടും ഘട്ടങ്ങൾ താഴെ കൊടുത്തിരിക്കുന്നു. ബാക്കി ഘട്ടങ്ങൾ എഴുതുക.

ഘട്ടം 1 : n = 1, നിബന്ധന ശരിയാണ്, ഒന്ന് പ്രദർശിപ്പിക്കുന്നു, n ന്റെ വില 2 ആകുന്നു.

ഘട്ടം 2 : നിബന്ധനശരിയാണ് , 2 പ്രദർശിപ്പിക്കുന്നു, n ന്റെ വില 3 ആകുന്നു.

ഘട്ടം 3 : ....

`for` ലൂപ്പ് ഉപയോഗിച്ച് ഒരു സംഖ്യയുടെ ഫാക്റ്റോറിയൽ കണ്ടുപിടിക്കാനുള്ള പ്രോഗ്രാം നമുക്കു എഴുതാം. N എന്ന സംഖ്യയുടെ ഫാക്റ്റോറിയൽ എന്നത് N! എന്ന് സൂചിപ്പിക്കുന്നു. ഇത് ആദ്യത്തെ N എണ്ണൽ സംഖ്യകളുടെ ഗുണനഫലമാണ്. ഉദാഹരണത്തിന് 5 ന്റെ ഫാക്റ്റോറിയൽ (5!)കണക്കാക്കുന്നത് $1 \times 2 \times 3 \times 4 \times 5 = 120$ എന്നാണ്.

പ്രോഗ്രാം 7.13: for ലൂപ്പ് ഉപയോഗിച്ച് ഒരു സംഖ്യയുടെ ഫാക്ടോറിയൽ കണ്ടുപിടിക്കുന്നതിന്.

```
#include <iostream>
using namespace std;
int main()
{   int n, i;
    long fact=1;
    cout<<"Enter the number: ";
    cin>>n;
    for (i=1; i<=n; ++i)
        fact = fact * i;
    cout << "Factorial of " << n << " is " << fact;
    return 0;
}
```

പ്രാരംഭ വില നൽകൽ, പരിശോധനാ പ്രയോഗം, പുതുക്കൽ പ്രസ്താവന

ലൂപ്പ് ചട്ടക്കൂട്

പ്രോഗ്രാം 7.13 ന്റെ ഒരു മാതൃക ഔട്ട്പുട്ട് താഴെ കൊടുത്തിരിക്കുന്നു.

```
Enter the number:  6
Factorial of 6 is 720
```

കമ്പ്യൂട്ടർ ആപ്ലിക്കേഷൻസ് എന്ന വിഷയത്തിലെ സ്കോറുകളുടെ ശരാശരി കാണുന്നതിനുള്ള മറ്റൊരു പ്രോഗ്രാമാണ് താഴെ കൊടുത്തിരിക്കുന്നത് പ്രോഗ്രാം 7.14-ൽ n നു(കുട്ടികളുടെ എണ്ണം) വില സ്വീകരിക്കുകയും പിന്നീട് ഓരോ വിദ്യാർഥികളേയും സ്കോർ ഇൻപുട്ടായി സ്വീകരിച്ച് ശരാശരി സ്കോർ പ്രിന്റ് ചെയ്യുന്നു.

പ്രോഗ്രാം 7.14 n വിദ്യാർഥികളുടെ ശരാശരി സ്കോർ കണ്ടുപിടിക്കുന്നതിന്

```
#include<iostream>
using namespace std;
int main()
{
    int i, sum, score, n;
    float avg;
    cout << "How many students? ";
    cin >> n ;
    for( i=1, sum=0;  i<=n; ++i)
    {
        cout << "Enter the score of student " << i << ": ";
        cin >> score;
        sum = sum + score;
    }
    avg = (float)sum / n;
    cout << "Class Average: " << avg;
    return 0;
}
```

രണ്ട് വേരിയബിളുകൾക്ക് പ്രാരംഭ വില നൽകുന്നു

എക്സ്പ്ലിസിറ്റ് ടൈപ്പ് കൺവേർഷൻ

പ്രോഗ്രാം 7.14 ന്റെ ഒരു മാതൃക ഔട്ട്പുട്ട് താഴെ കൊടുത്തിരിക്കുന്നു.

```
How many students? 5
Enter the score of student 1: 45
Enter the score of student 2: 50
Enter the score of student 3: 52
Enter the score of student 4: 34
Enter the score of student 5: 55
Class Average: 47.2
```

പ്രോഗ്രാം 7.14-ൽ പ്രാരംഭ വില നൽകുന്ന പ്രസ്താവനയിൽ ഒരു കോമ ഉപയോഗിച്ച് വേർതിരിച്ച രണ്ട് പ്രയോഗങ്ങൾ (`i=1, sum=0`) അടങ്ങിയിരിക്കുന്നു. `i, sum` എന്നീ വേരിയബിളുകൾക്ക് അവയുടെ ആദ്യ വിലയായ `0,1` യഥാക്രമം കിട്ടുന്നു. `i<=n` എന്ന പരിശോധന പ്രയോഗം വിലയിരുത്തുകയും അത് ശരിയായതിനാൽ ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിക്കുകയും ചെയ്യുന്നു. ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിച്ചതിനുശേഷം പരിഷ്കരിക്കൽ പ്രസ്താവനയായ `++i` പ്രവർത്തിക്കുന്നു. വീണ്ടും `i<=n` എന്ന പരിശോധന പ്രയോഗം വിലയിരുത്തുകയും നിബന്ധന ശരിയായതിനാൽ ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിക്കുകയും ചെയ്യുന്നു. പരിശോധന പ്രയോഗം തെറ്റായ വില തിരിച്ചു തരുന്നതുവരെ ഈ പ്രക്രിയ തുടരുന്നു. മാതൃക ഔട്ട്പുട്ടിൽ ഇത് സംഭവിക്കുന്നത് `i`-യുടെ വില 6 ആകുമ്പോഴാണ്.

നമുക്ക് ചെയ്യാം

തന്നിരിക്കുന്ന സംഖ്യയുടെ ഗുണനപട്ടിക പ്രദർശിപ്പിക്കുവാനുള്ള ഒരു പ്രോഗ്രാം എഴുതുക. സംഖ്യ ഇൻപുട്ട് ചെയ്യുന്നത് n എന്ന വേരിയബിളിലാണെന്ന് കരുതുക. ലൂപ്പിന്റെ ചട്ടക്കൂട് താഴെ കൊടുത്തിരിക്കുന്നു.

```
cout<<i<<" x " <<n<<" = "<< i * n << "\n";
```

ഔട്ട്പുട്ട് കൂടി കാണിക്കുക.

for ലൂപ്പ് ഉപയോഗിക്കുമ്പോൾ ചില കാര്യങ്ങൾ ശ്രദ്ധിക്കേണ്ടതുണ്ട്. തന്നിരിക്കുന്ന നാലു കോഡ് ശകലങ്ങൾ ഈ പ്രത്യേക സാഹചര്യത്തിൽ വിശദീകരിക്കുന്നു. കോഡിൽ ഉപയോഗിച്ചിട്ടുള്ള എല്ലാ വേരിയബിളുകളും int ഡാറ്റ ഇനത്തിലുള്ളതാണ് എന്ന് കരുതുക.

കോഡ് ശകലം 1:

```
for (n=1; n<5; n++);
    cout<<n;
```

for പ്രസ്താവനയുടെ ബ്രായ്ക്കറ്റ് കഴിഞ്ഞ് ഒരു അർധവിരാമം കാണപ്പെടുന്നു. ഇത് വാക്യഘടനയിലെ തെറ്റ് (syntax error) അല്ല. ഇതിന്റെ ഔട്ട്പുട്ട് നിങ്ങൾക്ക് പ്രവചിക്കാൻ കഴിയുമോ? 5 ആണെങ്കിൽ നിങ്ങൾ പറഞ്ഞത് ശരിയാണ്. ഈ ലൂപ്പിന് ചട്ടക്കൂട് ഇല്ല. പക്ഷെ ഇതിന്റെ പ്രവർത്തനം സാധാരണപോലെ പൂർത്തീകരിക്കുന്നു. പ്രാരംഭവില നൽകുന്ന പ്രസ്താവന n ന് 1 എന്ന വില നൽകുകയും നിബന്ധന വിലയിരുത്തുമ്പോൾ ശരിയാവുകയും ചെയ്യുന്നു. അവിടെ ലൂപ്പ് ചട്ടക്കൂട് ഇല്ലാത്തതിനാൽ പരിഷ്കരിക്കൽ പ്രസ്താവന പ്രവർത്തിക്കുകയും n ന്റെ വില 5 ആകുന്നതുവരെ ഈ പ്രവർത്തനം തുടരുകയും ചെയ്യുന്നു. ഈ സന്ദർഭത്തിൽ നിബന്ധന വിലയിരുത്തി തെറ്റാവുകയും പ്രോഗ്രാമിന്റെ നിയന്ത്രണം ലൂപ്പിൽ നിന്നും പുറത്തു വരികയും ചെയ്യുന്നു. ഔട്ട്പുട്ട് പ്രസ്താവന പ്രവർത്തിക്കുമ്പോൾ സ്ക്രീനിൽ 5 എന്ന് പ്രദർശിപ്പിക്കുന്നു.

കോഡ് ശകലം 2:

```
for (n=1; n<5; )
    cout<<n;
```

ഈ കോഡിൽ പരിഷ്കരിക്കൽ പ്രസ്താവന (update expression) ഇല്ല. ഇത് കോഡിന്റെ വാക്യഘടനയിൽ തെറ്റ് ഉണ്ടാക്കുന്നില്ല. പക്ഷെ ലൂപ്പ് പ്രവർത്തിക്കുമ്പോൾ ഒരിക്കലും അവസാനിക്കുന്നില്ല. 1 എന്ന സംഖ്യ അനന്തമായി പ്രദർശിപ്പിക്കുന്നു. ഇതിനെ നമുക്ക് അനന്തമായ ലൂപ്പ് (Infinite loop) എന്നു വിളിക്കാം.

കോഡ് ശകലം 3:

```
for ( ; n<5; n++)
    cout<<n;
```

ഈ കോഡിന്റെ ഔട്ട്പുട്ട് പ്രവചിക്കുവാൻ സാധ്യമല്ല. കാരണം നിയന്ത്രണവേരിയബിളിന് (Control Variable) പ്രാരംഭ വില നൽകിയിട്ടില്ല. അതിനാൽ നിയന്ത്രണ വേരിയബിൾ n - ന് ചില പൂർണസംഖ്യകൾ കിട്ടുന്നു. ചിലപ്പോൾ അത് 5-നെക്കാൾ കുറവാണെങ്കിൽ നിബന്ധന (condition) തെറ്റാവുന്നതുവരെ ചട്ടക്കൂട് പ്രവർത്തിക്കും. n ന്റെ തനത് വില 5-ഓ അതിൽ കൂടുതലോ ആണെങ്കിൽ ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിക്കാതെ തന്നെ ലൂപ്പ് അവസാനിക്കുന്നു.

കോഡ് ശകലം 4:

```
for (n=1; ; n++)
      cout<<n;
```

മുകളിൽ കൊടുത്തിരിക്കുന്ന കോഡിൽ പരിശോധന പ്രയോഗം (test expression) നൽകിയിട്ടില്ല. ഇത്തരം ഘട്ടത്തിൽ പരിശോധനാ പ്രയോഗത്തിന്റെ ഫലം ശരിയായി എടുക്കുകയും ലൂപ്പ് അനന്തമായി മാറുകയും ചെയ്യുന്നു.

മുകളിൽ കൊടുത്തിരിക്കുന്ന നാലു കോഡ് ശകലങ്ങളും സൂചിപ്പിക്കുന്നത് **for** ലൂപ്പിലെ എല്ലാ ഘടകങ്ങളും നിർബന്ധമില്ല എന്നാണ്. എന്നാൽ **while, do...while** പ്രസ്താവനകളുടെ കാര്യം ഇങ്ങനെയല്ല. ഈ രണ്ടു ലൂപ്പുകൾക്കും പരിശോധന പ്രയോഗങ്ങൾ നിർബന്ധമാണ്. എന്നാൽ മറ്റു ഘടകങ്ങൾ നിർബന്ധമില്ല. എന്നാൽ ഔട്ട്പുട്ട് സംബന്ധിച്ച് ജാഗ്രത പുലർത്തണം.

മറ്റൊരു വസ്തുത ശ്രദ്ധിക്കേണ്ടത് പരിശോധന പ്രയോഗത്തിനു പകരമായി നമുക്കു ഒരു സംഖ്യ നൽകുവാൻ സാധിക്കുമെന്നതാണ്. ഈ സംഖ്യ പൂജ്യമാണെങ്കിൽ പരിശോധന പ്രയോഗം തെറ്റായായും അല്ലെങ്കിൽ ശരിയായും ലൂപ്പ് പരിഗണിക്കും.

നമുക്ക് പരിശോധിക്കാം

1. *1 നും 49 നും ഇടയ്ക്കും എല്ലാ ഇരട്ടസംഖ്യകളുടെയും തുകയും ശരാശരിയും കണ്ടുപിടിക്കാനുള്ള പ്രോഗ്രാം എഴുതുക.*
2. *3 കൊണ്ടും 5 കൊണ്ടും ഹരിക്കാവുന്ന 10 നും 50 നും ഇടയ്ക്കുള്ള സംഖ്യകൾ പ്രദർശിപ്പിക്കുവാനുള്ള പ്രോഗ്രാം എഴുതുക.*
3. *താഴെ കൊടുത്തിരിക്കുന്ന കോഡിന്റെ ഔട്ട്പുട്ട് പ്രവചിക്കുക.*

```
for (i=1; i<=10; ++i);
cout<<i+2;
```

7.2.3 do...while പ്രസ്താവന (do...while statement)

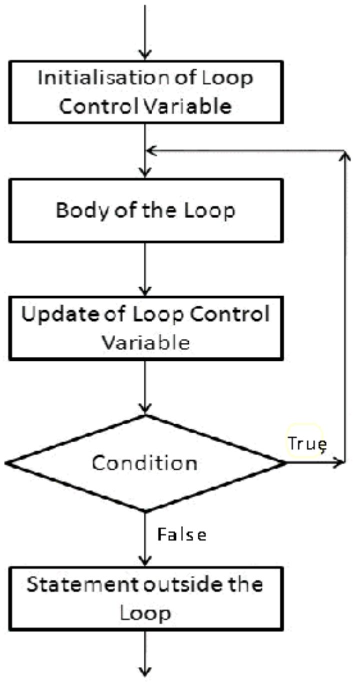
`for` ലൂപ്പിന്റെയും , `while` ലൂപ്പിന്റെയും കാര്യത്തിൽ ലൂപ്പ് ചട്ടക്കൂട് പ്രവർത്തിക്കുന്നതിന് മുമ്പ് പരിശോധന പ്രയോഗം വിലയിരുത്തുന്നു. ആദ്യ തവണ തന്നെ പരിശോധന പ്രയോഗം തെറ്റാണെങ്കിൽ ലൂപ്പ് പ്രവർത്തിക്കില്ല. എന്നാൽ ചില സാഹചര്യങ്ങളിൽ പരിശോധന പ്രയോഗത്തിന്റെ ഫലം പരിഗണിക്കാതെ തന്നെ ലൂപ്പിന്റെ ചട്ടക്കൂട് ഒരു പ്രാവശ്യമെങ്കിലും പ്രവർത്തിപ്പിക്കേണ്ടത് ആവശ്യമായി വരും. അത്തരം സാഹചര്യത്തിൽ **do...while** ലൂപ്പ് ഉപയോഗിക്കുന്നതാണ് നല്ലത്. `do...while` ലൂപ്പിന്റെ വാക്യഘടന (syntax) ഇതാണ്.

നിയന്ത്രണവേരിയബിളിന്റെ പ്രാരംഭ വില നൽകൽ;

```
do
{
ലൂപ്പിന്റെ ചട്ടക്കൂട്;
ലൂപ്പ് നിയന്ത്രണവേരിയബിളിന്റെ വില പുതുക്കൽ;
} while (പരിശോധന പ്രയോഗം) ;
```

```
initialisation of loop control variable;
do
{
      body of the loop;
      updation of loop control variable;
} while(test expression);
```

ചിത്രം 7.5-ൽ ഈ ലൂപ്പിന്റെ പ്രവർത്തന ക്രമം കാണിച്ചിരിക്കുന്നു. ഇവിടെ ലൂപ്പ് ചട്ടക്കൂട് പ്രവർത്തിച്ചതിനുശേഷം മാത്രമാണ് പരിശോധന പ്രസ്താവന വിലയിരുത്തുന്നത്. അതിനാൽ `do...while` ലൂപ്പ് ഒരു ബഹിർഗമന നിയന്ത്രണ ലൂപ്പ് (`Exit controlled loop`) ആകുന്നു. പരിശോധന പ്രയോഗം തെറ്റാണെങ്കിൽ ലൂപ്പിന്റെ പ്രവർത്തനം അവസാനിക്കുന്നു. ഇത് അർഥമാക്കുന്നത് പരിശോധന പ്രയോഗത്തിന്റെ ഫലം പരിഗണിക്കാതെ തന്നെ ലൂപ്പിന്റെ ചട്ടക്കൂട് ഒരു പ്രാവശ്യം പ്രവർത്തിക്കുന്നു എന്നാണ്.



ചിത്രം 7.5: *Execution of* `do..while` *loop*

`do...while` ലൂപ്പിന്റെ പ്രവർത്തനം വിശദീകരിക്കുന്നതിനായി താഴെ കൊടുത്തിരിക്കുന്ന പ്രോഗ്രാം ശകലം നമുക്ക് പരിശോധിക്കാം.

```
int k=1;
do
{
    cout << k << '\t';
    ++k;
} while(k<=3);
 cout << "\n Program Ends";
```

ലൂപ്പ് തുടങ്ങുന്നതിനുമുമ്പ് പ്രാരംഭ വില നൽകുന്നു.

ലൂപ്പിന്റെ ചട്ടക്കൂട്

ലൂപ്പിന്റെ ചട്ടക്കൂടിനകത്ത് വില പുതുക്കുന്നു

പരിശോധനാ പ്രയോഗം

ആദ്യം വേരിയബിൾ **k**-യുടെ വിലയായി 1 നൽകുന്നു. അതിനുശേഷം ലൂപ്പ് ചട്ടക്കൂട് പ്രവർത്തിക്കുകയും **k** യുടെ വിലയായ 1 എന്ന് പ്രദർശിപ്പിക്കുകയും ചെയ്യുന്നു. തുടർന്ന് **k**-യുടെ വില 1 വർദ്ധിപ്പിക്കുന്നു (ഇപ്പോൾ **k=2**). അതിനുശേഷം **k<=3** എന്ന വ്യവസ്ഥ പരിശോധിക്കുന്നു. ആ വ്യവസ്ഥ ശരിയായതിനാൽ ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിച്ച് **k**-യുടെ വില 2 എന്ന് സ്ക്രീനിൽ പ്രദർശിപ്പിക്കുന്നു. പുതുക്കൽ പ്രക്രിയ വീണ്ടും നടത്തി **k**-യുടെ വില 3 ആക്കുകയും **k<=3** എന്ന നിബന്ധന വീണ്ടും പരിശോധിക്കുകയും ചെയ്യുന്നു. നിബന്ധന ശരിയായതിനാൽ ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിപ്പിച്ച് **k**-യുടെ വിലയായ 3 പ്രദർശിപ്പിക്കുന്നു. **k**-യുടെ വില വീണ്ടും പരിഷ്കരിച്ച് 4 ആകുന്നു. ഇത് പ്രോഗ്രാമിന്റെ നിയന്ത്രണം ലൂപ്പിന് പുറത്ത് വരുന്നതിനും തുടർന്നുള്ള പ്രസ്താവന പ്രവർത്തിക്കുന്നതിനും കാരണമാകുന്നു. ആയതിനാൽ കോഡിന്റെ ഔട്ട്പുട്ട് ഇങ്ങനെയായിരിക്കും.

```
1	2	3
```

ഈ ലൂപ്പ് മറ്റു രണ്ടു ലൂപ്പിൽ നിന്നും എങ്ങനെ വ്യത്യാസപ്പെട്ടിരിക്കുന്നു എന്ന് ഇപ്പോൾ നമുക്കു നോക്കാം. **k**-യുടെ പ്രാരംഭവില 5 ആണെന്ന് സങ്കൽപിക്കുക. എന്ത് സംഭവിക്കും? ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിച്ച് **k**-യുടെ വിലയായ 5 സ്ക്രീനിൽ പ്രദർശിപ്പിക്കുന്നു. അതിനുശേഷം **k**-യുടെ വില ഒന്ന് വർദ്ധിപ്പിച്ച് 6 ആയി തീരുന്നു. **k<=3** എന്ന നിബന്ധന പരിശോധിച്ചപ്പോൾ പരിശോധന പ്രയോഗം തെറ്റാവുകയും നിയന്ത്രണം ലൂപ്പിന് പുറത്തേയ്ക്കു വരുകയും ചെയ്യുന്നു. `do...while` ലൂപ്പിന്റെ ചട്ടക്കൂടിലേക്ക് ആദ്യത്തെ പ്രാവശ്യം പ്രവേശിക്കുന്നതിന് യാതൊരു നിയന്ത്രണവും ഇല്ലെന്നാണ് ഇത് കാണിക്കുന്നത്. അതുകൊണ്ടു നിബന്ധനയുടെ ശരി (`True`) വില മാത്രം അനുസരിച്ചാണ് ലൂപ്പ് ചട്ടക്കൂട് പ്രവർത്തിക്കേണ്ടതെങ്കിൽ `while` ലൂപ്പോ, `for` ലൂപ്പോ ഉപയോഗിക്കുക.

ഉപയോക്താവിന്റെ ആവശ്യത്തിനനുസരിച്ച് പ്രവർത്തിക്കുന്ന ഒരു പ്രോഗ്രാം നമുക്കു നോക്കാം. ഇത്തരം പ്രോഗ്രാമുകൾ ഉപയോക്താവിന്റെ പ്രതികരണം സ്വീകരിച്ചുകൊണ്ട് കോഡ് ശകലം ആവർത്തിച്ചു പ്രവർത്തിപ്പിക്കുന്നു.

ഉപഭോക്താവിൽ നിന്നും ഓരോ ചതുരത്തിന്റേയും നീളവും വീതിയും ഇൻപുട്ടായി സ്വീകരിച്ച് ചതുരങ്ങളുടെ വിസ്തീർണം കണ്ട്പിടിക്കുന്നതിനുള്ള ഒരു പ്രോഗ്രാം `do...while` ലൂപ്പ് ഉപയോഗിച്ച് എഴുതിയിരിക്കുന്നു. (പ്രോഗ്രാം 7.15)

പ്രോഗ്രാം 7.15 ചതുരത്തിന്റെ വിസ്തീർണം കാണുന്നതിന്

```
#include <iostream>
using namespace std;
int main()
{
    float length, breadth, area;
    char ch;
    do
    {
        cout << "Enter length and breadth: ";
        cin >> length >> breadth;
        area = length * breadth;
        cout << "Area = " << area;
        cout << "Any more rectangle (Y/N)? ";
        cin >> ch;
    } while (ch == 'Y' || ch == 'y');
    return 0;
}
```

പ്രോഗ്രാം 7.15 ന്റെ ഒരു മാതൃക ഔട്ട്പുട്ട് താഴെ കൊടുക്കുന്നു.

```
Enter length and breadth: 3.5      7
      Area = 24.5
      Any more rectangle (Y/N)? Y
      Enter length and breadth: 6      4.5
      Area = 27
```

ഉപയോക്താവ് ഇൻപുട്ട് നൽകുന്നു

ഉപയോക്താവ് ഇൻപുട്ട് നൽകുന്നു

```
Any more rectangle (Y/N)? N
```

ഉപയോക്താവ് ഇൻപുട്ട് നൽകുന്നു

C++ ലെ മൂന്ന് ലൂപ്പിങ്ങ് പ്രസ്താവനകളെക്കുറിച്ചും നാം ചർച്ച ചെയ്തു. പട്ടിക 7.2 ൽ ഈ പ്രസ്താവനകൾ താരതമ്യം ചെയ്തിരിക്കുന്നു.

for ലൂപ്പ്	while ലൂപ്പ്	do...while ലൂപ്പ്
ആഗമന നിയന്ത്രണ ലൂപ് (Entry controlled loop)	ആഗമന നിയന്ത്രണ ലൂപ് (Entry controlled loop)	ബഹിർഗമന നിയന്ത്രണ ലൂപ് (Extit controlled loop)
ലൂപ്പിന്റെ നിർവചനത്തോടൊപ്പം തന്നെ പ്രാരംഭ വിലയും നൽകുന്നു.	ലൂപ് നിർവചനത്തിനു മുമ്പ് പ്രാരംഭവില നൽകുന്നു.	ലൂപ് നിർവചനത്തിനു മുമ്പ് പ്രാരംഭവില നൽകുന്നു.
ലൂപ്പിന്റെ ചട്ടക്കൂട് ഒരു പ്രാവശ്യമെങ്കിലും പ്രവർത്തിക്കുമെന്ന് ഉറപ്പില്ല	ലൂപ്പിന്റെ ചട്ടക്കൂട് രു പ്രാവശ്യമെങ്കിലും പ്രവർത്തിക്കുമെന്ന് ഉറപ്പില്ല.	നിബന്ധന തെറ്റാണെങ്കിലും ലൂപ്പിന്റെ ചട്ടക്കൂട് ഒരു പ്രാവശ്യം പ്രവർത്തിക്കും.

പട്ടിക 7.2: C++ ലൂപ് പ്രസ്താവനകളുടെ താരതമ്യം

7.2.4 ലൂപ്പുകളുടെ നെസ്റ്റിങ്ങ് (Nesting of loops)

ഒരു ലൂപ്പിനകത്ത് മറ്റൊരു ലൂപ്പ് ഉൾപ്പെടുത്തുന്നതിനെ ലൂപ്പുകളുടെ നെസ്റ്റിങ്ങ് എന്നു പറയുന്നു. രണ്ട് ലൂപ്പുകൾ നാം നെസ്റ്റ് ചെയ്യുമ്പോൾ പുറത്തുള്ള ലൂപ്പ് (Outer loop) അകത്തുള്ള ലൂപ്പ് എത്ര തവണ പ്രവർത്തിച്ചു എന്ന് തിട്ടപ്പെടുത്തുന്നു. ഇവിടെ രണ്ടു ലൂപ്പുകളുടെയും ലൂപ്പ് നിയന്ത്രണ വേരിയബിളുകൾ (Loop control variable) വ്യത്യസ്തമായിരിക്കണം.

നെസ്റ്റഡ് ലൂപ്പ് എങ്ങനെ പ്രവർത്തിക്കുന്നു എന്ന് നമുക്കു നോക്കാം. ഒരു ക്ലോക്കിലെ മിനുട്ട് സൂചിയുടെയും, സെക്കന്റ് സൂചിയുടെയും കാര്യം എടുക്കുക. നിങ്ങൾ ക്ലോക്കിന്റെ പ്രവർത്തനം ശ്രദ്ധിച്ചിട്ടുണ്ടോ? മിനുട്ട് സൂചി ഏതെങ്കിലും ഒരു സ്ഥാനത്ത് നിൽക്കുമ്പോൾ സെക്കന്റ് സൂചി ഒരു ഭ്രമണം പൂർത്തിയാക്കുന്നു (1 മുതൽ 60 വരെ). സെക്കന്റ് സൂചി ഒരു ഭ്രമണം പൂർത്തിയാക്കിയതിനു ശേഷം മിനുട്ട് സൂചി അടുത്ത സ്ഥാനത്തേക്ക് മാറുന്നു. മിനുട്ട് സൂചിയുടെ ഓരോ സ്ഥാനത്തിനും അനുസൃതമായി സെക്കന്റ് സൂചി കറക്കം പൂർത്തിയാക്കുന്നു. ഈ പ്രക്രിയ തുടർന്നു കൊണ്ടേയിരിക്കുന്നു. ഇവിടെ സെക്കന്റ് സൂചിയുടെ ചലനം ഉള്ളിലെ ലൂപ്പിന്റെ പ്രവർത്തനമായും മിനുട്ട് സൂചിയുടെ ചലനം ബാഹ്യലൂപ്പിന്റെ പ്രവർത്തനമായും കരുതാവുന്നതാണ്. C++ ലെ എല്ലാ ലൂപ്പുകളും നെസ്റ്റിങ്ങ് അനുവദിക്കുന്നു. താഴെ കൊടുത്തിരിക്കുന്ന ഉദാഹരണം for ലൂപ്പിന്റെ നെസ്റ്റിങ്ങ് പ്രവർത്തനം കാണിച്ചുതരുന്നു.

```
for( i=1; i<=2; ++i)        // ബാഹ്യ ലൂപ്
{
     for(j=1; j<=3; ++j)    // ആന്തരിക ലൂപ്
     {
          cout<< "\n" << i << " and " << j;
     }
}
```

ബാഹ്യലൂപ്പിലെ വേരിയബിളായ `i` ക്ക് പ്രാരംഭ വിലയായി 1 നൽകുന്നു. അതിന്റെ പരിശോധന പ്രയോഗം വിലയിരുത്തി ശരിയായതിനാൽ ലൂപ്പിന്റെ ചട്ടക്കൂട് പ്രവർത്തിക്കുന്നു. ചട്ടക്കൂടിൽ അടങ്ങിയിരിക്കുന്നത് നിയന്ത്രണ വേരിയബിൾ `j` യോടുകൂടിയ ആന്തരിക ലൂപ്പാണ്. `j` ക്ക് പ്രാരംഭ വിലയായ 1 നൽകി അതിന്റെ പ്രവർത്തനം ആരംഭിക്കുന്നു. `j` =1, `j`=2, `j`=3 ആയി ആന്തരിക ലൂപ്പ് 3 തവണ പ്രവർത്തിക്കുന്നു. ഓരോ തവണയും `j<=3` എന്ന പരിശോധന പ്രയോഗം വിലയിരുത്തുകയും ശരിയായതിനാൽ ഔട്ട്പുട്ട് പ്രദർശിപ്പിക്കുകയും ചെയ്യുന്നു.

```
1 and 1
1 and 2
1 and 3
```

ആദ്യത്തെ 1, i യുടെ വിലയും രണ്ടാമത്തെ 1, j യുടെ വിലയുമാണ്.

പരിശോധന പ്രയോഗം `j<=3` തെറ്റാവുമ്പോൾ പ്രോഗ്രാമിന്റെ നിയന്ത്രണം ആന്തരിക ലൂപ്പിൽ നിന്നും പുറത്തു കടക്കുന്നു. ഇപ്പോൾ ബാഹ്യ ലൂപ്പിന്റെ പുതുക്കൽ പ്രസ്താവന പ്രവർത്തിച്ച് `i`=2 ആക്കുന്നു. പരിശോധന പ്രയോഗമായ `i<=2` പരിശോധിച്ച് ശരിയായതിനാൽ ലൂപ്പിന്റെ ചട്ടക്കൂട് ഒന്നുകൂടി പ്രവർത്തിക്കുന്നു. `j` =1, `j`=2, `j`=3 ആയി ആന്തരിക ലൂപ്പ് വീണ്ടും മൂന്നു തവണ പ്രവർത്തിച്ച് ഔട്ട് പുട്ട് പ്രദർശിപ്പിക്കുന്നു.

```
2 and 1
2 and 2
2 and 3
```

ആന്തരിക ലൂപ്പിന്റെ പ്രവർത്തനം പൂർത്തിയാക്കിയതിനുശേഷം നിയന്ത്രണം പുറത്തെ ലൂപ്പിന്റെ വില പുതുക്കൽ പ്രയോഗത്തിൽ തിരിച്ചെത്തുന്നു. `i` യുടെ വില 1 വെച്ച് വർദ്ധിപ്പിക്കുന്നു. (ഇപ്പോൾ `i`=3) പരിശോധന പ്രയോഗം `i<=2` വിലയിരുത്തുമ്പോൾ തെറ്റാവുന്നു. ആയതിനാൽ ലൂപ്പ് അതിന്റെ പ്രവർത്തനം അവസാനിപ്പിക്കുന്നു. പട്ടിക 7.3 മുകളിൽ കൊടുത്ത പ്രോഗ്രാം ശകലത്തിന്റെ പ്രവർത്തനം വിവരിക്കുന്നു.

ആവർത്തനം	ബാഹ്യ ലൂപ്പ്	ആന്തരികലൂപ്പ്	ഔട്ട്പുട്ട്
1	1	1	1 and 1
2	1	2	1 and 2
3	1	3	1 and 3
4	2	1	2 and 1
5	2	2	2 and 2
6	2	3	2 and 3

പട്ടിക 7.3: നെസ്റ്റഡ് ലൂപ്പിന്റെ പ്രവർത്തനം

നെസ്റ്റഡ് ലൂപ്പുകളിൽ പ്രവർത്തിക്കുന്ന സമയത്ത് ബാഹ്യലൂപ്പിലെ നിയന്ത്രണ വേരിയബിളുകളിൽ അവയുടെ വിലയിൽ മാറ്റം വരുന്നത് ആന്തരികലൂപ്പ് പൂർത്തികരിച്ചതിനുശേഷം മാത്രമാണ്.

ഇനി താഴെ കൊടുത്തിരിക്കുന്ന രീതിയിലുള്ള ത്രികോണം പ്രദർശിപ്പിക്കാനുള്ള ഒരു പ്രോഗ്രാം നമുക്കു എഴുതാം.

```
*
* *
* * *
* * * *
* * * * *
```

പ്രോഗ്രാം 7.16: ത്രികോണാകൃതിയിൽ നക്ഷത്രചിഹ്നം പ്രദർശിപ്പിക്കുന്നതിന്.

```
#include<iostream>
using namespace std;
int main()
{   int i, j;
    char ch = '*';
    for(i=1; i<=5; ++i)          //ബാഹ്യ ലൂപ്പ്
    {
        cout<< "\n" ;
        for(j=1; j<=i; ++j)      // ആന്തരിക ലൂപ്പ്
            cout<<ch;
    }
    return 0;
}
```

നമുക്ക് ചെയ്യാം

1. താഴെ കൊടുത്തിരിക്കുന്ന പ്രോഗ്രാം ശകലത്തിന്റെ ഔട്ട്പുട്ട് പ്രവചിക്കുക.

```
sum = 0;
for( i=1; i<3; ++i)
{
    for(j=1; j<3; ++j)
    {
        sum = sum + i * j;
    }
 cout<<sum ;
```

2. താഴെ കൊടുത്തിരിക്കുന്ന ത്രികോണങ്ങൾ പ്രദർശിപ്പിക്കുന്നതിനുള്ള പ്രോഗ്രാം എഴുതുക.

```
1                1
2  2             1  2
3  3  3          1  2  3
4  4  4          1  2  3  4
5  5  5  5       1  2  3  4  5
```

7.2.5 നിയന്ത്രണ പ്രസ്താവനകളുടെ നെസ്റ്റിങ്ങ് (Nesting of Control Statements)

നാം ലൂപ്പുകളുടെയും പ്രസ്താവനകളുടെയും നെസ്റ്റിങ്ങിനെകുറിച്ച് ചർച്ച ചെയ്തു. നിയന്ത്രണ പ്രസ്താവന മറ്റൊരു നിയന്ത്രണ പ്രസ്താവന ഉപയോഗിച്ച് നെസ്റ്റ് ചെയ്യാം. ഒരു ലൂപ്പിൽ തിരഞ്ഞെടുക്കൽ പ്രസ്താവനകളായ `if`, `switch` എന്നിവ അടങ്ങിയിരിക്കാം. അതുപോലെ തിരഞ്ഞെടുക്കൽ പ്രസ്താവനകളിൽ ലൂപ്പ് പ്രസ്താവനകളായ `while`, `for`, `do...while` എന്നിവയും അടങ്ങിയിരിക്കാം. പ്രോഗ്രാം 7.17 ൽ ലൂപ്പും അതിന്റെ ചട്ടക്കൂടും ഒരു `switch` പ്രസ്താവനയും ഉൾപ്പെട്ടിരിക്കുന്നു. ഇത് സാധാരണയായിട്ടുള്ള ഒരു മെനു നിയന്ത്രിത ശൈലിയുള്ള പ്രോഗ്രാമാണ്.

പ്രോഗ്രാം 7.17 രണ്ട് സംഖ്യകൾ സ്വീകരിച്ച് ഉപയോക്താവിന്റെ താല്പര്യത്തിന് അടിസ്ഥാനമായി ഗണിത ക്രിയകൾ ചെയ്യൽ.

```
#include<iostream>
using namespace std;
int main()
  {
    char ch;
    float n1, n2;
    cout<<"Enter two numbers: ";
    cin>>n1>>n2;
    do
    {
          cout<<"\nNumber 1: "<<n1<<"\tNumber 2: "<<n2;
          cout<<"\n\t\tOperator Menu";
          cout<<"\n\t1. Addition (+)";
          cout<<"\n\t2. Subtraction (-)";
          cout<<"\n\t3. Multiplication (*)";
          cout<<"\n\t4. Division (/)";
          cout<<"\n\t5. Exit (E)";
          cout<<"\nEnter Option number or operator: ";
          cin>>ch;
          switch(ch)
          {
              case '1' :
              case '+' : cout<<n1<<" + "<<n2<<" = "<<n1+n2;
                         break;
              case '2' :
              case '-' : cout<<n1<<" - "<<n2<<" = "<<n1-n2;
                         break;
              case '3' :
              case '*' : cout<<n1<<" * "<<n2<<" = "<<n1*n2;
                         break;
```

```
            case '4' :
            case '/' : cout<<n1<<" / "<<n2<<" = "<<n1/n2;
                       break;
            case '5' :
            case 'E' :
            case 'e' : cout<<"Thank You for using the program";
                       break;
            default  : cout<<"Invalid Choice!!";
        }
    } while (ch!='5' && ch!='E' && ch!='e');
    return 0;
}
```

പ്രോഗ്രാം 7.17 ന്റെ മാതൃക ഔട്ട്പുട്ട് താഴെ കൊടുക്കുന്നു:

```
Enter two numbers: 25   4
Number 1: 25          Number 2: 4
            Operator Menu
     1. Addition (+)
     2. Subtraction (-)
     3. Multiplication (*)
     4. Division (/)
     5. Exit (E)
Enter Option number or operator: 1
25 + 4 = 29
```

ഉപയോക്താവ് നൽകുന്ന ഇൻപുട്ട്

```
Number 1: 25          Number 2: 4
            Operator Menu
     1. Addition (+)
     2. Subtraction (-)
     3. Multiplication (*)
     4. Division (/)
     5. Exit (E)
Enter Option number or operator: /
25 / 4 = 6.25
```

ഉപയോക്താവ് നൽകുന്ന ഇൻപുട്ട്

```
Number 1: 25          Number 2: 4
            Operator Menu
     1. Addition (+)
     2. Subtraction (-)
     3. Multiplication (*)
```

```
        4. Division (/)
        5. Exit (E)
    Enter Option number or operator: 5
    Thank You for using the program
```

ഉപയോക്താവ് നൽകുന്ന ഇൻപുട്ട്

കൺട്രോൾ പ്രസ്താവനകളുടെ നെസ്റ്റിങ്ങിന്റെ വിവിധ സംയോഗങ്ങൾ ഉപയോഗിക്കുന്ന കൂടുതൽ പ്രോഗ്രാമുകൾ പ്രോഗ്രാം ഗ്യാലറി വിഭാഗത്തിൽ നാം ചർച്ച ചെയ്യും.

7.3 ജമ്പ് പ്രസ്താവനകൾ (Jump Statements)

പ്രോഗ്രാമിന്റെ നിയന്ത്രണം ഒരു ഭാഗത്തുനിന്നും മറ്റൊരു ഭാഗത്തേക്ക് മാറ്റാൻ ഉപയോഗിക്കുന്ന പ്രസ്താവനകളെ ജമ്പ് പ്രസ്താവനകൾ (Jump statements) എന്നു പറയുന്നു. C++ ൽ പ്രത്യേക നിബന്ധനകളില്ലാതെ പ്രവർത്തിക്കുന്ന നാലുതരം ജമ്പ് പ്രസ്താവനകൾ ഉണ്ട്. അവ **return**, **goto**, **break**, **continue** എന്നിവയാണ് ഇതിനുപുറമെ, C++ ലെ **exit()** എന്ന സ്റ്റാൻഡേർഡ് ലൈബ്രറി ഫങ്ഷൻ പ്രോഗ്രാമിന്റെ പ്രവർത്തനം അവസാനിപ്പിക്കുന്നതിനും ഉപയോഗിക്കുന്നുണ്ട്.

return പ്രസ്താവന ഫങ്ഷനിൽ നിന്ന് പുറത്ത് വരുന്നതിനും നിയന്ത്രണം, വിളിച്ച പ്രോഗ്രാമിലേക്ക് തിരിച്ചു കൊണ്ടു പോകുന്നതിനും ഉപയോഗിക്കുന്നു. അധ്യായം 10 ൽ ഇതിനെക്കുറിച്ച് പിന്നീട് വിശദീകരിച്ചിട്ടുണ്ട്. ഇനി നമുക്ക് മറ്റു ജമ്പ് പ്രസ്താവനകളെക്കുറിച്ച് ചർച്ച ചെയ്യാം.

7.3.1 goto പ്രസ്താവന

goto പ്രസ്താവന ഉപയോഗിച്ച് പ്രോഗ്രാം നിയന്ത്രണത്തെ ഫങ്ഷനിലെ ഏതു സ്ഥലത്തേക്കും മാറ്റാൻ സാധിക്കും. ഒരു **goto** പ്രസ്താവനയുടെ ലക്ഷ്യസ്ഥാനം ലേബൽ (ഒരു ഐഡന്റിഫയർ) ഉപയോഗിച്ച് അടയാളപ്പെടുത്തുന്നു.

goto പ്രസ്താവനയുടെ വാക്യഘടന താഴെ കൊടുക്കുന്നു.

```
        goto ലേബൽ ;
        ............;
        ............;
  ലേബൽ: ..........;
        ............;


        goto label;
        ............;
        ............;
  label: ..........;
        ............;
```

goto പ്രസ്താവനക്ക് മുമ്പോ പിൻപോ ഒരു പ്രോഗ്രാമിൽ കാണപ്പെടുന്നു. ലേബലിനുശേഷം ഒരു അപൂർണവിരാമം (**:**) ചിഹ്നം ആവശ്യമാണ്. ഉദാഹരണത്തിന് 1 മുതൽ 50 വരെ പ്രിന്റ് ചെയ്യാനുള്ള കോഡ് ശകലം പരിഗണിക്കുക.

```
        int i=1;
start:                          (ലേബൽ)
        cout<<i;
        ++i;
        if (i<=50)
              goto start;
```

ഇവിടെ cout, പ്രസ്താവന 1 എന്ന വില പ്രിന്റ് ചെയ്യുന്നു. അതിനുശേഷം i യുടെ വില 1 വർദ്ധിപ്പിക്കുന്നു. (ഇപ്പോൾ i=2), ഇപ്പോൾ പരിശോധന പ്രയോഗം i<=50 വിലയിരുത്തുന്നു. നിബന്ധന ശരിയായതിനാൽ start എന്ന ലേബലിലേക്ക് പ്രോഗ്രാം നിയന്ത്രണത്തെ മാറ്റുന്നു. നിബന്ധന തെറ്റാവുമ്പോൾ പ്രവർത്തനം അവസാനിപ്പിച്ച് പ്രോഗ്രാം നിയന്ത്രണം if പ്രസ്താവനക്കു ശേഷം എത്തുന്നു.

നമുക്കു മറ്റൊരു ഉദാഹരണം നോക്കാം. ഇവിടെ ഒരു സംഖ്യ സ്വീകരിക്കുകയും മുൻകൂട്ടി നിശ്ചയിച്ച വിലയുമായി താരതമ്യം ചെയ്യുകയും ചെയ്യുന്നു. തുല്യമാണെങ്കിൽ പ്രോഗ്രാം തുടരും അല്ലെങ്കിൽ അത് അവസാനിക്കുന്നു.

```
  int p;
  cout<<"Enter the Code: ";
  cin>>p;
  if(p!=7755)
     goto end;
  cout<<"Enter the details";
  ........................;
  ........................;
end:                              (ലേബൽ)
  cout<<"Sorry, the code number is wrong. Try again!";
```

ഇവിടെ ഉപയോക്താവ് ഇൻപുട്ടിന്റെ സാധ്യത പരിശോധിക്കുന്നു. കോഡ് സാധുവായതാണെങ്കിൽ പ്രോഗ്രാം മറ്റു വിശദാംശങ്ങൾ സ്വീകരിക്കുന്നു. ഇല്ലെങ്കിൽ നിയന്ത്രണം end എന്ന ലേബലിലേക്ക് പോകുന്നു. സ്ട്രക്ചേർഡ് പ്രോഗ്രാമിങ് goto ന്റെ ഉപയോഗം പ്രോത്സാഹിപ്പിക്കുന്നില്ല

7.3.2 break (ബ്രേക്ക്) പ്രസ്താവന

ഒരു പ്രോഗ്രാമിൽ break പ്രസ്താവന കാണപ്പെട്ടാൽ പ്രോഗ്രാമിന്റെ നിയന്ത്രണം തൊട്ടടുത്ത ലൂപ്പിനോ (for, while, do...while), switch പ്രസ്താവനയ്ക്കോ പുറത്തേക്ക് മാറ്റുന്നു. കൺട്രോൾ ചട്ടക്കൂടിന് ശേഷമുള്ള പ്രസ്താവന മുതൽ പ്രവർത്തനം തുടരുന്നു. switch പ്രസ്താവനയിൽ break ന്റെ പ്രവാഹത്തെ കുറിച്ച് നാം ഇതിനോടകം ചർച്ച ചെയ്തു കഴിഞ്ഞു. ഇത് ലൂപ്പുകളുടെ പ്രവർത്തനത്തെ എങ്ങനെ സ്വാധീനിക്കുന്നു എന്ന് നമുക്ക് നോക്കാം. താഴെ കൊടുത്തിരിക്കുന്ന രണ്ട് പ്രോഗ്രാം ശകലങ്ങൾ പരിഗണിക്കുക.

കോഡ് ശകലം 1

```
i=1;
while(i<=10)
{
     cin>>num;
     if (num==0)
          break;
     cout<<"Entered number is: "<<num;
     cout<<"\nInside the loop";
     ++i;
}
cout<<"\nComes out of the loop";
```

മുകളിലെ പ്രോഗ്രാം 10 സംഖ്യകളെ ഇൻപുട്ട് ചെയ്യാൻ അനുവദിക്കുന്നു. ഇൻപുട്ട് ചെയ്യുമ്പോൾ ഏതെങ്കിലും ഒരു സംഖ്യ 0 ആണെങ്കിൽ ലൂപ്പ് ചട്ടക്കൂടിലെ ബാക്കി പ്രസ്താവനകളെ ഒഴിവാക്കി പ്രോഗ്രാമിന്റെ നിയന്ത്രണം ലൂപ്പിനു പുറത്ത് വരികയും "Comes out of the loop" എന്ന സന്ദേശം സ്ക്രീനിൽ പ്രദർശിപ്പിക്കുകയും ചെയ്യുന്നു. ഒരു നെസ്റ്റഡ് ലൂപ്പിൽ break പ്രസ്താവനളപയോഗിക്കുന്ന മറ്റൊരു കോഡ് ശകലം നമുക്കു പരിഗണിക്കാം.

കോഡ് ശകലം 2

```
for(i=1; i<=5; ++i)    //outer loop
{
     cout<<"\n";
     for(j=1; j<=i; ++j)     //inner loop
     {
          cout<<"* ";
          if (j==3)
               break;
     }
}
```

ഈ കോഡ് ശകലം താഴെ കൊടുത്തിരിക്കുന്ന മാതൃക പ്രദർശിപ്പിക്കുന്നു.

```
*
*  *
*  *  *
*  *  *
*  *  *
```

j യുടെ വില എപ്പോൾ മൂന്ന് ആകുന്നുവോ അപ്പോൾ ആന്തരിക ലൂപ്പിന്റെ പ്രവർത്തനം അവസാനിക്കുന്നു.

നെസ്റ്റഡ് ലൂപ്പ് സാധാരണയായി i=1, i=2, i=3 എന്നീ വിലകൾക്ക് അനുസരിച്ച് പ്രവർത്തിക്കുന്നു. i യുടെ ഓരോ വിലക്കനുസരിച്ച് j, 1 മുതൽ i വരെയുള്ള വിലകൾ സ്വീകരിക്കും. i യുടെ വില 4 ഓ 5 ഓ ആകുമ്പോൾ ഉള്ളിലുള്ള ലൂപ്പ് j = 1, j=2, j=3 എന്നീ വിലകൾക്കനുസരിച്ച് പ്രവർത്തിച്ച് break നു ശേഷം ലൂപ്പിൽ നിന്നും പുറത്ത് പോകുന്നു.

7.3.3 continue (കണ്ടിന്യൂ) പ്രസ്താവന

continue പ്രസ്താവന മറ്റൊരു ജംമ്പ് പ്രസ്താവനയാണ് അത് ലൂപ്പ് ചട്ടക്കൂടിന്റെ ഒരു ഭാഗം ഒഴിവാക്കി അടുത്ത ആവർത്തനത്തിലേക്ക് എത്തിക്കുന്നതിനു വേണ്ടി ഉപയോഗിക്കുന്നു. break പ്രസ്താവന ലൂപ്പിന്റെ പ്രവർത്തനം നിർത്തി വെയ്ക്കുമ്പോൾ continue പ്രസ്താവന ചില ഭാഗങ്ങൾ ഒഴിവാക്കി അടുത്ത ആവർത്തനം നടത്താൻ നിർബന്ധിക്കുന്നു. താഴെ കൊടുത്തിരിക്കുന്ന പ്രോഗ്രാം ശകലം continue പ്രസ്താവനയുടെ പ്രവർത്തനം വിവരിക്കുന്നു.

```
for (i=1; i<=10; ++i)
    {
        if (i==6)
            continue;
        cout<<i<<"\t";
    }
```

ഈ കോഡ് താഴെ പറയുന്ന ഔട്ട്പുട്ട് തരുന്നു.

```
1    2    3    4    5    7    8    9    10
```

6 ലിസ്റ്റിൽ ഇല്ല എന്നത് ശ്രദ്ധിക്കുക. i യുടെ വില 6 ആകുമ്പോഴാണ് continue പ്രസ്താവന പ്രവർത്തിക്കുന്നത്. അതിന്റെ ഫലമായി ഔട്ട്പുട്ട് പ്രസ്താവന ഒഴിവാക്കി പ്രോഗ്രാം നിയന്ത്രണം അടുത്ത ആവർത്തനത്തിലെ പുതുക്കൽ പ്രസ്താവനയിൽ എത്തിച്ചേരുന്നു.

ഒരു ലൂപ്പിനകത്തെ break പ്രസ്താവന ലൂപ്പിനെ അവസാനിപ്പിക്കുകയും ലൂപ്പിനു ശേഷമുള്ള പ്രസ്താവനകളിലേക്ക് പ്രോഗ്രാം നിയന്ത്രണത്തെ എത്തിക്കുകയും ചെയ്യുന്നു. continue പ്രസ്താവന നിലവിലുള്ള ആവർത്തനത്തിലെ ശേഷിച്ച ഭാഗം ഉപേക്ഷിച്ച് ലൂപ്പിന്റെ അടുത്ത ആവർത്തനം ആരംഭിക്കുന്നു. While ലൂപ്പിലും, do...while ലൂപ്പിലും continue പ്രസ്താവന ഉപയോഗിക്കുമ്പോൾ ലൂപ്പ് അനന്തമാകുന്നത് ഒഴിവാക്കണം എന്നത് ശ്രദ്ധിക്കണം. പട്ടിക 7.4 break, continue എന്നീ പ്രസ്താവനകൾ തമ്മിലുള്ള താരതമ്യം കാണിക്കുന്നു.

break പ്രസ്താവന	continue പ്രസ്താവന
switch ന്റെയും ലൂപ്പിന്റെയും കൂടെ ഉപയോഗിക്കാം.	ലൂപ്പിന്റെ കൂടെ മാത്രം ഉപയോഗിക്കുന്നു.
ബ്ലോക്കിലെ അവശേഷിക്കുന്ന പ്രസ്താവനകൾ ഒഴിവാക്കി പ്രോഗ്രാമിന്റെ നിയന്ത്രണം സ്വിച്ചിനോ ലൂപ്പിനോ പുറത്തേക്കു കൊണ്ടു വരുന്നു.	ബ്ലോക്കിലെ അവശേഷിക്കുന്ന പ്രസ്താവനകൾ ഒഴിവാക്കി പ്രോഗ്രാമിന്റെ നിയന്ത്രണം ലൂപ്പിന്റെ ആരംഭത്തിലേക്ക് കൊണ്ടു വരുന്നു.
പരിശോധന പ്രസ്താവന ശരിയാണങ്കിലും പ്രോഗ്രാമിന്റെ നിയന്ത്രണം ലൂപ്പിനു പുറത്തു പോകുന്നു.	പരിശോധന പ്രസ്താവനയുടെ വില തെറ്റാകുമ്പോൾ മാത്രം പ്രോഗ്രാമിന്റെ നിയന്ത്രണം ലൂപ്പിന് പുറത്തു കൊണ്ടു പോകുന്നു.

പട്ടിക 7.4 break, continue എന്നീ പ്രസ്താവനകൾ തമ്മിലുള്ള താരതമ്യം

ഒരു പ്രോഗ്രാം അതിന്റെ തന്നെ പ്രവർത്തനം നിർത്തുന്നതിന് **exit()** എന്ന ഒരു ബിൽട്ട്-ഇൻ ഫങ്ഷൻ C++ ൽ ഉപയോഗിക്കുന്നു. **cstdlib** എന്ന ഹെഡർ ഫയൽ (ടർബോ C++ ൽ process .h) പ്രോഗ്രാമിൽ ഉൾപ്പെടുത്തിയാൽ മാത്രമേ. **exit ()** എന്ന ഫങ്ഷൻ ഉപയോഗിക്കാൻ കഴിയും. പ്രോഗ്രാം 7.18 ഈ ഫങ്ഷന്റെ പ്രവർത്തനം വിശദീകരിക്കുന്നു.

പ്രോഗ്രാം 7.18: തന്നിരിക്കുന്ന സംഖ്യ അഭാജ്യ സംഖ്യയാണോ അല്ലയോ എന്ന് പരിശോധിക്കുന്നതിന്.

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
    int i, num;
    cout<<"Enter the number: ";
    cin>>num;
    for(i=2; i<=num/2; ++i)
    {
        if(num%i == 0)
        {
            cout<<"Not a Prime Number";
            exit(0);
        }
    }
    cout<<"Prime Number";
    return 0;
}
```

പ്രോഗ്രാം 7.18 ലെ `for` ലൂപ്പിലെ പരിശോധന പ്രസ്താവന `i<=sqrt(num)` ഉപയോഗിച്ച് മാറ്റി എഴുതാം. ഇവിടെ `sqrt()` എന്നത് തന്നിരിക്കുന്ന സംഖ്യയുടെ വർഗ്ഗമൂലം കണ്ടുപിടിക്കുന്നതിനുള്ള ഒരു ഫങ്ഷനാണ്. ഒരു സംഖ്യക്ക് 2 മുതൽ അതിന്റെ വർഗ്ഗമൂലം വരെ ഘടകങ്ങൾ ഇല്ലെങ്കിൽ അത് ഒരു അവിഭാജ്യ (prime) സംഖ്യയാണ്. `sqrt()` ഉപയോഗിക്കുന്നതിന് `#include<cmath>` എന്ന പ്രസ്താവന ഉൾപ്പെടുത്തണം.

പ്രോഗ്രാം 7.18 ന്റെ മാതൃക ഔട്ട്പുട്ടുകൾ താഴെ കാണിച്ചിരിക്കുന്നു.

ഔട്ട്പുട്ട് 1

```
Enter the number: 17
Prime Number
```

ഔട്ട്പുട്ട് 2

```
Enter the number: 18
Not a Prime Number
```

നമുക്ക് പരിശോധിക്കാം

1. `goto` പ്രസ്താവന താഴെ പറയുന്ന ഏതിലേക്ക് നിയന്ത്രണം പോകുന്നതിന് കാരണമാകുന്നു.
   (a) ഒരു ഓപ്പറേറ്റർ (b) ഒരു ലേബൽ (c) ഒരു വേരിയബിൾ
   (d) ഒരു ഫങ്ഷൻ
2. break പ്രസ്താവന ഏതിൽ നിന്ന് പുറത്തുപോകാൻ കാരണമാകുന്നു.
   (a) ഏറ്റവും അകത്തുള്ള ലൂപ്പിൽ നിന്ന് മാത്രം
   (b) ഏറ്റവും ഉള്ളിലുള്ള switch ൽ നിന്ന് മാത്രം
   (c) എല്ലാ ലൂപ്പിൽ നിന്നും, switch ൽ നിന്നും
   (d) ഏറ്റവും അകത്തുള്ള ലൂപ്പിൽ നിന്നോ സ്വിച്ചിൽ നിന്നോ
3. exit( ) ഫങ്ഷൻ ഏതിൽ നിന്ന് പുറത്തേക്ക് എത്തിക്കുന്നു.
   (a) അത് കാണപ്പെടുന്ന ഫങ്ഷനിൽ നിന്നും
   (b) അത് കാണപ്പെടുന്ന ലൂപ്പിൽ നിന്നും
   (c) അത് കാണപ്പെടുന്ന ബ്ലോക്കിൽ നിന്നും
   (d) അത് കാണപ്പെടുന്ന പ്രോഗ്രാമിൽ നിന്നും
4. exit( ) ഫങ്ഷൻ ഉപയോഗിക്കുന്നതിന് ഉൾപ്പെടുത്തത്തേണ്ട ഹെഡർ ഫയലിന്റെ പേരെഴുതുക.

പ്രോഗ്രാം ഗാലറി

ഈ പാഠഭാഗത്ത് പ്രശ്ന പരിഹാരത്തിനായി വിവിധ നിയന്ത്രണ പ്രസ്താവനകൾ ഉപയോഗിച്ചുള്ള പ്രോഗ്രാമുകളുടെ ശേഖരമാണ് ഉള്ളത്. പ്രോഗ്രാമുകളുടെ മാതൃക ഔട്ട്പുട്ടുകൾ ഓരോ പ്രോഗ്രാമിനും ശേഷവും കൊടുത്തിട്ടുണ്ട്.

പ്രോഗ്രാം 7.19 $ax^2 + bx + c = 0$ എന്ന രൂപത്തിലുള്ള ഒരു ദ്വിമാന സമവാക്യത്തിന്റെ മൂന്ന് കോയിഫിഷന്റുകൾ സ്വീകരിക്കുകയും അതിന്റെ മൂല്യസംഖ്യ കണക്കാക്കുകയും ചെയ്യുന്നു. a യുടെ വില 0 (പൂജ്യം) ആകരുത്. ഈ പ്രശ്നം നിർദ്ധാരണം ചെയ്യുന്നതിന് $(b^2 - 4ac)$ എന്ന സൂത്രവാക്യം ഉപയോഗിച്ച് ദ്വിമാന സമവാക്യത്തിന്റെ ഡിസ്ക്രിമിനന്റ് മൂല്യം കണ്ടുപിടിക്കണം. മൂല്യസംഖ്യയുടെ തരം കണ്ടുപിടിക്കുന്നതിന് വർഗമൂലം കണ്ടുപിടിക്കുന്നതിനുള്ള സൂത്രവാക്യവും ലഭ്യമാണ്. ഈ പ്രോഗ്രാമിൽ `sqrt ()` എന്ന ഫങ്ഷനാണ് സംഖ്യയുടെ വർഗമൂലം കണ്ടുപിടിക്കാൻ ഉപയോഗിക്കുന്നത്. ഈ ഫങ്ഷൻ ഉപയോഗിക്കുന്നതിന് `cmath` (ടർബോ C++ ൽ `math.h`) എന്ന ഹെഡർ ഫയൽ പ്രോഗ്രാമിൽ ഉൾപ്പെടുത്തേണ്ടതാണ്.

Program 7.19: To find the roots of a quadratic equation

```
#include <iostream>
#include <cmath>// to use sqrt()function
using namespace std;
int main()
```

```
{
    float a, b, c, root1, root2, d;
    cout<< "Enter the three coefficients: ";
    cin >> a >> b >> c ;
    if (!a)  // equivalent to if (a == 0)
        cout<<"Value of \'a \' should not be zero\n"
            <<"Aborting!!!!!\n";
    else
    {
        d =b*b-4*a*c;       //beginning of else block
        if (d > 0)
        {
            root1 = (-b + sqrt(d))/(2*a);
            root2 = (-b - sqrt(d))/(2*a);
            cout<<"Roots are REAL and UNEQUAL\n";
            cout<<"Root1 = "<<root1<<"\tRoot2 = "<<root2;
        }
        else if (d == 0)
        {
            root1 = -b/(2*a);
            cout<<"Roots are REAL and EQUAL\n";
            cout<<"Root1 =" <<root1;
        }
        else
            cout<<"Roots are COMPLEX and IMAGINARY";
    }// end of else block of outer if
    return 0;
}
```

ഔട്ട്പുട്ട് 1:

```
Enter the three coefficients:    2     3     4
Roots are COMPLEX and IMAGINARY
```

ഔട്ട്പുട്ട് 2:

```
Enter the three coefficients:   3     5     1
Roots are REAL and UNEQUAL
Root1 =   -0.232408    Root2 = -1.434259
```

പ്രോഗ്രാം 7.20 ഒരു ഫിബിനോസി ശ്രേണിയിലെ N പദങ്ങൾ പ്രദർശിപ്പിക്കുന്നതിനുള്ളതാണ്. ശ്രേണി തുടങ്ങുന്നത് 0, 1 എന്ന പദങ്ങളിൽ നിന്നാണ് അടുത്ത പദം വരുന്നത് തൊട്ടു മുമ്പുള്ള രണ്ടു പദങ്ങളുടെ തുകയായാണ്. ശ്രേണി ഇപ്രകാരമാണ് 0, 1, 1, 2, 3, 5, 8, 13, ...

പ്രോഗ്രാം 7.20: ഫിബിനോസി ശ്രേണിയിലെ N പദങ്ങൾ പ്രദർശിപ്പിക്കൽ

```
#include <iostream>
using namespace std;
int main()
{
    int first=0, second=1, third, n;
    cout<<"\nEnter number of terms in the series: ";
    cin>>n;
    cout<<first<<"\t"<<second;
    for(int i=3; i<=n; ++i)
    {
        third = first + second;
        cout<<"\t"<<third;
        first = second;
        second = third;
    }
    return 0;
}
```

first, second എന്നീ വേരിയബിളുകളുടെ പ്രാരംഭ വിലകൾ യഥാക്രമം -1, +1 എന്ന് നൽകിയാൽ നമുക്ക് ശ്രേണിയിലെ ആദ്യത്തെ രണ്ട് പദങ്ങൾ പ്രദർശിപ്പിക്കാനുള്ള cout പ്രസ്താവന ഒഴിവാക്കാം.

ഔട്ട് പുട്ട്

```
Enter number of terms in the series: 10
0    1    1    2    3    5    8    13    21    34
```

പ്രോഗ്രാം 7.21 ഒരു സംഖ്യ സ്വീകരിക്കുകയും അത് പാലിൻഡ്രോം ആണോ അല്ലയോ എന്ന് പരിശോധിക്കുകയും ചെയ്യുന്നു. ഒരു സംഖ്യ പാലിൻഡ്രോം എന്ന് പറയണമെങ്കിൽ ആ സംഖ്യയും അതിന്റെ പ്രതിബിംബവും തുല്യമായിരിക്കണം. പ്രതിബിംബം എന്നത് കൊണ്ട് അർത്ഥമാക്കിയത് ഒരു സംഖ്യ തിരിച്ചെഴുതിയാലും അതേ സംഖ്യ കിട്ടുന്നു എന്നതാണ്.
അതായത് 163 ന്റെ പ്രതിബിംബം 361 ആണ്. ഈ രണ്ടു സംഖ്യകളും തുല്യമല്ലാത്തതിനാൽ 163 എന്ന സംഖ്യ പാലിൻഡ്രോം അല്ല. എന്നാൽ 232 എന്നത് ഒരു പാലിഡ്രോം സംഖ്യയാണ്.

പ്രോഗ്രാം 7.21: തന്നിരിക്കുന്ന സംഖ്യ പാലിഡ്രോം ആണോ അല്ലയോ എന്ന് പരിശോധിക്കുന്നതിന്.

```
#include <iostream>
using namespace std;
int main()
{
    int num, copy, digit, rev=0;
    cout<<"Enter the number: ";
    cin>>num;
    copy=num;
    while(num != 0)
```

ലൂപ്പിന്റെ പ്രവർത്തനം പൂർത്തിയാകുമ്പോൾ num എന്ന വേരിയബിളിന്റെ വില പൂജ്യമാകുന്നു. അതുകൊണ്ടാണ് അതിന്റെ ആദ്യത്തെ വില മറ്റൊരു വേരിയബിളിലേക്ക് പകർത്തിയത്

```
    {
        digit = num % 10;
        rev = (rev * 10)+ digit;
        num = num/10;
    }
    cout<<"The reverse of the number is: "<<rev;
    if (rev == copy)
        cout<<"\nThe given number is a palindrome.";
    else
        cout<<"\nThe given number is not a palindrome.";
    return 0;
}
```

ഔട്ട്പുട്ട് 1:

```
Enter the number: 363
The reverse of the number is: 363
The given number is a palindrome.
```

ഔട്ട്പുട്ട് 2:

```
Enter the number: 257
The reverse of the number is: 752
The given number is not a palindrome.
```

പ്രോഗ്രാം 7.22: N പൂർണ സംഖ്യകൾ സ്വീകരിച്ച് അതിലെ ഏറ്റവും വലിയ സംഖ്യ പ്രിന്റ് ചെയ്യുന്നതിന്

```
#include <iostream>
using namespace std;
int main()
{
    int num, big, count;
    cout<<"How many Numbers in the list? ";
    cin >> count;
    cout<<"\nEnter first number: ";
    cin >> num;
    big = num;
    for(int i=2; i<=count; i++)
    {
        cout<<"\nEnter next number: ";
        cin >> num;
        if(num > big) big = num;
    }
    cout<<"\nThe largest number is " << big;
    return 0;
}
```

ഔട്ട്പുട്ട്:

```
How many Numbers in the list? 5
Enter first number: 23
Enter next number: 12
Enter next number: -18
Enter next number: 35
Enter next number: 18
The largest number is 35
```

നമുക്ക് സംഗ്രഹിക്കാം

തീരുമാനങ്ങൾ എടുക്കുന്നതിനോ ആവർത്തന പ്രവർത്തനങ്ങൾ നടപ്പാക്കുന്നതിനോ ഉള്ള സൗകര്യങ്ങൾ ഉള്ള പ്രസ്താവനകളെ നിയന്ത്രണ പ്രസ്താവനകൾ എന്ന് അറിയപ്പെടുന്നു. നിയന്ത്രണ പ്രസ്താവനകൾ ഒരു കമ്പ്യൂട്ടർ പ്രോഗ്രാമിന്റെ നട്ടെല്ലാണ്. ഈ അധ്യായത്തിൽ വിവിധ തരം നിയന്ത്രണ പ്രസ്താവനകളായ തിരഞ്ഞെടുക്കൽ പ്രസ്താവനകൾ (`if`, `if...else`, `if...else if`, `switch`), ആവർത്തന പ്രസ്താവനകൾ (`for`, `while`, `do...while`) ജമ്പ് പ്രസ്താവനകൾ (`goto`, `break`, `continue`, `exit`) എന്നിവ നാം പഠിച്ചു. ഈ നിയന്ത്രണ പ്രസ്താവനകൾ കാര്യക്ഷമമായ C++ പ്രോഗ്രാമുകൾ എഴുതുന്നതിന് നമ്മെ സഹായിക്കും.

പഠനനേട്ടങ്ങൾ

ഈ അധ്യായം പൂർത്തിയാകുമ്പോൾ പഠിതാവ്

- പ്രശ്നങ്ങൾ നിർദ്ധാരണം ചെയ്യുന്നതിന് C++ ലെ നിയന്ത്രണ പ്രസ്താവനകൾ ഉപയോഗിക്കുന്നു.
- നിയന്ത്രണ പ്രസ്താവനകൾ ഒരു പ്രോഗ്രാമിൽ ഏതു സാഹചര്യത്തിലാണ് ഉപയോഗിക്കേണ്ടത് എന്ന് തിരിച്ചറിയുന്നു.
- സാഹചര്യത്തിന് അനുയോജ്യമായ ശരിയായ നിയന്ത്രണ പ്രസ്താവനകൾ ഉപയോഗിക്കുന്നു.
- വിവിധ തരം നിയന്ത്രണ പ്രസ്താവനകളെ തരം തിരിക്കുന്നു.
- C++ ലെ വിവിധ തരം ജമ്പ് പ്രസ്താവനകളെ തിരിച്ചറിയുന്നു.
- നിയന്ത്രണ പ്രസ്താവനകൾ ഉപയോഗിച്ച് C++ പ്രോഗ്രാം എഴുതുന്നു.

മാതൃകാ ചോദ്യങ്ങൾ

ഹ്രസ്വോത്തര ചോദ്യങ്ങൾ

1. switch പ്രസ്താവനയിൽ break- പ്രസ്താവനയുടെ പ്രാധാന്യം എഴുതുക. switch പ്രസ്താവനയിൽ break-ന്റെ അഭാവം എന്ത് ഫലം ഉളവാക്കും?

2. താഴെ കൊടുത്തിരിക്കുന്ന കോഡ് ശകലത്തിന്റെ ഔട്ട്പുട്ട് എന്തായിരിക്കും?

```
for(i=1;i<=10;++i) ;
cout<<i+5;
```

3. താഴെ പറയുന്ന പ്രസ്താവനയെ **while, do while** ലൂപ്പുകൾ ഉപയോഗിച്ച് മാറ്റി എഴുതുക.

```
for (i=1; i<=10; i++) cout<<i;
```

4. താഴെ കൊടുത്തിരിക്കുന്ന ലൂപ്പ് എത്ര തവണ പ്രവർത്തിക്കും.

```
int  s=0, i=0;
while(i++<5)
      s+=i;
```

5. exit( ) ഫങ്ഷൻ അടങ്ങിയിരിക്കുന്ന ഹെഡർ ഫയലിന്റെ പേരെഴുതുക.

6. പ്രോഗ്രാമിന്റെ നിയന്ത്രണം ഒരു ലേബലിലേക്ക് കൈമാറാൻ സാധിക്കുന്ന C++ ലെ പ്രസ്താവന ഏത്?

7. switch പ്രസ്താവനയിൽ default പ്രസ്താവനയുടെ ആവശ്യകത എന്ത്?

ലഘു ഉപന്യാസ ചോദ്യങ്ങൾ

1. താഴെ കൊടുത്തിരിക്കുന്ന രണ്ട് കോഡ് ശകലങ്ങൾ പരിഗണിക്കുക.

```
// version 1
cin>>mark;
if (mark > = 90)
cout<<" A+";
if (mark > = 80 && mark <90)
cout<<" A";
if (mark > = 70 && mark <80)
cout<<" B+";
if (mark > = 60 && mark <70)
cout<<" B";
```

```
//version 2
cin>>mark;
if (mark>=90)
cout<<" A+";
else if (mark>=80 && mark <90)
cout<<" A";
else if (mark>=70 && mark <80)
cout<<" B+";
else if (mark>=60 && mark <70)
cout<<" B";
```

വേർഷൻ 2 ന് വേർഷൻ 1 നെ അപേക്ഷിച്ചുള്ള മേന്മകൾ ചർച്ച ചെയ്യുക.

2. ഒരു for ലൂപ്പിന്റെ പ്രവർത്തനം അതിന്റെ വാക്യഘടന (Syntax) യോടുകൂടി ചുരുക്കി വിവരിക്കുക. നിങ്ങളുടെ ഉത്തരം സാധൂകരിക്കുന്നതിന് for ലൂപ്പിന്റെ ഒരു ഉദാഹരണം നൽകുക.
3. വിവിധ സാഹചര്യങ്ങളിൽ മൂന്നു ലൂപ്പുകളുടെ അനുയോജ്യത താരതമ്യം ചെയ്ത് ചർച്ച ചെയ്യുക.
4. താഴെ കൊടുത്തിരിക്കുന്ന if.. else if പ്രസ്താവന പരിഗണിക്കുക. switch പ്രസ്താവന കൊണ്ട് അത് മാറ്റി എഴുതുക.

```
if (a==1)
    cout << "One";
else if (a==0)
    cout << "Zero";
else
    cout << "Not a binary digit";
```

5. z = 3 ആണെങ്കിൽ താഴെ കൊടുത്തിരിക്കുന്ന while പ്രസ്താവനയിലെ തെറ്റ് എന്താണ്?

```
while(z>=0)
    sum+=z;
```

6. താഴെ കൊടുത്തിരിക്കുന്ന കോഡ് ശകലത്തിന്റെ ഔട്ട്പുട്ട് എന്തായിരിക്കും?

```
for (outer=10; outer > 5; --outer)
{
    for (inner=1; inner<4; ++inner)
        cout<<outer <<"\t"<<inner <<endl;
}
```

7. താഴെ കൊടുത്തിരിക്കുന്ന കോഡ് ശകലത്തിന്റെ ഔട്ട്പുട്ട് എന്തായിരിക്കും? വിശദീകരിക്കുക.

```
for (n = 1; n <= 10; ++n)
{
    for ( m=1; m <= 5 ; ++m)
        num = n*m;
    cout<<num <<endl;
}
```

8. ഒരു ലൂപ്പ് നിയന്ത്രണ വേരിയബിളിന്റെ പ്രാധാന്യം എഴുതുക. ഒരു ലൂപ്പിന്റെ വിവിധ ഭാഗങ്ങളെക്കുറിച്ച് ചുരുക്കി വിവരിക്കുക.

ഉപന്യസിക്കുക

1. താഴെ കൊടുത്തിരിക്കുന്ന കോഡ് ശകലം ഔട്ട്പുട്ട് എന്ത്?

```
int val, res, n=1000;
cin>>val;
res = n+val > 1750 ? 400 : 200;
```

(a) ഇൻപുട്ട് 2000 ആണെങ്കിൽ

(b) ഇൻപുട്ട് 500 ആണെങ്കിൽ

2. താഴെ പറയുന്നവ ഉപയോഗിച്ച് ഒരു സംഖ്യയിലെ അക്കങ്ങളുടെ തുക കണ്ടുപിടിക്കുന്നതിനുള്ള ഒരു പ്രോഗ്രാം എഴുതുക.

   (a) ആഗമന നിയന്ത്രണ ലൂപ്പ്

   (b) ബഹിർഗമന നിയന്ത്രണ ലൂപ്പ്

3. 1000 ൽ താഴെയുള്ള ആംസ്ട്രോങ്ങ് സംഖ്യ പ്രിന്റ് ചെയ്യുന്നതിനുള്ള ഒരു പ്രോഗ്രാം എഴുതുക. (ഒരു ആംസ്ട്രോങ്ങ് സംഖ്യ എന്നാൽ അതിലെ ഓരോ അക്കത്തിന്റെയും ക്യൂബുകളുടെ തുകയ്ക്ക് തുല്യമായിരിക്കും. ഉദാഹരണത്തിന് $153 = 1^3+5^3+3^3$)

4. C++ ലെ ലഭ്യമായ വിവിധ ജംമ്പ് പ്രസ്താവനകൾ വിശദീകരിക്കുക.

5. നെസ്റ്റഡ് ലൂപ്പ് ഉപയോഗിച്ച് താഴെ കൊടുത്തിരിക്കുന്ന ഔട്ട്പുട്ട് സൃഷ്ടിക്കുന്നതിനുള്ള ഒരു പ്രോഗ്രാം എഴുതുക.

```
A
A   B
A   B   C
A   B   C   D
A   B   C   D   E
```

6. `if...else` പ്രസ്താവനയിൽ `else` എന്ന വാക്ക് നിങ്ങൾ എഴുതാൻ മറന്നുപോയി എന്ന് വിചാരിക്കുക. നിങ്ങളുടെ പ്രോഗ്രാമിന്റെ ഔട്ട്പുട്ടിനെ ഇത് എങ്ങനെ ബാധിക്കുമെന്ന് ചർച്ച ചെയ്യുക.