3 ഡാറ്റാസ്ട്രക്ചറുകളും പ്രവർത്തനങ്ങളും

പഠന നേട്ടങ്ങൾ

ഈ അധ്യായത്തിന്റെ പൂർത്തീകരണത്തിന് ശേഷം പഠിതാവിന്

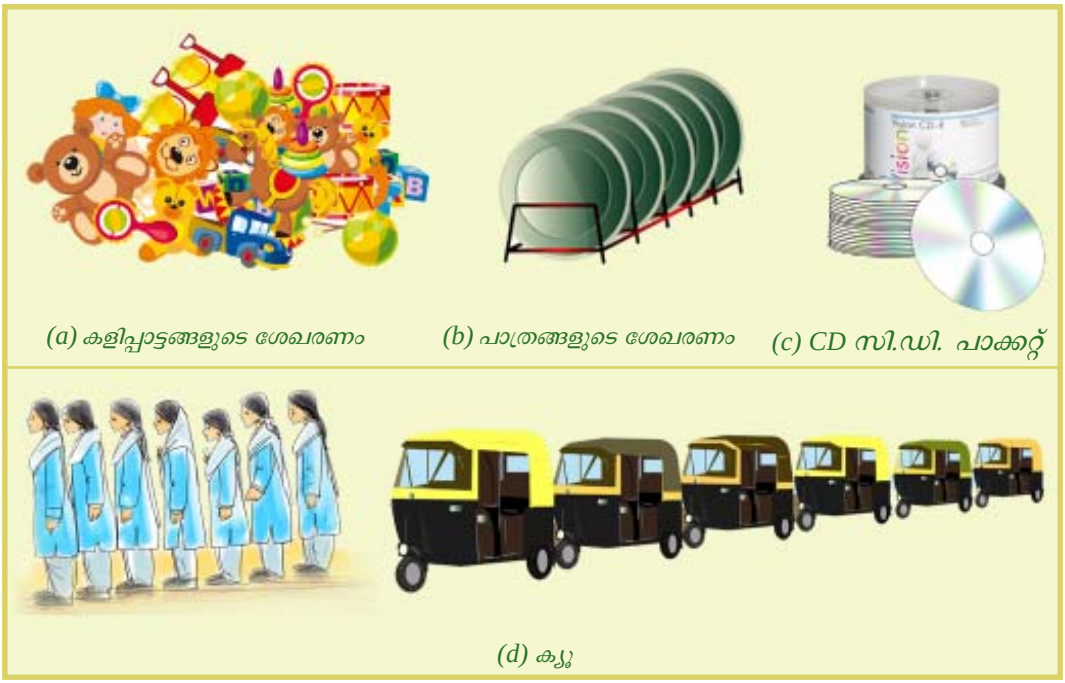
- ഡാറ്റ സ്ട്രക്ചറുകളുടെ ആശയം ഉദാഹരണ സഹിതം വിവരിക്കുവാൻ സാധിക്കുന്നു.
- ഡാറ്റ സ്ട്രക്ചറുകളെ വിവിധ മാനദണ്ഡങ്ങൾക്കനുസരിച്ചു തരംതിരിക്കാൻ സാധിക്കുന്നു.
- ഡാറ്റ സ്ട്രക്ചറുകളിലുള്ള വിവിധ പ്രവർത്തനങ്ങൾ ഏതൊക്കെ എന്നും മനസ്സിലാക്കാനും അവയെക്കുറിച്ച് വിശദീകരിക്കുവാനും സാധിക്കുന്നു.
- സ്റ്റാക്ക് ഡാറ്റ സ്ട്രക്ചറിന്റെ ഘടന ഉദാഹരണസഹിതം വിശദീകരിക്കാൻ സാധിക്കുന്നു.
- സ്റ്റാക്കിലെ പുഷ്, പോപ്പ് പ്രവർത്തനങ്ങൾക്കുള്ള അൽഗോരിതം വികസിപ്പിക്കാൻ സാധിക്കുന്നു.
- ക്യൂ സ്ട്രക്ചറിന്റെ ഘടന ഉദാഹരണ സഹിതം വിശദീകരിക്കാൻ സാധിക്കുന്നു.
- രേഖീയമായ ക്യൂവിൽഉൾപ്പെടുത്താനും, നീക്കം ചെയ്യൽ പ്രവർത്തനങ്ങൾക്കുള്ള അൽഗോരിതം വികസിപ്പിക്കാനും സാധിക്കുന്നു.
- വൃത്താകൃതിയിലുള്ള ക്യൂവിനു രേഖീയമായ ക്യൂവിനു മേലുള്ള നേട്ടങ്ങൾ വിശദീകരിക്കാൻ സാധിക്കുന്നു.
- ലിങ്ക്ഡ് ലിസ്റ്റ് ഡാറ്റ സ്ട്രക്ചറുകളുടെ ആശയം വിശദീകരിക്കാനും അറെകളുടെയും മറ്റു സ്ഥിര ഡാറ്റ സ്ട്രക്ചറുകളുടെയും മേൽ അവക്കുള്ള നേട്ടങ്ങൾ വിവരിക്കാനും സാധിക്കുന്നു.

കമ്പ്യൂട്ടറുകൾ ഉപയോഗിച്ച് പ്രശ്നം പരിഹരിക്കുമ്പോൾ മിക്ക സന്ദർഭങ്ങളിലും ഡാറ്റ പ്രോസസ് ചെയ്യേണ്ടി വരാം. ഈ ഡാറ്റയുടെ തരം അടിസ്ഥാനപരമായതോ സംഗ്രഹമായതോ (ഒരു കൂട്ടം) ആകാം. ഇത്തരം ഡാറ്റയെ പരാമർശിക്കുന്നതിനു വേണ്ടി വേരിയബിളുകൾ ആവശ്യമാണെന്ന് നമ്മുക്കറിയാവുന്നതാണ്. C, C++, ജാവ മുതലായ ഭാഷകളിലെ പ്രോഗ്രാമിൽ വേരിയബിളുകൾ ഉപയോഗിക്കുന്നതിനു മുമ്പ് അവയുടെ പ്രഖ്യാപനം വേണമെന്നു ഊന്നിപ്പറയുന്നു. അടിസ്ഥാനപരമായ ഡാറ്റകൾക്ക് വേണ്ടിയുള്ള വേരിയബിളുകൾ `int`, `char`, `float`, `double` മുതലായ അടിസ്ഥാന ഡാറ്റതരങ്ങളോ അവയുടെ ടൈപ്പ് മോഡിഫൈറുകളോ ഉപയോഗിച്ചാണ് പ്രഖ്യാപിക്കുന്നത് എന്നു നമ്മൾ പഠിച്ചതാണ്. കൂട്ടമായിട്ടുള്ള ഡാറ്റയെ സൂചിപ്പിക്കാൻ അറെകളും സ്ട്രക്ചറുകളും ഉപയോഗിക്കുന്നു എന്നും നാം കണ്ടതാണ്. ഒരേ തരത്തിലുള്ള ഡാറ്റയുടെ ശേഖരത്തെ അറെ എന്നും വ്യത്യസ്ത തരത്തിലുള്ള ഡാറ്റയുടെ ശേഖരത്തെ സ്ട്രക്ചർ എന്നും പറയുന്നു.

വിവിധ തത്ത്വങ്ങളുടെയും മാനദണ്ഡങ്ങളുടെയും അടിസ്ഥാനത്തിൽ പ്രോഗ്രാമിങ് ഭാഷകളിൽ ഡാറ്റയെ വ്യത്യസ്ത തരങ്ങളായി ചിട്ടപ്പെടുത്താൻ ഉപയോഗിക്കുന്ന സംവിധാനങ്ങളെയാണ് ഈ അധ്യായത്തിൽ അവതരിപ്പിച്ചിരിക്കുന്നത്. ഒരു തരത്തിൽ ഉൾക്കൊള്ളാവുന്ന ഡാറ്റായുടെ അളവും അതിന്റെ മുകളിൽ പ്രാവർത്തിക മാക്കാവുന്ന പ്രവർത്തനങ്ങളും, ഡാറ്റാ ഗണങ്ങൾ ചിട്ടപ്പെടുത്തുവാൻ ഉപയോഗിക്കുന്ന തത്ത്വങ്ങളുടെ അടിസ്ഥാനത്തിൽ വ്യത്യസ്തമായതായിരിക്കും.

3.1 ഡാറ്റ സ്ട്രക്ചർ

ചിത്രം 3.1 ൽ ചില ഗ്രൂപ്പുകൾ കാണിച്ചിരിക്കുന്നു. ഓരോ ഗ്രൂപ്പിലും ക്രമീകരണത്തിനായി പലതരത്തിലുള്ള തന്ത്രങ്ങൾ ഉപയോഗിച്ചിരിക്കുന്നു. ഇതിലെ ഓരോ ചിത്രവും ഓരോ വസ്തുക്കളുടെ ശേഖരമാണ്. ഓരോ ഗ്രൂപ്പിലും വസ്തുക്കളെ ക്രമീകരിക്കാൻ ഉപയോഗിച്ചിരിക്കുന്ന തത്ത്വം അല്ലെങ്കിൽ മാതൃക ഏതാണെന്നു പറയാമോ?



(a) കളിപ്പാട്ടങ്ങളുടെ ശേഖരണം *(b) പാത്രങ്ങളുടെ ശേഖരണം* *(c) CD സി.ഡി. പാക്കറ്റ്*

(d) ക്യൂ

Fig. 3.1: വിവിധ തരത്തിലുള്ള ശേഖരങ്ങൾ

കളിപ്പാട്ടങ്ങളുടെ ഒരു ശേഖരമാണ് ചിത്രം 3.1(a) ൽ കാണുന്നത്. പ്രത്യേകിച്ച് ഒരു ക്രമമോ ക്രമീകരണമോ ഇല്ലാതെ കൂട്ടിയിട്ടിരിക്കുകയാണ് ഈ കളിപ്പാട്ടങ്ങൾ. ചിത്രം 3.1(b) ൽ കാണിച്ചിരിക്കുന്നത് ഒരു തട്ടിലെ ഒരു കൂട്ടം പാത്രങ്ങളാണ്. ഒന്നിന് പുറകിൽ മറ്റൊന്നായിട്ടാണ് പാത്രങ്ങൾ വച്ചിരിക്കുന്നത്. തട്ടിൽ സൂക്ഷിക്കാവുന്ന പാത്രങ്ങളുടെ എണ്ണത്തിന് ഒരു പരിമിതിയുണ്ട്. സ്ഥലമുണ്ടെങ്കിൽ തട്ടിൽ എവിടെ വേണമെങ്കിലും പുതിയ പാത്രം വയ്ക്കാവുന്നതും ഏതു പാത്രം വേണമെങ്കിലും തിരികെ എടുക്കാവുന്നതുമാണ്. സി ഡി പായ്ക്കറ്റിലെ ഒരു കൂട്ടം സി ഡികൾ ആണ് ചിത്രം 3.1(c) ൽ കാണിച്ചിരിക്കുന്നത്. ഈ ശേഖരത്തിലും ഡിസ്കുകളുടെ എണ്ണത്തിന് ഒരു പരിധിയുണ്ട്. ഒരു പുതിയ സി.ഡി. ശേഖരത്തിന്റെ മുകളിൽ മാത്രമേ ചേർക്കാൻ സാധ്യമാകുകയുള്ളൂ. അതുപോലെ മുകളിലുള്ള സി.ഡി. യെ മാത്രമെ ശേഖരത്തിൽ നിന്നും ഒഴിവാക്കുവാനും സാധിക്കുകയുള്ളൂ. പുറകുവശത്തു കൂടി മാത്രം അടുത്ത വ്യക്തിക്ക് (അല്ലെങ്കിൽ അടുത്ത ഓട്ടോറിക്ഷക്ക്) വരിയിൽ ചേരാൻ കഴിയുന്ന തരത്തിലുള്ള ഒരു വരി ആണ് ചിത്രം 3.1(d) ൽ സൂചിപ്പിക്കുന്നത്. വരിയുടെ മുൻവശത്തു കൂടി മാത്രമെ ഒരു വ്യക്തിക്ക് (അല്ലെങ്കിൽ ഓട്ടോറിക്ഷയ്ക്ക്) വരിയിൽ നിന്ന് പുറത്തേക്കു പോകുവാൻ സാധിക്കുകയുള്ളൂ. ഈ വരിയിൽ വ്യക്തികളുടെ എണ്ണത്തിന് പരിധി ഉണ്ടാകണമെന്നില്ല. എന്നാൽ ചില സാഹചര്യങ്ങളിൽ വരിയുടെ വലുപ്പത്തിന് പരിധിയുണ്ടാകാം.

ചിത്രം 3.1 ലെ (b), (c), (d) എന്നീ ചിത്രങ്ങളിലെ ശേഖരങ്ങളുമായി സാമ്യമുള്ളതാണ് ഡാറ്റാസ്ട്രക്ചറുകളുടെ തത്ത്വം. പതിനൊന്നാം തരത്തിൽ നമ്മൾ പഠിച്ച അറെയുമായി സാമ്യമുള്ളതാണ് ചിത്രം 3.1(b) ൽ കാണിച്ചിരിക്കുന്ന ശേഖരം. അതുകൊണ്ട് അറെ ഒരു ഡാറ്റ സ്ട്രക്ചർ ആണെന്ന് നമുക്ക് പറയാവുന്നതാണ്. ഒറ്റ ഘടകമായി പ്രോസസ് ചെയ്യാവുന്ന യുക്തിപരമായി ഒരേപോലെ ഉള്ളതും അല്ലെങ്കിൽ വ്യത്യസ്തമായതും ആയ ഡാറ്റയെ പ്രത്യേക രീതിയിൽ ക്രമീകരിക്കാനുള്ള മാർഗത്തെയാണ് കമ്പ്യൂട്ടർ ശാസ്ത്രത്തിൽ ഡാറ്റ സ്ട്രക്ചർ എന്ന് പറയുന്നത്. ഉപയോക്താവിന് വിവിധ തരത്തിലുള്ള ഡാറ്റയെ കൂട്ടിയോജിപ്പിക്കാൻ മാത്രമല്ല അവയെ ഒറ്റ ഘടകമായി കണക്കാക്കി പ്രവർത്തിക്കാനും ഡാറ്റാസ്ട്രക്ചർ സഹായിക്കുന്നു.

3.1.1 ഡാറ്റാസ്ട്രക്ചറുകളുടെ തരം തിരിക്കൽ (Classification of data structures)

പ്രാഥമികമായ ഡാറ്റാസ്ട്രക്ചറുകളായ അറെ, സ്ട്രക്ചർ മുതലായവ നമുക്ക് പരിചിതരാണ്.

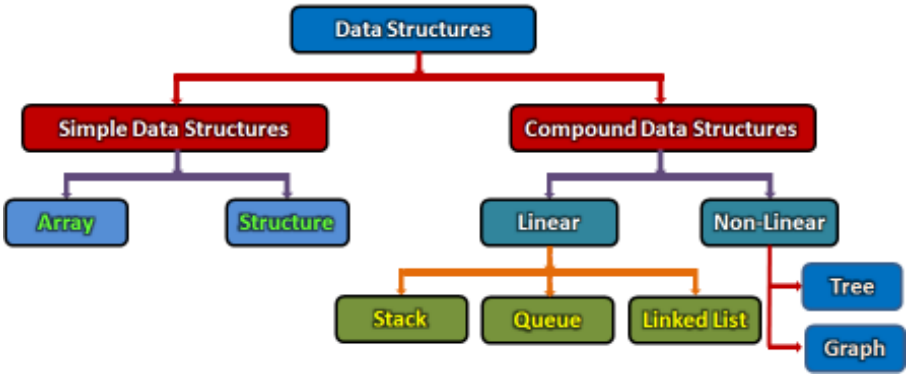


Fig. 3.2: Classification of data structures

ഡാറ്റയുടെ ശേഖരത്തെ സൂചിപ്പിക്കുന്നത്തിനായി C++ പ്രോഗ്രാമുകളിൽ നമ്മൾ അവ ഉപയോഗിച്ചതുമാണ്. ഇത്തരം പ്രാഥമിക ഡാറ്റാസ്ട്രക്ചറുകളെ വിവിധ തരത്തിൽ കൂട്ടിയോജിപ്പിച്ച് സങ്കീർണമായ ഡാറ്റ സ്ട്രക്ചറുകൾ രൂപപ്പെടുത്തുന്നു. ചിത്രം 3.2 ൽ കാണിച്ചിരിക്കുന്നത് പോലെ സങ്കീർണമായ ഡാറ്റ സ്ട്രക്ചറുകളെ രേഖീയമായതും (Linear) രേഖീയമല്ലാത്തതും (Non-Linear) എന്ന് വീണ്ടും തരംതിരിച്ചിരിക്കുന്നു. ഒരു ഡാറ്റാസ്ട്രക്ചറിലെ ഡാറ്റാ അംഗങ്ങൾ അനുവർത്തനമായി ക്രമീകരിക്കപ്പെടുകയാണെങ്കിൽ ആ ഡാറ്റാ സ്ട്രക്ചറിനെ രേഖീയമായത് എന്ന് പറയുന്നു. രേഖീയമായ ഡാറ്റാസ്ട്രക്ചറിന്റെ അംഗങ്ങളെ പ്രതിനിധീകരിക്കാൻ തുടർച്ചയായ മെമ്മറി ലൊക്കേഷനുകൾ ഉപയോഗിക്കുന്നു. ഒരു ഡാറ്റാസ്ട്രക്ചറിലെ അംഗങ്ങൾ മെമ്മറിയിൽ അനുവർത്തന ക്രമത്തിലല്ലെങ്കിൽ ആ ഡാറ്റാസ്ട്രക്ചറിനെ ഒരു രേഖീയമല്ലാത്ത ഡാറ്റാസ്ട്രക്ചർ എന്ന് പറയുന്നു. ഇത്തരം ഡാറ്റാസ്ട്രക്ചറുകളിൽ ക്രമത്തിലല്ലാതെയുള്ള മെമ്മറി ലൊക്കേഷനുകളിലാണ് ഡാറ്റാ അംഗങ്ങൾ സംഭരിക്കപ്പെടുക. മാത്രമല്ല അവ ഉപയോഗിക്കപ്പെടുന്നത് ക്രമത്തിലായിരിക്കണമെന്നില്ല. രേഖീയമല്ലാത്ത ഡാറ്റാ സ്ട്രക്ചറുകൾ വളരെ സങ്കീർണമായതിനാൽ ഉപരിപഠന സമയത്ത് അവയെപ്പറ്റി വിശദമായി പഠിക്കുന്നതാണ്. രേഖീയമായ ഡാറ്റാസ്ട്രക്ചറുകളായ സ്റ്റാക്ക് (Stack), ക്യൂ (Queue), ലിങ്ക്ഡ് ലിസ്റ്റ് (Linked List) എന്നിവയെപ്പറ്റി വിശദമായി ഈ പാഠഭാഗത്തിൽ അവതരിപ്പിച്ചിരിക്കുന്നു.

ഡാറ്റാസ്ട്രക്ചറുകൾ ഡാറ്റായുടെ ശേഖരത്തെ സൂചിപ്പിക്കുന്നതിനാൽ അവയ്ക്ക് കമ്പ്യൂട്ടറിന്റെ മെമ്മറിയുമായി വളരെ അടുത്ത ബന്ധമാണുള്ളത്. എന്തെന്നാൽ ഡാറ്റയെ സംഭരിക്കാനുള്ള സ്ഥലമാണ് മെമ്മറി. മെമ്മറി പ്രാഥമികമോ ദ്വിതീയമോ ആകാം. മെമ്മറി നീക്കിവയ്പ്പിനനുസരിച്ച് ഡാറ്റാസ്ട്രക്ചറുകളെ സ്ഥിര (static) ഡാറ്റാസ്ട്രക്ചർ എന്നും അസ്ഥിര (dynamic) ഡാറ്റാസ്ട്രക്ചർ എന്നും രണ്ടായി തരം തിരിക്കാം. സ്ഥിര ഡാറ്റാ സ്ട്രക്ചറുകൾ പ്രാഥമിക മെമ്മറിയുമായി മാത്രം ബന്ധപ്പെട്ടിരിക്കുന്നു. പ്രോഗ്രാമിന്റെ പ്രവർത്തനത്തിനു മുമ്പായി ആവശ്യമായ മെമ്മറി നീക്കിവയ്ക്കുകയും പ്രവർത്തന ഘട്ടത്തിലുടനീളം അതിനു മാറ്റംവരാതെ നിലകൊള്ളുകയും ചെയ്യുന്നു. അതായത് ഡാറ്റാസ്ട്രക്ചർ രൂപകൽപ്പന ചെയ്യുമ്പോൾത്തന്നെ അതിന്റെ വലുപ്പം നിർണയിക്കുന്നതുകൊണ്ട് പിന്നീട് അത് മാറ്റാൻ സാധിക്കുന്നതല്ല. അറെകൾ ഉപയോഗിച്ച് രൂപകൽപ്പന ചെയ്തിരിക്കുന്നതും പ്രാവർത്തികമാക്കിയിരിക്കുന്നതും ആയ ഡാറ്റാ സ്ട്രക്ചറുകൾ സ്ഥിര സ്വഭാവമുള്ളവയായിരിക്കും. പക്ഷെ അസ്ഥിര ഡാറ്റാ സ്ട്രക്ചറുകൾക്ക് പ്രവർത്തന ഘട്ടത്തിലാണ് മെമ്മറി നിർണയിക്കപ്പെടുക. ലിങ്ക്ഡ് ലിസ്റ്റുകൾ ഉപയോഗിച്ച് പ്രാവർത്തികമാക്കിയിരിക്കുന്ന ഡാറ്റാസ്ട്രക്ചറുകൾ അസ്ഥിരങ്ങളായിരിക്കും. അസ്ഥിര ഡാറ്റാസ്ട്രക്ചറിൽ ശേഖരത്തിന്റെ വലുപ്പം മുൻകൂട്ടി പ്രസ്താവിച്ചിരിക്കുകയില്ല, പകരം ഉപയോക്താവിന്റെ ഇഷ്ടാനുസരണം പ്രവർത്തന വേളയിൽ വളരുകയും ചുരുങ്ങുകയും ചെയ്തു കൊണ്ടിരിക്കും. ഡാറ്റാ സംഭരണത്തിനായി ദ്വിതീയ മെമ്മറി നമ്മൾ ഉപയോഗിക്കുമ്പോൾ അത് ഫയലുകളുടെ രൂപത്തിലായിരിക്കും. ഡാറ്റ കൂട്ടിച്ചേർക്കുന്നതിനനുസരിച്ചു ഫയലിന്റെ വലുപ്പം കൂടുകയും ഡാറ്റ നീക്കം ചെയ്യുന്നതിനനുസരിച്ച് വലുപ്പം കുറയുകയും ചെയ്യുന്നു. അതിനാൽ ഫയലുകളെയും അസ്ഥിര ഡാറ്റാസ്ട്രക്ചറുകൾ എന്ന് പറയുന്നു.

3.1.2 ഡാറ്റാസ്ട്രക്ചറുകളുടെ മേലുള്ള പ്രവർത്തനങ്ങൾ (Operations on data structures)

ഡാറ്റാസ്ട്രക്ചറുകൾ ഉപയോഗിച്ച് പ്രതിനിധീകരിച്ചിരിക്കുന്ന ഡാറ്റ ചില പ്രവർത്തനങ്ങളുടെ സഹായത്തോടെയാണ് പ്രോസസ് ചെയ്യപ്പെടുന്നത്. വാസ്തവത്തിൽ ചില പ്രത്യേക പ്രവർത്തനങ്ങൾ എത്ര തവണ പ്രവർത്തിക്കണം എന്നതിനനുസരിച്ചാണ് ഡാറ്റാസ്ട്രക്ചർ തിരഞ്ഞെടുക്കുന്നത്. കടന്നു പോകുക, തിരയുക, ഉൾപ്പെടുത്തുക, നീക്കം ചെയ്യുക, ക്രമപ്പെടുത്തുക, ലയിപ്പിക്കുക മുതലായവയാണ് ഡാറ്റാസ്ട്രക്ചറുകളുടെ മേൽ ചെയ്യാവുന്ന പ്രവർത്തനങ്ങൾ. ഇവയെ നമുക്ക് പരിചയപ്പെടാം.

a. കടന്നുപോകുക (Traversing)

ഡാറ്റാ സ്ട്രക്ച്ചറിലെ ഓരോ അംഗത്തെയും സന്ദർശിക്കുന്ന പ്രവർത്തനമാണ് കടന്നു പോകൽ. ആദ്യത്തെ അംഗത്തിൽ തുടങ്ങി അവസാനത്തെ അംഗം വരെ സഞ്ചാരം തുടരുന്നു. സന്ദർശിച്ച അംഗത്തെ എങ്ങനെ പ്രോസസ് ചെയ്യണമെന്നുള്ളത് പ്രശ്നത്തിന്റെ ആവശ്യകതയ്ക്കനുസരിച്ചായിരിക്കും. ഒരു അറേയിലുള്ള എല്ലാ അംഗങ്ങളും തിരിച്ചെടുക്കുന്നത് കടന്നുപോകുന്ന പ്രവർത്തനത്തിന് ഒരു ഉദാഹരണമാണ് (പതിനൊന്നാം തരത്തിലെ കമ്പ്യൂട്ടർ സയൻസ് പുസ്തകത്തിലെ അധ്യായം 8 നോക്കുക).

b. തിരയൽ (Searching)

ഒരു ഡാറ്റാസ്ട്രക്ചറിലെ ഒരു പ്രത്യേകഅംഗത്തിന്റെ സ്ഥാനം കണ്ടുപിടിക്കാനുള്ള പ്രക്രിയയെയാണ് തിരയൽ എന്നത് കൊണ്ടുദ്ദേശിക്കുന്നത്. ഒന്നോ അതിലധികമോ വ്യവസ്ഥകൾ നിറവേറ്റുന്ന എല്ലാ അംഗങ്ങളുടെയും സ്ഥാനം കണ്ടുപിടിക്കാനുള്ള

പ്രക്രിയയെയും തിരയൽ എന്നുപറയാം. മറ്റൊരർഥത്തിൽ തിരയലെന്നാൽ ഡാറ്റാ സ്ട്രക്ചറിൽ സംഭരിച്ചു വച്ചിരിക്കുന്ന വിലകൾ കണ്ടെത്തി ഉപയോഗിക്കുക എന്നാണ്. പതിനൊന്നാം തരത്തിൽ തിരയലിനുള്ള രണ്ടുമാർഗങ്ങൾ നമ്മൾ പഠിച്ചതാണ്.

c. ഉൾപ്പെടുത്തൽ (Inserting)

ഡാറ്റാസ്ട്രക്ചറിൽ ഒരു പ്രത്യേക സ്ഥാനത്തേക്ക് പുതിയ അംഗത്തെ ചേർക്കുന്ന പ്രക്രിയയാണ് ഉൾപ്പെടുത്തൽ എന്ന് പറയുന്നത്. ചില സാഹചര്യങ്ങളിൽ, പ്രത്യേകിച്ച് ഡാറ്റാസ്ട്രക്ചറിലെ ഡാറ്റാഅംഗങ്ങൾ ക്രമത്തിലാണെങ്കിൽ, പുതിയ ഡാറ്റ ഉൾപ്പെടുത്തേണ്ട സ്ഥാനം ആദ്യം കണ്ടെത്തുകയും അതിനു ശേഷം ഡാറ്റാ ഉൾപ്പെടുത്തുകയും ചെയ്യുന്നു.

d. നീക്കം ചെയ്യൽ (Deleting)

ഡാറ്റസ്ട്രക്ചറിൽ നിന്നും ഒരു അംഗത്തെ ഒഴിവാക്കുന്ന പ്രക്രിയയെയാണ് നീക്കം ചെയ്യൽ എന്നു പറയുന്നത്. നീക്കം ചെയ്യുന്നതിനായി ഡാറ്റ അംഗത്തിന്റെ സ്ഥാനമോ നീക്കം ചെയ്യേണ്ട അംഗത്തെ തന്നെയോ പരാമർശിക്കണം.

e. ക്രമപ്പെടുത്തൽ (Sorting)

ബബിൾ സോർട്ട്, സെലക്ഷൻ സോർട്ട് എന്നീ രണ്ടു രീതികൾ ഉപയോഗിച്ച് ഒരു അറെയെ ക്രമപ്പെടുത്തുന്നത് നമ്മൾ നേരത്തെ പരിചയപ്പെട്ടതാണ്. അംഗങ്ങളെ ഒരു പ്രത്യേക ക്രമത്തിൽ അതായത് ആരോഹണ ക്രമത്തിലോ അവരോഹണ ക്രമത്തിലോ അടുക്കി വയ്ക്കുന്ന രീതിയെയാണ് ക്രമപ്പെടുത്തൽ എന്ന് പറയുന്നത്. ഡാറ്റാസ്ട്രക്ചറിലെ ഡാറ്റാ അംഗങ്ങളെ ക്രമപ്പെടുത്തുന്നത് തിരയൽ വേഗത്തിലാക്കാൻ സഹായിക്കുന്നു.

f. ലയിപ്പിക്കൽ (Merging)

ക്രമപ്പെടുത്തിയിരിക്കുന്ന രണ്ടു ഡാറ്റാസ്ട്രക്ചറുകളുടെ അംഗങ്ങളെ കൂട്ടിയോജിപ്പിച്ച് പുതിയ ഒരു ഡാറ്റാസ്ട്രക്ചർ രൂപീകരിക്കുന്ന പ്രക്രിയയെ ലയിപ്പിക്കൽ എന്ന് പറയുന്നു. ലയിപ്പിക്കലിന്റെ ഏറ്റവും ലളിതമായ രൂപം, അംഗങ്ങളില്ലാത്ത ഒരു ഡാറ്റാസ്ട്രക്ചറിലേക്കു മറ്റു രണ്ടു ഡാറ്റാസ്ട്രക്ചറുകളെ കൂട്ടിച്ചേർക്കുന്നതാണ്. അറെ ഉപയോഗിച്ചു ചെയ്യുകയാണെങ്കിൽ ആദ്യം ഒരു അറെയിലെ അംഗങ്ങൾ, അംഗങ്ങളില്ലാത്ത മൂന്നാമത്തെ അറെയിലേക്ക് പകർത്തുകയും, ശേഷം രണ്ടാമത്തെ അറെയിലെ അംഗങ്ങൾ മൂന്നാമത്തെ അറെയിലേക്ക് കൂട്ടിച്ചേർക്കുകയും ചെയ്യുന്നു.

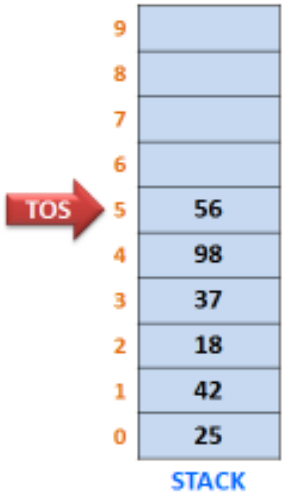
തിരയൽ, ക്രമപ്പെടുത്തൽ, ലയിപ്പിക്കൽ എന്നീ മൂന്ന് പ്രവർത്തനങ്ങളാണ് സംഭരണ ഉപകരണങ്ങളിൽ നിന്നും ഡാറ്റ വീണ്ടെടുക്കുന്ന പ്രക്രിയ എളുപ്പവും വേഗത്തിലും കാര്യക്ഷമവുമാക്കുന്നത്. അറെ എന്ന ഡാറ്റാസ്ട്രക്ചറിനുക്കുറിച്ചും അതിന്മേൽ പ്രാവർത്തികമാക്കാവുന്ന മേല്പറഞ്ഞ പ്രവർത്തനങ്ങളെ പറ്റിയും നമ്മൾ പതിനൊന്നാം തരത്തിൽ പഠിച്ചതാണ്. മാത്രമല്ല സ്ട്രക്ചറുകളെയും അവയുടെ അംഗങ്ങളുടെ മേലുള്ള പ്രവർത്തനങ്ങളെയും കുറിച്ച് ഈ പുസ്തകത്തിന്റെ ഒന്നാം അധ്യായത്തിൽ നാം ചർച്ച ചെയ്തിരുന്നു. ഇനി നമുക്ക് സങ്കീർണമായ രേഖീയ ഡാറ്റസ്ട്രക്ചറുകളായ സ്റ്റാക്ക്, ക്യൂ, ലിങ്ക്ഡ് ലിസ്റ്റ് മുതലായവയെപ്പറ്റി ചർച്ച ചെയ്യാം.

3.2 സ്റ്റാക്കുകൾ (Stack)

ചിത്രം 3.1(c) ഒരിക്കൽക്കൂടി ശ്രദ്ധിക്കുക. താഴെ കൊടുത്തിരിക്കുന്ന ചിത്രം 3.3 ൽ കാണിച്ചിരിക്കുന്ന ശേഖരവും നോക്കുക. ഈ രണ്ടു വിഭാഗത്തിലും കൊടുത്തിരിക്കുന്ന ഇനങ്ങളുടെ ഘടന ഒന്നുതന്നെയാണ്.

ചിത്രം 3.3: നിത്യജീവിതത്തിലെ സ്റ്റാക്കിന് ഉദാഹരണങ്ങൾ

ഒരു ഇനത്തിന്റെ മുകളിൽ മറ്റൊന്ന് ചേർത്തിട്ടാണ് ഇവിടെ ശേഖരം രൂപീകരിച്ചിരിക്കുന്നത്. മറ്റൊരു തരത്തിൽ പറഞ്ഞാൽ ഇനങ്ങൾ ഏറ്റവും മുകളിലാണ് ചേർക്കുന്നത്. അതുപോലെ ഏറ്റവും അവസാനം ചേർത്ത ഇനം മാത്രമേ നമ്മുക്ക് നീക്കം ചെയ്യാൻ സാധിക്കൂ. ഈ ക്രമീകരണ തത്ത്വത്തെ ലാസ്റ്റ് ഇൻ ഫസ്റ്റ് ഔട്ട് (LIFO) എന്ന് പറയുന്നു. LIFO തത്ത്വം പിന്തുടരുന്ന ഡാറ്റ സ്ട്രക്ചറിനെ സ്റ്റാക് എന്നു പറയുന്നു. LIFO എന്ന് പറയുന്ന അറ്റത്തു നിന്ന് മാത്രം ഇനങ്ങൾ കൂട്ടിച്ചേർക്കുകയും നീക്കംചെയ്യുകയും ചെയ്യുന്ന ക്രമപ്പെടുത്തിയ പട്ടികയാണ് ഇത്.



ചിത്രം 3.4: പൂർണ്ണസംഖ്യകളുടെ സ്റ്റാക്ക്

സ്റ്റാക്ക് എന്നത് യുക്തിപരമായ ഒരു ആശയമാണ്. സ്റ്റാക്ക് നിർമിക്കാൻ പ്രോഗ്രാമിങ് ഭാഷകളിൽ ഒരു പ്രത്യേക സംവിധാനം ഇല്ല. അറെ ഉപയോഗിച്ച് നമ്മുക്ക് സ്റ്റാക്ക് പ്രാവർത്തികമാക്കാവുന്നതാണ്. അത്തരത്തിലുള്ള ഒരു സ്റ്റാക്കിന് അറെയുടെ എല്ലാ ഗുണങ്ങളും ഉണ്ടായിരിക്കുന്നതാണ്. ഇതിന്റെ വലുപ്പം മുമ്പേ നിർണയിക്കപ്പെട്ടിരിക്കുന്നതിനാൽ ഇത് സ്ഥിരമായിരിക്കുന്നതാണ്. അറെ ഉപയോഗിച്ച് പ്രാവർത്തികമാക്കിയിരിക്കുന്ന ഒരു സ്റ്റാക്ക് ആണ് ചിത്രം 3.4 ൽ കാണിച്ചിരിക്കുന്നത്. പരമാവധി 10 പൂർണസംഖ്യകളെ ഇതിൽ ഉൾപ്പെടുത്താനാകും. ചിത്രം 3.4 പ്രകാരം നിലവിൽ സ്റ്റാക്കിൽ ആറു അംഗങ്ങൾ ആണുള്ളത്. അതുപോലെ അഞ്ചാം സ്ഥാനത്തുള്ള 56 എന്നതാണ് ഈ സ്റ്റാക്കിലെ അവസാനത്തെ അംഗം. അങ്ങനെയെങ്കിൽ **TOS** ന്റെ മൂല്യം 5 ആയിരിക്കും. സ്റ്റാക്കിന്റെ അവസാനത്തെ അംഗത്തെ സൂചിപ്പിക്കുന്നതിനു വേണ്ടി സ്റ്റാക്കിന്റെ പേരിന് **StackName[TOS]** എന്ന പദപ്രയോഗം ഉപയോഗിക്കുന്നു. ഇവിടെ **STACK[TOS]** എന്നത് 56 ആയിരിക്കും. എന്തെന്നാൽ സ്റ്റാക് നിർമിക്കാൻ നമ്മൾ അറേ ആണ് ഉപയോഗിച്ചിരിക്കുന്നത്. അറെയിൽ ആദ്യത്തെ അംഗത്തെ സൂചിപ്പിക്കാൻ 0 എന്ന സൂചികയാണ് ഉപയോഗിക്കുന്നത്.ഇവിടെ **STACK[0]** എന്നത് 25 ആകുന്നു.

3.2.1 സ്റ്റാക്ക് പ്രാവർത്തികമാക്കൽ (Implementation of stack)

അറെ ഉപയോഗിച്ച് സ്റ്റാക്ക് പ്രാവർത്തികമാക്കാം എന്ന് നേരത്തെ പരാമർശിച്ചിരുന്നു. അങ്ങനെയെങ്കിൽ ഒരു സ്റ്റാക്ക് ഉപയോഗിച്ച് പ്രതിനിധീകരിക്കാവുന്ന അംഗങ്ങളുടെ എണ്ണത്തിന് പരിധിയുണ്ട്. മാത്രമല്ല അത് അറെയുടെ വലുപ്പത്തിനനുസരിച്ചായിരിക്കും. തുടക്കത്തിൽ സ്റ്റാക്ക് ശൂന്യമാണ് എന്നതു സൂചിപ്പിക്കുന്നതിനായി, TOS ന്റെ വില 1 ആയി നൽകുന്നു. അറെയുടെ സൂചികയുടെ ഏറ്റവും ഉയർന്ന വില എത്തുന്നത് വരെ, സ്റ്റാക്കിലേക്ക് ഓരോ അംഗവും കൂട്ടിച്ചേർക്കപ്പെടുമ്പോൾ TOS ന്റെ വില ഒന്നു വച്ച് കൂടുന്നു. N അംഗങ്ങളുള്ള Top എന്ന അറെ ഉപയോഗിച്ച് സ്റ്റാക്ക് പ്രവർത്തികമാക്കുകയാണെങ്കിൽ, TOS ന്റെ വില 0 മുതൽ (N-1) വരെ മാറി വരാം. അതുപോലെ സ്റ്റാക്കിലെ അംഗങ്ങളെ സൂചിപ്പിക്കാൻ STACK[0], STACK[1], STACK[2], ..., STACK[N-1] മുതലായ പദങ്ങൾ ഉപയോഗിക്കുന്നു.

3.2.2 സ്റ്റാക്കിലെ പ്രവർത്തനങ്ങൾ (Operations on stack)

സ്റ്റാക്ക് ഡാറ്റ സ്ട്രക്ചറുകൾ അറെ ഉപയോഗിച്ചാണ് പ്രാവർത്തികമാക്കിയിരിക്കുന്നതെങ്കിലും അറെക്കു ബാധകമായിട്ടുള്ള എല്ലാ പ്രവർത്തനങ്ങളും സ്റ്റാക്കിൽ അതേ രൂപത്തിൽ ഉപയോഗിക്കുന്നില്ല. ഉദാഹരണത്തിന് അറെയിൽ ഏതു സ്ഥാനത്തു വേണമെങ്കിലും ഉൾപ്പെടുത്തൽ, നീക്കം ചെയ്യൽ എന്നീ പ്രവർത്തനങ്ങൾ ചെയ്യാവുന്നതാണ്. എന്നാൽ സ്റ്റാക്കിൽ മുകളിൽ നിന്ന് മാത്രമേ ഈ പ്രവർത്തനങ്ങൾ ചെയ്യാൻ സാധിക്കൂ. സ്റ്റാക്കിൽ നിർവഹിക്കുന്ന ഉൾപ്പെടുത്തൽ, നീക്കം ചെയ്യൽ പ്രവർത്തനങ്ങളെ യഥാക്രമം പുഷ് (***push***) എന്നും പോപ്പ് (***pop***) എന്നും പറയുന്നു. ഈ പ്രവർത്തനങ്ങൾ എങ്ങനെയാണ് ചെയ്യുന്നത് എന്ന് നമുക്ക് നോക്കാം.

a. പുഷ് പ്രവർത്തനം (Push operation)

സ്റ്റാക്കിലേക്കു ഒരു പുതിയ അംഗത്തെ ഉൾപ്പെടുത്തുന്ന പ്രക്രിയയെയാണ് പുഷ് പ്രവർത്തനം എന്ന് പറയുന്നത്. സ്റ്റാക്കിന്റെ നിർമാണം എന്ന് പറയുന്നത് പുഷ് പ്രവർത്തനത്തിന്റെ ആവർത്തിച്ചുള്ള പ്രയോഗം കൊണ്ടാണ് സാധ്യമാകുന്നത്.

നമുക്ക് ചെയ്യാം

പുഷ് പ്രവർത്തനം നടപ്പിലാക്കുമ്പോഴുള്ള സ്റ്റാക്കിന്റെ സ്ഥിതിയാണ് ചിത്രം 3.5 ൽ കാണിച്ചിരിക്കുന്നത്. സ്റ്റാക്ക് പ്രാവർത്തികമാക്കുന്നതിനായി ഒരു അറെ നിർമിച്ചതായും TOS-1 ആയും നമുക്ക് അനുമാനിക്കാം. ചിത്രം നിരീക്ഷിച്ചു പ്രവർത്തനം നിർവഹിക്കാനുള്ള നടപടിക്രമങ്ങൾ എഴുതുക.

ചിത്രം 3.5: തുടർച്ചയായ പുഷ്പ്രവർത്തനത്തിന് ശേഷമുള്ള സ്റ്റാക്കിന്റെ അവസ്ഥ

ചിത്രം 3.5 തുടർച്ചയായ പുഷ് പ്രവർത്തനത്തിന് ശേഷമുള്ള സ്റ്റാക്കിന്റെ അവസ്ഥ സ്റ്റാക്കിന് മേലുള്ള പുഷ് പ്രവർത്തനത്തിനു വേണ്ടി താഴെ പറയുന്ന ഘട്ടങ്ങൾ നിർവചിക്കാൻ കഴിയുന്നുണ്ടോ എന്ന് പരിശോധിക്കുക.

ഘട്ടം 1: സ്റ്റാക്കിലേക്കു ഉൾപ്പെടുത്തുവാനുള്ള വില ഒരു വേരിയബിളിലേക്കു സ്വീകരിക്കുക.

ഘട്ടം 2: Top ന്റെ വില ഒന്ന് വച്ച് കൂട്ടുക.

ഘട്ടം 3: TOS ന്റെ സ്ഥാനത്തെ വില സംഭരിക്കുക.

സ്റ്റാക്കിന്റെ ശൂന്യമായ സ്ഥലം ഉണ്ടെങ്കിൽ മാത്രമേ പുതിയ അംഗങ്ങളെ ഉൾപ്പെടുത്താനുള്ള മുകളിൽ പറഞ്ഞിരിക്കുന്ന ഘട്ടങ്ങളുടെ പ്രവർത്തനങ്ങൾ സാധ്യമാകുകയുള്ളൂ. ചിത്രം 3.5 ൽ കാണിച്ചിരിക്കുന്ന സ്റ്റാക്കിൽ TOS ന്റെ വില 9 ആയിക്കഴിഞ്ഞാൽ പുതിയ അംഗത്തെ ഉൾപ്പെടുത്താൻ കഴിയില്ല. സ്റ്റാക്ക് നിറയുകയും പുതിയ ഒരു അംഗത്തെ ഉൾപ്പെടുത്താൻ നാം ശ്രമിക്കുകയും ചെയ്താൽ ഉടലെടുക്കുന്ന സാഹചര്യത്തെ (***stack overflow***) എന്ന് പറയുന്നു. സ്റ്റാക്കിലെ പുഷ് പ്രവർത്തനത്തിനുള്ള അൽഗോരിതം നമുക്ക് നോക്കാം.

സ്റ്റാക്കിന് മേൽ പുഷ് പ്രവർത്തനം ചെയ്യാനുള്ള അൽഗോരിതം

N പരമാവധി വലുപ്പമുള്ള STACK[N] എന്ന അറെ പരിഗണിക്കുക. സ്റ്റാക്കിന്റെ മുകൾഭാഗത്തിന്റെ സ്ഥാനം സൂചിപ്പിക്കുന്നതിനായി TOS എന്ന വേരിയബിളും ഉണ്ട്. VAL എന്ന വേരിയബിളിൽ ഡാറ്റ സൂക്ഷിച്ചിരിക്കുന്നു. VAL-നെ സ്റ്റാക്കിലേക്ക് ചേർക്കണം. എന്ന നിർദേശത്തിനും അവസാനിപ്പിക്കുന്ന നിർദേശത്തിനും ഇടയിലായി പുഷ് പ്രവർത്തനത്തിനാവശ്യമായ ഘട്ടങ്ങൾ നൽകിയിരിക്കുന്നു.

```
തുടങ്ങുക
1:   അഥവാ (TOS < N-1) ആണെങ്കിൽ //സ്റ്റാക്ക് ശൂന്യമാണോ എന്ന് പരിശോധിക്കുന്നു
2:        TOS = TOS + 1
3:        STACK[TOS] = VAL
4:   അല്ല എങ്കിൽ
```

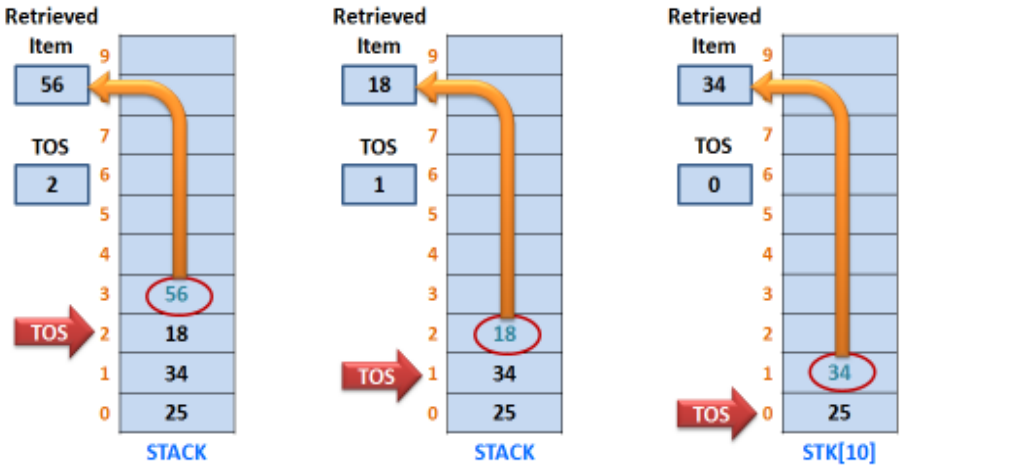
5: 'സ്റ്റാക്ക് ഓവർ ഫ്ളോ' എന്ന് പ്രിൻറ് ചെയ്യുക
6: പുറത്തേക്കു പോകുക
അവസാനിപ്പിക്കുക

a. പോപ്പ് പ്രവർത്തനം (Pop operation)

ഒരു സ്റ്റാക്കിന്റെ ഏറ്റവും മുകളിലുള്ള അംഗത്തെ നീക്കം ചെയ്യുന്ന പ്രക്രിയയെയാണ് പോപ്പ് പ്രവർത്തനം എന്ന് പറയുന്നത്. ഓരോ പോപ്പ് പ്രവർത്തനത്തിന് ശേഷവും TOS ന്റെ വില ഒന്ന് വച്ച് കുറയുന്നു.

നമുക്ക് ചെയ്യാം

ഒരു സ്റ്റാക്കിന്റെ ഏറ്റവും മുകളിലുള്ള അംഗത്തെ നീക്കം ചെയ്യാനുള്ള ഘട്ടങ്ങൾ നിർവചിക്കാൻ നമുക്ക് ശ്രമിക്കാം. ഇതിനായി ചിത്രം 3.6 തന്നിരിക്കുന്നു. ചിത്രം നിരീക്ഷിച്ചു പോപ്പ് ഓപ്പറേഷന് വേണ്ടിയുള്ള ഘട്ടങ്ങൾ കണ്ടുപിടിച്ചു എഴുതുക.



അംഗത്തെ തിരിച്ചെടുത്തശേഷം TOS 2 ആയി മാറുന്നു

അംഗത്തെ തിരിച്ചെടുത്തശേഷം TOS 1 ആയി മാറുന്നു

അംഗത്തെ തിരിച്ചെടുത്തശേഷം TOS 0 ആയി മാറുന്നു

ചിത്രം 3.6: തുടർച്ചയായ പോപ്പ് പ്രവർത്തനത്തിന് ശേഷമുള്ള സ്റ്റാക്കിന്റെ അവസ്ഥ

സ്റ്റാക്കിനു മേലുള്ള പോപ്പ് പ്രവർത്തനത്തിനുവേണ്ടി താഴെ പറയുന്ന ഘട്ടങ്ങൾ നിർമിക്കാൻ കഴിയുന്നുണ്ടോ എന്ന് പരിശോധിക്കുക

ഘട്ടം 1: TOS ന്റെ സ്ഥാനത്തുള്ള അംഗത്തിന്റെ വില ഒരു വേരിയബിളിൽ സംഭരിക്കുക.
ഘട്ടം 2: TOS ന്റെ വില ഒന്ന് കുറയ്ക്കുക.

സ്റ്റാക്കിൽ അംഗങ്ങളുണ്ടെങ്കിൽ മുകളിൽ പറഞ്ഞ രണ്ടു ഘട്ടങ്ങൾ പ്രവർത്തിക്കുന്നു. അറെ ഉപയോഗിച്ചുള്ള സ്റ്റാക്കിൽ, പോപ്പ് പ്രവർത്തനം നടത്തുമ്പോൾ യഥാർഥത്തിൽ അംഗങ്ങളെ നീക്കം ചെയ്യുന്നില്ല. മറിച്ച് TOS ന്റെ വില കുറച്ചുകൊണ്ട് അത്തരം അംഗങ്ങളെ ഉപയോഗിക്കുന്നതിൽ നിന്നും തടസ്സപ്പെടുത്തുന്നു. ഏറ്റവും അവസാനത്തെ അംഗം നീക്കം ചെയ്യപ്പെടുന്നതുവരെ ചിത്രം 3.6 ൽ കൊടുത്തിരിക്കുന്ന സ്റ്റാക്കിൽ പോപ്പ് പ്രവർത്തനം ചെയ്യാവുന്നതാണ്. ഏറ്റവും അവസാനത്തെ അംഗം നീക്കം ചെയ്യപ്പെട്ടു കഴിഞ്ഞാൽ TOS

ന്റെ വില -1 ആയി മാറുന്നു. ഇപ്പോൾ സ്റ്റാക്ക് ശൂന്യമാണ്. ഇങ്ങനെ ശൂന്യമായ സ്റ്റാക്കിൽ നിന്നും ഒരു അംഗത്തെ നീക്കം ചെയ്യാൻ ശ്രമിക്കുകയാണെങ്കിൽ ഉടലെടുക്കുന്ന പ്രതിസന്ധിയെ സ്റ്റാക്ക് അണ്ടർ ഫ്ളോ (***stack underflow***) എന്ന് പറയുന്നു. ഇനി നമുക്ക് സ്റ്റാക്കിന് മേലുള്ള പോപ്പ് പ്രവർത്തനത്തിന്റെ അൽഗോരിതം എഴുതി നോക്കാം.

സ്റ്റാക്കിന് മേലുള്ള പോപ്പ് പ്രവർത്തനത്തിനുള്ള അൽഗോരിതം

പരമാവധി N അംഗങ്ങളെ സംഭരിക്കാവുന്ന ഒരു സ്റ്റാക്കിനെ പ്രാവർത്തികമാക്കാൻ വേണ്ടി STACK[N] എന്ന ഒരു അറെ പരിഗണിക്കുക. സ്റ്റാക്കിലെ ഏറ്റവും മുകളിലെ സ്ഥാനത്തിന്റെ വിവരം സൂക്ഷിക്കുന്നതിനായി TOS എന്നൊരു വേരിയബിൾ ഉപയോഗിക്കുന്നു. സ്റ്റാക്കിൽ നിന്നും നീക്കം ചെയ്യുന്ന അംഗങ്ങളെ സൂക്ഷിക്കുന്നതിനായി VAL എന്ന വേരിയബിൾ ഉപയോഗിക്കുന്നു. പോപ്പ് പ്രവർത്തനത്തിനുള്ള നിർദേശങ്ങൾ തുടങ്ങുക, അവസാനിക്കുക എന്നിവയ്ക്കിടയിൽ കൊടുത്തിരിക്കുന്നു.

തുടങ്ങുക

1: അഥവാ (TOS > -1) ആണെങ്കിൽ //സ്റ്റാക്ക് ശൂന്യമാണോ എന്ന് പരിശോധിക്കുന്നു (അണ്ടർ ഫ്ളോ)
2: VAL = STACK[TOS]
3: TOS = TOS - 1
4: അല്ല എങ്കിൽ
5: 'സ്റ്റാക് അണ്ടർ ഫ്ളോ' എന്ന് പ്രിന്റ് ചെയ്യുക
6: പുറത്തേക്കു പോകുക

അവസാനിപ്പിക്കുക

സ്റ്റാക്ക് പ്രവർത്തങ്ങൾക്ക് വേണ്ടിയുള്ള *C++* ഫങ്ഷനുകൾ

`tos`, `n` എന്നിവ ഗ്ലോബൽ വേരിയബിൾ ആണെന്ന് കരുതുക

പുഷ് പ്രവർത്തനം

```
void push(int stack[],int val)
{
    if (tos < n-1)
    {
        tos++;
        stack[tos]=val;
    }
    else
        cout<<"Overflow";
}
```

പോപ്പ് പ്രവർത്തനം

```
int pop(int stack[])
{
    int val;
    if (tos > -1)
    {
        val=stack[tos];
        tos--;
    }
    else
        cout<<"Underflow";
    return val;
}
```

TOS, N എന്നീ വേരിയബിളുകൾ ഗ്ലോബൽ വേരിയബിളുകളായി പരിഗണിച്ചിരിക്കുന്നു.

സ്റ്റാക്കിന്റെ ഉപയോഗം

സ്റ്റാക്കുകൾ LIFO തത്ത്വത്തെ പിന്തുടരുന്നതിനാൽ സ്ട്രിങ്ങിനെ തിരിച്ചെഴുതുവാനും, പോളിഷ് സ്ട്രിങ് നിർമിക്കാനും മറ്റും ഉപയോഗിക്കുന്നു. സ്ട്രിങ്ങിനെ തിരിച്ചെഴുതുക എന്ന് പറഞ്ഞാൽ തന്നിരിക്കുന്ന സ്ട്രിങ്ങിലെ അക്ഷരങ്ങളെ അവസാനത്തേതിൽ നിന്നും ആദ്യത്തേതിലേക്ക് എന്ന ക്രമത്തിൽ ഒരു സ്ട്രിങ് രൂപപ്പെടുത്തുക എന്നാണ്. ഉദാഹരണത്തിന് "SAD" എന്ന സ്ട്രിങ്ങിനെ തിരിച്ചെഴുതുകയാണെങ്കിൽ "DAS" എന്ന് ലഭിക്കുന്നു. പോളിഷ് സ്ട്രിങ് എന്നാൽ ഓപ്പറന്റുകൾക്കു മുമ്പോ ശേഷമോ ഓപ്പറേറ്ററുകൾ വരുന്ന തരത്തിലുള്ള ഒരു ഗണിത പ്രയോഗശൈലിയാണ്. ഉദാഹരണത്തിന് A+B എന്നത് AB+ എന്നോ +AB എന്നോ ആക്കി മാറ്റാവുന്നതാണ്. A+B എന്നതിനെ ഇൻഫിക്സ് (infix) പ്രയോഗശൈലി എന്ന് പറയുന്നു. AB+, +AB എന്നിവ യഥാക്രമം പോസ്റ്റ്ഫിക്സ് (postfix) എന്നും പ്രീഫിക്സ് (prefix) പ്രയോഗശൈലി എന്നും അറിയപ്പെടുന്നു. ഇവ ALU വിന്റെ പ്രവർത്തനത്തിന് ആവശ്യമാണ്. ഇൻഫിക്സിനെ മറ്റു രണ്ടു രൂപത്തിലേക്ക് വിവർത്തനം ചെയ്യാൻ പുഷ്, പോപ്പ് പ്രവർത്തനങ്ങളാണ് കമ്പ്യൂട്ടർ ഉപയോഗിക്കുന്നത്. മാത്രമല്ല ഈ പ്രയോഗശൈലികളെ വിലയിരുത്താനും ALU സ്റ്റാക്കുകളെ ആശ്രയിക്കുന്നു.

3.3 ക്യൂ (Queue)

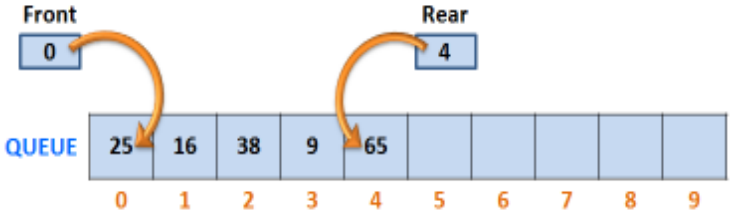
വരികൾ നമുക്ക് സുപരിചിതങ്ങളാണ്. പല സാഹചര്യങ്ങളിലും നമ്മൾ വരികളുടെ ഭാഗമാകാറുമുണ്ട്. ചിത്രം 3.7 ൽ പോളിങ് സ്റ്റേഷനിലെ വരിയാണ് സൂചിപ്പിച്ചിരിക്കുന്നത്. ഇവിടെ വരിയുടെ മുമ്പിലുള്ള ആദ്യത്തെ വോട്ടർ വോട്ട് ചെയ്യുന്നു. പുതിയ വ്യക്തി വരിയിൽ ചേരുമ്പോൾ ഏറ്റവും പുറകിലായിട്ടാണ് വന്നു നിൽക്കുക. ഈ ഉദാഹരണത്തിൽ നിന്നും വ്യക്തമാകുന്നത്, ആദ്യം വരിയിൽ നിൽക്കുന്ന വ്യക്തിയാണ് വരിയിൽ നിന്നും ആദ്യം പുറത്തു പോകുക. ഈ തരത്തിൽ ഒരു കൂട്ടം ഇനങ്ങളെ ക്രമീകരിക്കുന്നതിനെ ഫസ്റ്റ് ഇൻ ഫസ്റ്റ് ഔട്ട് (FIFO) തത്ത്വം എന്ന് പറയുന്നു. അതിനാൽ FIFO തത്ത്വത്തിന്റെ അടിസ്ഥാനത്തിൽ പ്രവർത്തിക്കുന്ന ഡാറ്റ സ്ട്രക്ചറിനെ ക്യൂ (Queue) എന്ന് വിളിക്കുന്നു. ചിത്രത്തിൽ കണ്ടതു പോലെ ഒരു ക്യൂവിനു രണ്ട് അറ്റങ്ങൾ ഉണ്ട്: മുൻഭാഗവും (Front) പിൻഭാഗവും (Rear). ക്യൂവിൽ പുതിയ ഡാറ്റ അംഗങ്ങളെ ഉൾപ്പെടുത്തുന്നത് പിൻഭാഗത്തും നീക്കം ചെയ്യുന്നത് മുൻഭാഗത്തും ആയിരിക്കും. സ്റ്റാക്കിൽ പറഞ്ഞത് പോലെ ക്യൂവും ഒരു യുക്തിപരമായ ആശയം മാത്രമാണ്. ഒരു അറെ ഉപയോഗിച്ചാണ് ക്യൂ പ്രാവർത്തികമാക്കുന്നതെങ്കിൽ അത്തരത്തിലുള്ള ക്യൂ (സ്റ്റാറ്റിക്) സ്വഭാവമുള്ളതായിരിക്കും.

ചിത്രം 3.7: പോളിങ് സ്റ്റേഷനിലെ വരി

3.3.1 ക്യൂവിനെ പ്രവർത്തികമാക്കൽ (Implementation of Queue)

ഒരു അറെ ഉപയോഗിച്ച് ക്യൂ പ്രായോഗികമാക്കുകയാണെങ്കിൽ, അതിൽ ഉൾകൊള്ളിക്കാവുന്ന അംഗങ്ങൾക്ക് ഒരു പരിധിയുണ്ട്. Queue[10] എന്ന അറെ ഉപയോഗിച്ച്

നടപ്പിലാക്കിയിരിക്കുന്ന ഒരു ക്യൂ ആണ് ചിത്രം 3.8 ൽ കാണുന്നത്. ഇതിന് പരമാവധി 10 അംഗങ്ങളെ മാത്രമേ ഉൾക്കൊള്ളാനാകൂ. ഈ ചിത്രത്തിൽ പ്രകാരം 0 മുതൽ 4 വരെ സ്ഥാനങ്ങളിലായി അഞ്ച് അംഗങ്ങളാണ് ക്യൂവിലുള്ളത്. അതായത് QUEUE 0 വും Rear 4 ഉം ആയിരിക്കും. ക്യൂവിലെ ആദ്യത്തെ അംഗത്തെ QUEUE[Front] എന്നും അവസാനത്തെ അംഗത്തെ QUEUE[Rear] എന്നും സൂചിപ്പിക്കാവുന്നതാണ്.



ചിത്രം 3.8: അറെ ഉപയോഗിച്ച് പ്രവർത്തികമാക്കിയ ക്യൂ

അറെയുടെ അവസാന സൂചികയായ 9 ആണ് ക്യൂവിന് അനുവദനീയമായിട്ടുള്ള ഏറ്റവും ഉയർന്ന വില. തുടക്കത്തിൽ മുൻഭാഗത്തിന്റെയും (Rear) പിൻഭാഗത്തിന്റെയും (Front) വില -1 ആയിരിക്കും. ഇത് ക്യൂ ശൂന്യമാണെന്നതിനെ സൂചിപ്പിക്കുന്നു. ക്യൂ വിലേക്കു ആദ്യത്തെ അംഗത്തെ ഉൾപ്പെടുത്തുമ്പോൾ ഈ രണ്ടു വിലകളും 0 ആയി മാറുന്നു. തുടർന്ന് ഓരോ പുതിയ അംഗത്തെ ഉൾപ്പെടുത്തുമ്പോഴും Rear ന്റെ വില ഒന്ന് വച്ചു കൂടുന്നു. ഏറ്റവും ഉയർന്ന സൂചിക (ഇവിടെ 9) എത്തുന്നത് വരെ ഇത് തുടർന്ന് പോകാവുന്നതാണ്. അത് പോലെ ഓരോ അംഗത്തെ നീക്കം ചെയ്യുമ്പോഴും Front ന്റെ വില ഒന്നുവച്ച് കൂടുന്നതാണ്. Front ന്റെ വില Rear നെക്കാൾ കൂടുതലാകുന്നതു വരെ ഇത് തുടരാവുന്നതാണ്.

3.3.2 ക്യൂവിലെ പ്രവർത്തനങ്ങൾ (Operations on queue)

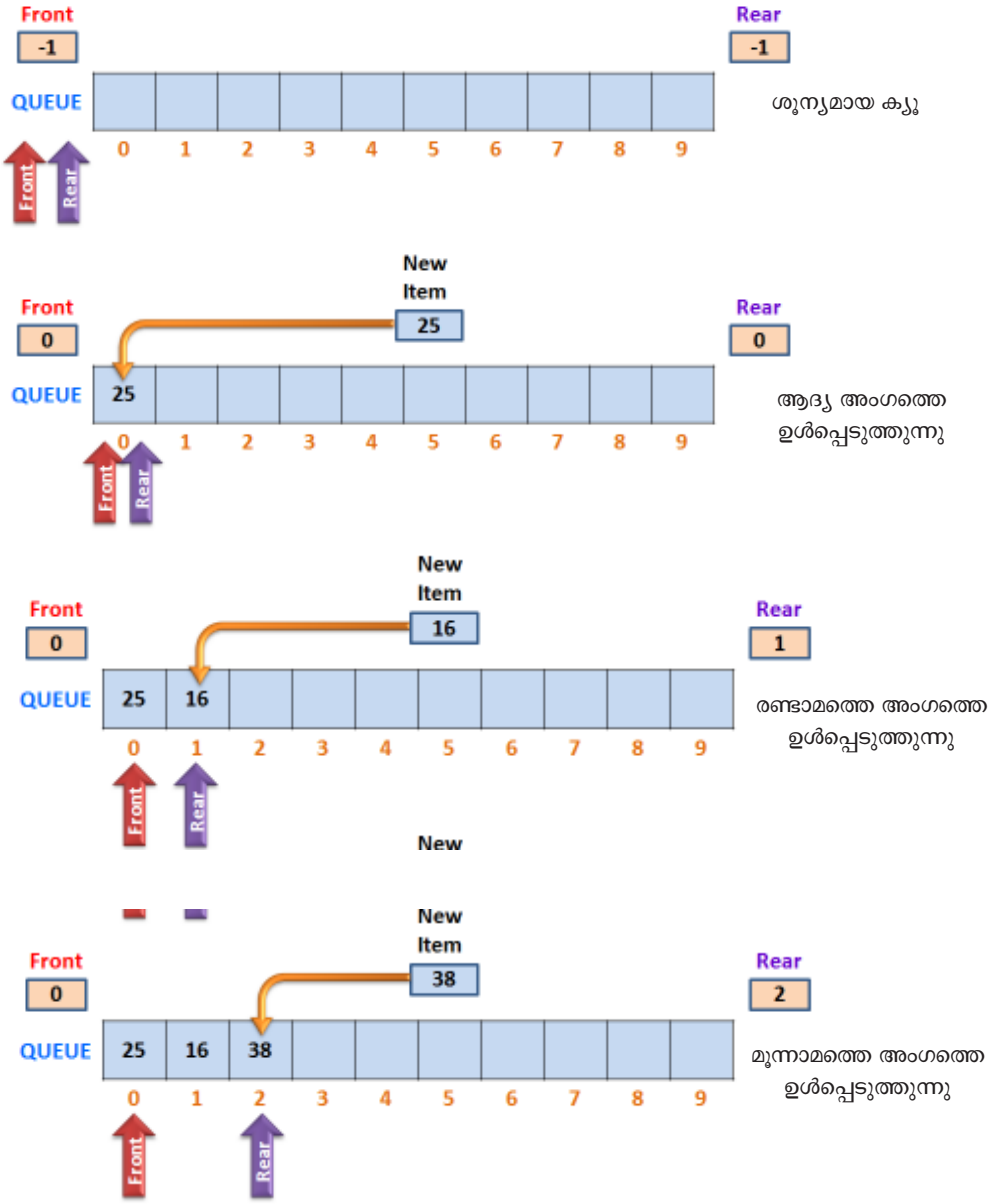
സ്റ്റാക്കിലേതു പോലെ ക്യൂവിലും ഉൾപ്പെടുത്തൽ, നീക്കം ചെയ്യൽ പ്രവർത്തനങ്ങൾക്കു ചില നിയന്ത്രണങ്ങളുണ്ട്. ഒരു സാധാരണ അറെയിൽ ഉൾപ്പെടുത്തൽ, നീക്കം ചെയ്യൽ എന്നീ പ്രവർത്തനങ്ങൾ ഏതു സ്ഥാനത്തു വേണമെങ്കിലും ചെയ്യാവുന്നതാണ്. എന്നാൽ ഒരു സ്റ്റാക്കിൽ ഇത് ഏറ്റവും മുകളിൽ നിന്ന് മാത്രമായിരിക്കും. അതുപോലെ ക്യൂവിൽ ഉൾപ്പെടുത്തൽ, നീക്കം ചെയ്യൽ പ്രവർത്തനങ്ങൾ രണ്ട് അറ്റത്തു മാത്രമായിരിക്കും നടക്കുക.

a. ഉൾപ്പെടുത്തൽ പ്രവർത്തനം (Insertion operation)

ക്യൂവിന്റെ പിൻഭാഗത്തു പുതിയ അംഗത്തെ കൂട്ടിച്ചേർക്കുന്ന പ്രക്രിയയെയാണ് ഉൾപ്പെടുത്തൽ എന്ന് പറയുന്നത്. അടുത്ത മെമ്മറി സ്ഥാനത്തെ സൂചിപ്പിക്കാൻ Rear ന്റെ വില ആദ്യം കൂട്ടുകയും പിന്നീട് ആ സ്ഥാനത്തു അംഗത്തെ ഉൾപ്പെടുത്തുകയും ചെയ്യുന്നു. ഉൾപ്പെടുത്തൽ പ്രവർത്തനം തുടർച്ചയായി ചെയ്യുമ്പോഴാണ് ഒരു ക്യൂ രൂപപ്പെടുന്നത്.

നമുക്ക് ചെയ്യാം

ഉൾപ്പെടുത്തൽ പ്രവർത്തനത്തിന് വിധേയമാകുന്ന ഒരു ക്യൂവിന്റെ അവസ്ഥ ചിത്രം 3.9 ൽ കാണിച്ചിരിക്കുന്നു.
ഇതിനായി ഒരു അറെ നിർമ്മിക്കുകയും, Front, Rear എന്നിവയ്ക്ക് -1 എന്ന വില നൽകുകയും ചെയ്യുന്നു ചിത്രം നിരീക്ഷിച്ചു പ്രവർത്തനം ചെയ്യുവാനുള്ള നടപടിക്രമം എഴുതുക.



ചിത്രം 3.9: തുടർച്ചയായ ഉൾപ്പെടുത്തൽ പ്രവർത്തനം നടത്തിയതിന് ശേഷമുള്ള ക്യൂവിന്റെ അവസ്ഥ

ഉൾപ്പെടുത്തൽ പ്രവർത്തനത്തിന് വേണ്ടി താഴെ പറയുന്ന ഘട്ടങ്ങൾ നിർവചിക്കുവാൻ കഴിയുമോ എന്ന് പരിശോധിക്കുക.

ഘട്ടം 1: ക്യൂവിലേക്കു ഉൾപ്പെടുത്തുവാനുള്ള വില ഒരു വേരിയബിളിലേക്കു സ്വീകരിക്കുക

ഘട്ടം 2: Rear ന്റെ വില ഒന്ന് കൂട്ടുക.

ഘട്ടം 3: Rear ന്റെ സ്ഥാനത്ത് വില സംഭരിക്കുക.

ശൂന്യമായ ഒരു ക്യൂവിലേക്കു ആദ്യമായി ഉൾപ്പെടുത്തൽ പ്രവർത്തനം ചെയ്യുമ്പോൾ Front, Rear എന്നിവയുടെ വില ഒന്നുവച്ചു കൂട്ടുന്നു. അതായത് Rear ഉം Front ഉം 0 ആയി മാറുന്നു. അതിനു ശേഷം വരുന്ന ഉൾപ്പെടുത്തലുകൾക്കെല്ലാം Rear മാത്രം കൂട്ടുന്നു. Rear 9 ആകുന്നതു വരെ (അറെ യുടെ അവസാനത്തെ സൂചിക) ഇത് തുടരുന്നതാണ്. അതിനു ശേഷം വീണ്ടും ഉൾപ്പെടുത്താൻ പ്രവർത്തനം ചെയ്യുവാൻ ശ്രമിക്കുകയാണെങ്കിൽ സ്റ്റാക്കിലേതു പോലെ 'ക്യൂ ഓവർ ഫ്ളോ' (***queue overflow***) സംഭവിക്കുന്നു. ക്യൂവിൽ ഉൾപ്പെടുത്തൽ പ്രവർത്തനം നടത്തുവാനുള്ള അൽഗോരിതം നമുക്ക് എഴുതി നോക്കാം.

ക്യൂവിൽ ഉൾപ്പെടുത്തൽ പ്രവർത്തനം ചെയ്യുവാനുള്ള അൽഗോരിതം

ക്യൂ നടപ്പിലാക്കുന്നതിന് വേണ്ടി, N വലുപ്പമുള്ള QUEUE[N] എന്ന ഒരു അറേ പരിഗണിക്കുക. ക്യൂവിന്റെ മുൻഭാഗത്തെയും പിൻഭാഗത്തെയും സൂചിപ്പിക്കുന്നതിനായി FRONT, REAR എന്നീ രണ്ടു വേരിയബിളുകൾ ഉപയോഗിക്കുന്നു. ക്യൂവിലേക്കു ഉൾപ്പെടുത്താനുള്ള ഡാറ്റ VAL എന്ന വേരിയബിളിൽ സംഭരിച്ചു വയ്ക്കുന്നു. തുടങ്ങുക, അവസാനിപ്പിക്കുക എന്നീ നിർദേശങ്ങൾക്കിടയിൽ കൊടുത്തിരിക്കുന്ന ഘട്ടങ്ങൾ ഉൾപ്പെടുത്തൽ പ്രവർത്തനത്തെ വിശദീകരിക്കുന്നു.

തുടങ്ങുക

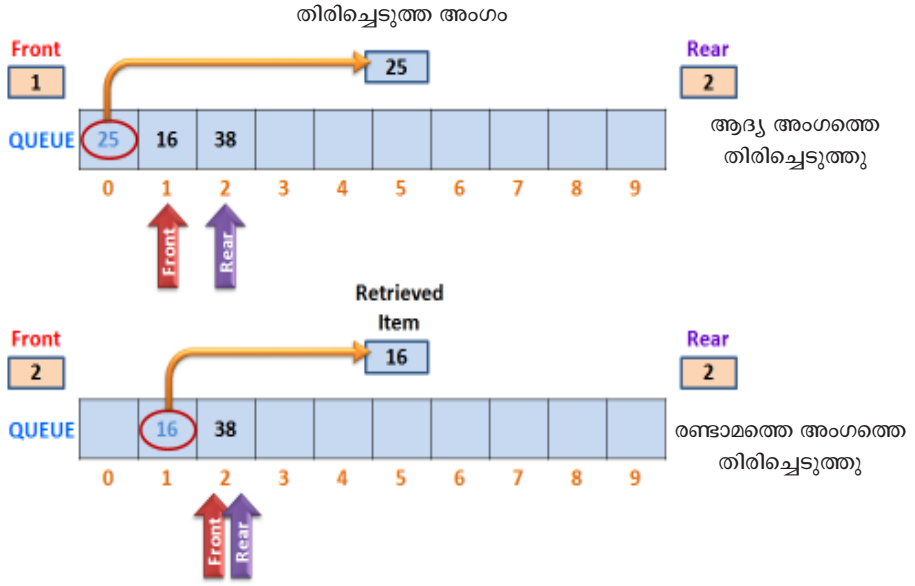
1: അഥവാ (REAR == –1) ആണെങ്കിൽ //സ്ഥലത്തിന്റെ ലഭ്യത പരിശോധിക്കുന്നു
2: FRONT = REAR = 0 'ക്യൂ ഓവർ ഫ്ളോ' പ്രിൻറ് ചെയ്യുക
3: Q[REAR] = VAL പുറത്തേക്കു പോകുക
4: അല്ലെങ്കിൽ അഥവാ (REAR < N-1) ആണെങ്കിൽ // സ്ഥലത്തിന്റെ ലഭ്യത പരിശോധിക്കുന്നു.
5: REAR = REAR + 1
6: Q[REAR] = VAL
7: അല്ലെങ്കിൽ
8: Print "Queue Overflow "
9: പുറത്തേക്കു പോകുക
അവസാനിപ്പിക്കുക

b. നീക്കം ചെയ്യൽ പ്രവർത്തനം (Deletion operation)

ക്യൂ വിന്റെ മുൻഭാഗത്തു നിന്നും അംഗത്തെ ഒഴിവാക്കുന്ന പ്രക്രിയയെയാണ് നീക്കം ചെയ്യൽ എന്ന് പറയുന്നത്. നീക്കം ചെയ്യലിന് ശേഷം Front ന്റെ വില ഒന്നുവച്ചു കൂട്ടുന്നു. അറെ ഉപയോഗിച്ചുള്ള ക്യൂവിൽ നീക്കം ചെയ്യൽ പ്രവർത്തനം നടത്തുമ്പോൾ യഥാർഥത്തിൽ അംഗങ്ങളെ നീക്കം ചെയ്യുന്നില്ല. മറിച്ച് Front ന്റെ വില കൂട്ടിക്കൊണ്ട് അത്തരം അംഗങ്ങളെ ഉപയോഗിക്കുന്നതിൽ നിന്നും തടസ്സപ്പെടുത്തുന്നു.

നമുക്ക് ചെയ്യാം

അറെ ഉപയോഗിച്ച് നടപ്പിലാക്കിയിരിക്കുന്ന ഒരു ക്യൂ പരിഗണിക്കുക. ചിത്രം 3.10 ൽ നീക്കം ചെയ്യൽ പ്രവർത്തനത്തിന് വിധേയമാകുന്ന ക്യൂ വിന്റെ അവസ്ഥകൾ കാണിച്ചിരിക്കുന്നു. ഈ പ്രവർത്തനത്തിനുള്ള വിവിധ ഘട്ടങ്ങൾ നിർവചിക്കാൻ നമുക്ക് ശ്രമിക്കാം.



ചിത്രം 3.10: തുടർച്ചയായ നീക്കം ചെയ്യൽ പ്രവർത്തനം നടത്തിയതിന് ശേഷമുള്ള ക്യൂവിന്റെ അവസ്ഥ

നീക്കം ചെയ്യൽ പ്രവർത്തനത്തിന് താഴെ പറയുന്ന ഘട്ടങ്ങൾ നമുക്ക് നിർവചിക്കാം.

Step 1: Front ന്റെ സ്ഥാനത്തുള്ള അംഗത്തെ ഒരു വേരിയബിളിലേക്കു സ്വീകരിക്കുക.
Step 2: Front ന്റെ വില ഒന്ന് കൂട്ടുക.

ചിത്രം 3.10 പ്രകാരം ക്യൂവിന്റെ മുൻഭാഗത്തേക്ക് അംഗങ്ങളുടെ സ്ഥാനമാറ്റം നടക്കുന്നില്ല. അതായത് നിത്യജീവിതത്തിലെപ്പോലെയല്ല ഇവിടത്തെ ക്യൂവിന്റെ സങ്കൽപ്പം. ക്യൂ ഡാറ്റ സ്ട്രക്ചറിൽ നീക്കം ചെയ്യൽ പ്രവർത്തത്തിൽ അംഗങ്ങളുടെ സ്ഥാനമാറ്റം നടത്തുന്നതിന് പകരം Front ന്റെ വില കൂട്ടുകയാണ് ചെയ്യുന്നത്. ക്യൂവിൽ അംഗങ്ങൾ ള്ളിടത്തോളം കാലം നീക്കം ചെയ്യൽ പ്രവർത്തനത്തിന് മേൽനിർവചിച്ച രണ്ടു ഘട്ടങ്ങൾ മതിയാകുന്നതാണ്. രണ്ടാമത്തെ അംഗത്തിന്റെ നീക്കം ചെയ്യലിന് ശേഷമുള്ള ക്യൂവിന്റെ അവസ്ഥ നോക്കുക. Front ന്റെയും Rear ന്റെയും വില ഒരേ അംഗത്തെ സൂചിപ്പിക്കുന്നു. അതായത് രണ്ട് എന്ന അറെയുടെ സൂചിക. ക്യൂവിൽ ഒരു നീക്കം ചെയ്യൽ പ്രവർത്തനം കൂടി നടത്തുന്നതായി അനുമാനിക്കുക. നടപടി ക്രമം പ്രകാരം Front ന്റെ വില 3 ആയി മാറുന്നു. ഇത് Rear ന്റെ വിലയേക്കാൾ കൂടുതലാണ്. നമുക്കറിയാം ഒരു ക്യൂവിൽ അത് ഉചിതമല്ല. മാത്രമല്ല ക്യൂ ഇപ്പോൾ ശൂന്യമാണെന്നു നമുക്ക് കാണാവുന്നതാണ്. ക്യൂ ശൂന്യമായാൽ Front ന്റെയും Rear ന്റെയും വില-1 ആകണമെന്ന് നേരത്തെ പരാമർശിച്ചതാണ്. അപ്പോൾ അവസാനത്തെ അംഗത്തെ നീക്കം ചെയ്താൽ Front ന്റെയും Rear ന്റെയും വില -1 ആക്കാനുള്ള ഘട്ടം കൂടി അൽഗോരിതത്തിൽ ഉൾപ്പെടുത്തേണ്ടതാണ്. ഈ അവസ്ഥയിൽ വീണ്ടും നീക്കം ചെയ്യൽ അനുവദനീയമല്ല. ശൂന്യമായ ക്യൂവിൽ നിന്നും നീക്കം ചെയ്യാൻ ശ്രമിക്കുമ്പോളുണ്ടാകുന്ന അവസ്ഥയെ 'ക്യൂ അണ്ടർഫ്ലോ' (***queue underflow***) എന്ന് പറയുന്നു. ഇനി നമുക്ക് ക്യൂവിൽ നിന്നും നീക്കം ചെയ്യൽ പ്രവർത്തനം നടത്താനുള്ള അൽഗോരിതം തയാറാക്കാം.

ഒരു ക്യൂവിൽ നിന്ന് നീക്കം ചെയ്യൽ പ്രവർത്തനം നടത്താനുള്ള അൽഗോരിതം

ക്യൂ നടപ്പിലാക്കാൻ വേണ്ടി പരമാവധി N അംഗങ്ങളുള്ള QUEUE[N] എന്ന ഒരു അറെ പരിഗണിക്കുക. FRONT, REAR എന്നീ വേരിയബിളുകൾ ക്യൂവിന്റെ മുൻഭാഗത്തെയും പിൻഭാഗത്തെയും സൂചിപ്പിക്കാൻ ഉപയോഗിക്കുന്നു. ക്യൂവിൽ നിന്നും നീക്കം ചെയ്യുന്ന അംഗത്തെ സംഭരിക്കുന്നതിനായി VAL എന്ന വേരിയബിൾ ഉപയോഗിക്കുന്നു. നീക്കം ചെയ്യൽ പ്രവർത്തനത്തിനുള്ള ഘട്ടങ്ങൾ തുടങ്ങുക, അവസാനിപ്പിക്കുക എന്നീ നിർദേശങ്ങൾക്കിടയിൽ നൽകിയിരിക്കുന്നു.

തുടങ്ങുക

1: അഥവാ (FRONT > −1) ആണെങ്കിൽ // ശൂന്യമായ അവസ്ഥ പരിശോധിക്കുന്നു
2: VAL = Q[FRONT]
3: FRONT = FRONT + 1
4: അല്ലെങ്കിൽ
5: 'ക്യൂ ഓവർ ഫ്ളോ' എന്നു പ്രിൻറ് ചെയ്യുക
6: പുറത്തേക്കു പോകുക
7: അഥവാ (FRONT > REAR) ആണെങ്കിൽ // അവസാനത്തെ അംഗത്തിന്റെ നീക്കം ചെയ്യൽ പരിശോധിക്കുന്നു
8: FRONT = REAR = -1
9: പുറത്തേക്കു പോകുക

അവസാനിപ്പിക്കുക

ക്യൂ പ്രവർത്തങ്ങൾക്ക് വേണ്ടിയുള്ള C++ ഫങ്ഷനുകൾ

`n, front, rear` **എന്നിവ ഗ്ലോബൽ വേരിയബിൾ ആണെന്ന് കരുതുക**

ക്യൂ വിൽ ഉൾപ്പെടുത്തൽ പ്രവർത്തനം

```
void ins_q(int queue[],int val)
{
    if (rear == -1)
    {
        front=0;
        rear=0;
        q[rear]=val;
    {
    else (if rear < n-1)
    {
        rear++;
        q[rear]=val;
    }
    else
        cout<<"Overflow";
}
```

ക്യൂവിൽ നീക്കം ചെയ്യൽ പ്രവർത്തനം

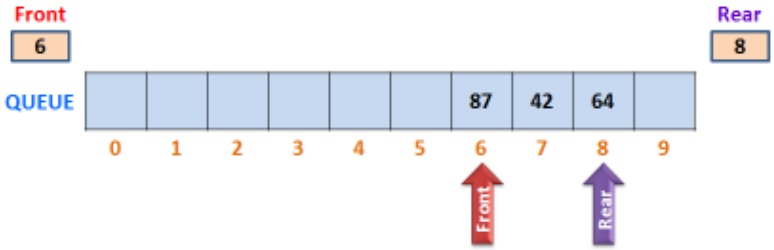
```
int del_q(int queue[])
{
    int val;
    if (front > -1)
    {
        val=q[front];
        front++;
    }
    else
        cout<<"Underflow";
    if (front > rear)
    {
        front= -1;
        rear= -1
    }
    return val;
}
```

ക്യൂവിന്റെ ഉപയോഗം

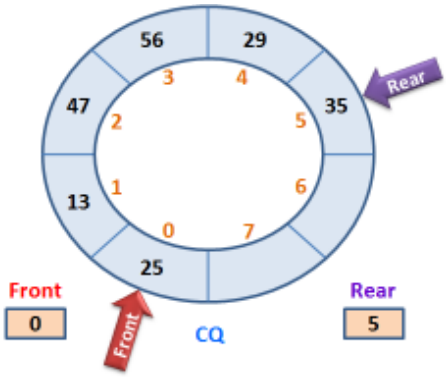
ക്യൂ അധികവും കമ്പ്യൂട്ടർ സയൻസിൽ ജോബ് ആസൂത്രണം ചെയ്യുവാൻ വേണ്ടിയാണ് ഉപയോഗിക്കുന്നത്. മെമ്മറി, പ്രോസസ്സർ, ഫയലുകൾ മുതലായവ ആസൂത്രണം ചെയ്യുന്നതിനാണ് ഓപ്പറേറ്റിംഗ് സിസ്റ്റം ക്യൂ ഉപയോഗിക്കുന്നത്. ഇതിന് ഒരു ഉദാഹരണമാണ് പ്രിൻറ് ക്യൂ. പ്രോസെസ്സറിനെ അപേക്ഷിച്ചു പ്രിന്ററിന് വേഗത കുറവായതിനാൽ പ്രിൻറ് ചെയ്യുവാനുള്ള ജോലികളെല്ലാം പ്രിൻറ് ബഫറിൽ നിക്ഷേപിക്കുന്നു. പ്രിന്റ് ബഫർ FIFO തത്ത്വം പിന്തുടരുന്നതിനാൽ അതിനെ ഒരു ക്യൂ ആയി കണക്കാക്കാം.

3.3.3 വൃത്താകൃതി ക്യൂ (Circular queue)

നമ്മൾ ഇതുവരെ ചർച്ച ചെയ്ത ക്യൂവിനെ രേഖീയ ക്യൂ എന്നാണ് പറയുക. ഇതിലെ അംഗങ്ങൾ ഒരു നേർരേഖയായിട്ട് അല്ലെങ്കിൽ ഒരു നേർവരിയായിട്ടാണ് കാണപ്പെടുക. ഇത്തരം ക്യൂകളുടെ രണ്ടറ്റങ്ങൾ തമ്മിൽ ഒരിക്കലും കൂട്ടിമുട്ടുകയില്ല. രേഖീയ ക്യൂവിൽ ഒരു ന്യൂനതയുണ്ട്. ചിത്രം 3.11 ൽ കൊടുത്തിരിക്കുന്ന, 6 അംഗങ്ങളെ നീക്കം ചെയ്യൽ പ്രവർത്തനങ്ങൾക്കു വിധേയമായ ക്യൂ പരിഗണിക്കുക. നിലവിൽ അതിൽ മൂന്ന് അംഗങ്ങൾ മാത്രമേയുള്ളു. സ്വാഭാവികമായിട്ടും Front ന്റെ വില 6 ഉം Rear ന്റെ വില 8 ഉം ആയിരിക്കും.

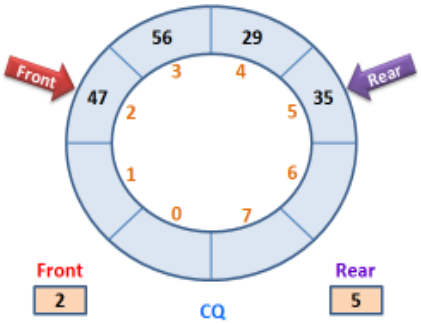


ചിത്രം 3.11: തുടർച്ചയായ 6 അംഗങ്ങളെ നീക്കം ചെയ്തതിനു ശേഷം 3 അംഗങ്ങളുള്ള ക്യൂ



ചിത്രം 3.12: വൃത്താകൃതി ക്യൂ

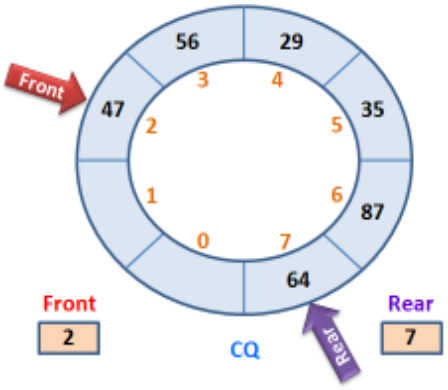
ആദ്യത്തെ 6 സ്ഥാനങ്ങൾ ശൂന്യമാണെങ്കിലും ഉൾപ്പെടുത്തൽ അൽഗോരിതം പ്രകാരം നമുക്ക് ഒരു അംഗത്തെ മാത്രമേ കൂട്ടിച്ചേർക്കാൻ കഴിയൂ. അവസാനത്തെ സ്ഥാനത്ത് ഒരു അംഗം മാത്രമേ ഉള്ളു എന്ന് കരുതുക. അങ്ങനെയെങ്കിൽ പുതിയതായി ഒരു അംഗത്തെ കൂട്ടിച്ചേർക്കുവാൻ ശ്രമിക്കുകയാണെങ്കിൽ 'ക്യൂ ഓവർ ഫ്ളോ' (Queue Overflow) സംഭവിക്കുന്നു. രേഖീയ ക്യൂവിന്റെ ഈ പരിമിതി വൃത്താകൃതി ക്യൂ ഉപയോഗിച്ച് പരിഹരിക്കാവുന്നതാണ്. ചിത്രം 3.12 ൽ കാണിച്ചിരിക്കുന്നത് പോലെ രണ്ടറ്റങ്ങളും കൂട്ടി മുട്ടുന്ന ഒരു ക്യൂ ആണിത്.



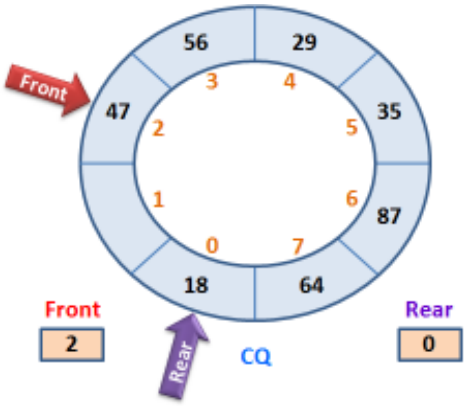
ചിത്രം 3.13: നീക്കം ചെയ്യൽ പ്രവർത്തനത്തിന് ശേഷമുള്ള വൃത്താകൃതി ക്യൂ

ചിത്രം 3.13 ൽ രണ്ട് അംഗങ്ങളെ നീക്കംചെയ്യൽ പ്രവർത്തനങ്ങൾ നടത്തി എന്ന് അനുമാനിക്കുക. അപ്പോൾ Front ന്റെ വില 2 ഉം ചിത്രം 3.13(a) ൽ കാണിച്ചിരിക്കുന്നത് പോലെ 4 ശൂന്യമായ സ്ഥാനങ്ങളും സ്ഥലങ്ങളും

നമുക്ക് ലഭിക്കുന്നു. ഈ സ്ഥാനങ്ങളുടെ സൂചികകളുടെ വില യഥാക്രമം 6, 7, 0, 1 എന്നിങ്ങനെയായിരിക്കും. നമ്മൾ വീണ്ടും രണ്ട് അംഗങ്ങളെ കൂട്ടിച്ചേർക്കുകയാണെങ്കിൽ Rear ന്റെ വില 7 ആയി മാറുന്നു. ചിത്രം 3.13(b) ൽ കാണിച്ചിരിക്കുന്നത് പോലെ 0,1 എന്നീ സൂചികകളുടെ സ്ഥാനം ഇപ്പോഴും ശൂന്യമായിട്ടാണുള്ളത്. അതിനാൽ വീണ്ടും ഉൾപ്പെടുത്തൽ പ്രവർത്തനം ഇവിടെ നിർവഹിക്കാവുന്നതാണ്. ഇത്തവണ Rear ന്റെ വില 0 ആയി നിശ്ചയിക്കുകയും അതിനു ശേഷം ഉൾപ്പെടുത്താൻ പ്രവർത്തനം നിർവഹിക്കുകയും ചെയ്യാവുന്നതാണ്. ക്യൂവിന്റെ ഈ അവസ്ഥയെ ചിത്രം 3.13(c) ൽ കാണിച്ചിരിക്കുന്നു.



ചിത്രം 3.13 (b): Rear ന് അതിന്റെ ഉയർന്ന വില



ചിത്രം 3.13(c): Rear ന് അതിന്റെ താഴ്ന്ന വില

നമുക്ക് വിലയിരുത്താം

1 .ഡാറ്റാ സ്ട്രക്ചർ എന്നാലെന്ത്?

2. ഡാറ്റ ക്രമീകരണത്തിനായി സ്റ്റാക്കിൽ ഉപയോഗിക്കുന്ന തത്ത്വം _____ ആകുന്നു.

3 .FIFO എന്ന തത്ത്വം ഉപയോഗപ്പെടുത്തുന്ന ഡാറ്റാസ്ട്രക്ചറിന്റെ പേരെഴുതുക.

4 .അറെയിലെ അണ്ടർഫ്ളോ (underflow) എന്നാലെന്ത്?

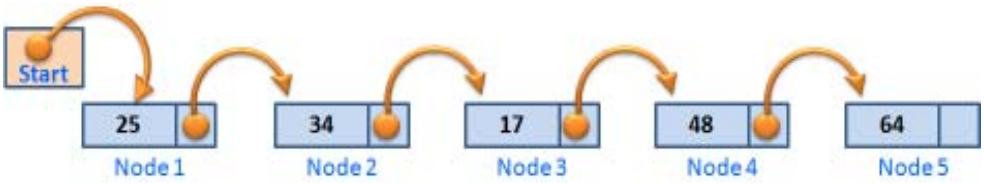
5. സ്റ്റാക്കിന്റെ ഏത് അംഗമാണ് നീക്കം ചെയ്യാൻ സാധിക്കുക (ആദ്യത്തേത് അല്ലെങ്കിൽ അവസാനത്തേത്)?

3.4 ലിങ്ക്ഡ് ലിസ്റ്റ് (Linked list)

അംഗങ്ങളുടെ എണ്ണത്തിന് പരിമിതിയില്ലാത്ത ഡാറ്റ അംഗങ്ങളുടെ ശേഖരത്തെയാണ് ലിങ്ക്ഡ് ലിസ്റ്റ് എന്ന് പറയുന്നത്. അറെകൾ ഉപയോഗിച്ച് പ്രാവർത്തികമാക്കുന്ന യുക്തിപരമായ ആശയങ്ങൾ മാത്രമാണ് കഴിഞ്ഞ ഭാഗങ്ങളിൽ വിശദീകരിച്ച സ്റ്റാക്ക്, ക്യൂ മുതലായ ഡാറ്റാസ്ട്രക്ചറുകൾ. അതിനാൽ ഇവ (സ്റ്റാറ്റിക്) ഡാറ്റാസ്ട്രക്ചറുകളാണ്. എന്നാൽ ലിങ്ക്ഡ് ലിസ്റ്റ് ഒരു (ഡൈനാമിക്) ഡാറ്റാസ്ട്രക്ചർ ആണ്. പുതിയ ഡാറ്റ ഇനങ്ങൾ കൂട്ടിച്ചേർക്കുമ്പോൾ ഇത് വളരുകയും നീക്കം ചെയ്യുമ്പോൾ ചുരുങ്ങുകയും ചെയ്യുന്നു. ലിങ്ക്ഡ് ലിസ്റ്റിന് മുഴുവൻ അംഗങ്ങൾക്കും ആവശ്യമായ മെമ്മറി ആദ്യമേ

അനുവദിക്കുകയില്ല. ലിസ്റ്റിലേക്ക് ഒരു പുതിയ അംഗത്തെ ഉൾപ്പെടുത്തുന്നതിന് തൊട്ടു മുമ്പാണ് ആ അംഗത്തിന് ആവശ്യമായ മെമ്മറി അനുവദിക്കുന്നത്. അതുകൊണ്ടാണ് ഇതിനെ (ഡൈനാമിക്) ഡാറ്റാ സ്ട്രക്ചറുകളായി കരുതുന്നത്. അറെ അധിഷ്ഠിത ഡാറ്റാ സ്ട്രക്ചറുകളുമായി ലിങ്ക്ഡ് ലിസ്റ്റിനുള്ള മറ്റൊരു വ്യത്യാസം ലിങ്ക്ഡ് ലിസ്റ്റിലെ അംഗങ്ങൾ മെമ്മറിയിൽ പല ഭാഗത്തായിട്ടാണ് സംഭരിക്കപ്പെടുന്നത്. എന്നാൽ അവ പോയിന്ററുകളുടെ സഹായത്തോടെ പരസ്പരം ബന്ധിപ്പിച്ചിരിക്കുന്നു. നമ്മൾ അധ്യായം ഒന്നിൽ പഠിച്ചത് പ്രകാരം പോയിന്റർ എന്നാൽ മെമ്മറി സ്ഥാനങ്ങളുടെ അഡ്രസ് സംഭരിച്ചു വച്ചിരിക്കുന്ന വേരിയബിളാണ്. അപ്പോൾ ഒരുകാര്യം വ്യക്തമാണ് ലിങ്ക്ഡ് ലിസ്റ്റിലെ ഒരു അംഗത്തിന് ഡാറ്റയും അഡ്രസ്സും ഉണ്ടായിരിക്കും. ലിങ്ക്ഡ് ലിസ്റ്റിലെ ഒരു അംഗത്തെ നോഡ് (Node) എന്ന് വിളിക്കുന്നു. നോഡിൽ അടങ്ങിയിരിക്കുന്ന അഡ്രസ്സിനെ ലിങ്ക് (Link) എന്ന് പറയുന്നു.

ഒരു ഡാറ്റയും ലിങ്കും (ലിസ്റ്റിലെ അടുത്ത നോഡിലേക്കുള്ള പോയിന്റർ) അടങ്ങിയ നോഡുകളുടെ ശേഖരമാണ് ലിങ്ക്ഡ് ലിസ്റ്റ്. അതായത് ലിസ്റ്റിലെ ആദ്യത്തെ നോഡിൽ ആദ്യത്തെ ഡാറ്റ അംഗവും രണ്ടാമത്തെ നോഡിന്റെ അഡ്രസ്സും അടങ്ങിയിരിക്കുന്നു. രണ്ടാമത്തെ നോഡിൽ രണ്ടാമത്തെ ഡാറ്റ അംഗവും മൂന്നാമത്തെ നോഡിന്റെ അഡ്രസ്സും അടങ്ങിയിരിക്കുന്നു. ഇപ്രകാരം ലിസ്റ്റ് തുടരുന്നു. അവസാനത്തെ നോഡിൽ അവസാനത്തെ ഡാറ്റയും ഒരു നൾ പോയിന്ററും (Null Pointer) അടങ്ങിയിരിക്കുന്നു. എവിടെയും പോയിൻറ് ചെയ്യാത്ത പോയിന്ററാണ് Null Pointer. അങ്ങനെയെങ്കിൽ ആദ്യത്തെ നോഡിന്റെ അഡ്രസ് എവിടെയാണുണ്ടാകുക? ആദ്യത്തെ നോഡിന്റെ അഡ്രസ്സ് അടങ്ങിയ ഒരു പ്രത്യേക പോയിന്റർ എല്ലാ ലിങ്ക്ഡ് ലിസ്റ്റിലും ഉണ്ടായിരിക്കും. ഇതിനെ സ്റ്റാർട്ട് (Start) അല്ലെങ്കിൽ ഹെഡർ (Header) എന്നു പറയുന്നു. അഞ്ച് അംഗങ്ങൾ അടങ്ങിയ ഒരു ലിങ്ക്ഡ് ലിസ്റ്റിനെ ചിത്രം 3.14 ൽ പ്രദർശിപ്പിച്ചിരിക്കുന്നു.



ചിത്രം 3.14: നോഡുകളുള്ള ലിങ്ക്ഡ് ലിസ്റ്റ്

ഡാറ്റയായി ഒരു സംഖ്യയും പോയിന്റർ ആയി അടുത്ത നോഡിലേക്കു ചൂണ്ടിക്കാണിക്കുന്ന ലിങ്കും അടങ്ങിയ നോഡുകൾ ആണ് ചിത്രത്തിൽ കാണിച്ചിരിക്കുന്നത്. എല്ലാ നോഡുകളുടെയും വലുപ്പം സമമായിരിക്കും. അതായത് ഓരോ നോഡിനു വേണ്ടിയും മാറ്റി വയ്ക്കുന്ന മെമ്മറി സമമായിരിക്കും.

3.4.1 ലിങ്ക്ഡ് ലിസ്റ്റ് പ്രാവർത്തികമാക്കൽ (Implementation of linked list)

ഓരോ നോഡിനും മെമ്മറി അനുവദിക്കുന്ന നോഡുകളുടെ ശേഖരമാണ് ലിങ്ക്ഡ് ലിസ്റ്റ് എന്ന് നമ്മൾ നേരത്തെ കണ്ടു കഴിഞ്ഞു. നോഡിനു വേണ്ടിയുള്ള മെമ്മറിയിൽ ഏറ്റവും കുറഞ്ഞത് രണ്ടു തരത്തിൽപ്പെട്ട ഡാറ്റായാണ് അടങ്ങിയിരിക്കുന്നത്, അതിൽ ഒന്ന് അംഗത്തിന്റെ യഥാർഥ ഡാറ്റയും രണ്ടാമത്തേത് അടുത്ത നോഡിലേക്കുള്ള പോയിന്ററും

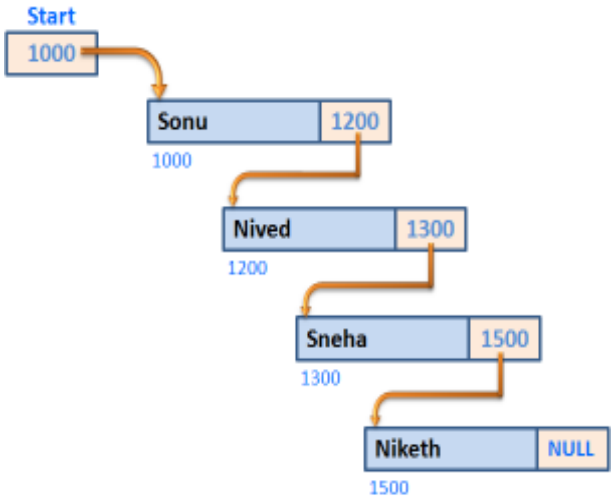
ആയിരിക്കും. വിവിധ തരത്തിൽപ്പെട്ട ഡാറ്റകൾ അടങ്ങിയ ഉപയോക്തൃ നിർവചിതമായ ഡാറ്റ ടൈപ്പ് ആണ് സ്ട്രക്ചർ എന്നത് നാം അധ്യായം ഒന്നിൽ കണ്ടതാണ്. സ്ട്രക്ചറിലെ അംഗം ഒരു പോയിന്റർ ആകാം എന്ന് മാത്രമല്ല അതേ സ്ട്രക്ചറിലേക്കു സൂചിപ്പിക്കുന്ന പോയിന്ററുമാകാം. അത്തരം സ്ട്രക്ചറുകളെ സ്വയം സൂചിതസ്ട്രക്ചറുകൾ (Self Referential Sructure) എന്ന് പറയുന്നു. അതായത് സ്വയം സൂചിത സ്ട്രക്ചറുകളുടെ സഹായത്തോടെയാണ് ലിങ്ക്ഡ് ലിസ്റ്റുകൾ നിർമ്മിക്കുന്നത്. താഴെ പറയുന്ന സ്വയം സൂചിതസ്ട്രക്ചർ ഉപയോഗിച്ച് ചിത്രം 3.14 ൽ കാണിച്ചിരിക്കുന്ന ലിങ്ക്ഡ് ലിസ്റ്റിന്റെ നോഡുകൾ രൂപകൽപ്പന ചെയ്തിരിക്കുന്നു.

```
struct Node
{
    int data;
    Node *link;
};
```

സ്ട്രക്ചറിന്റെ പേര് Node ആണെന്നും സ്ട്രക്ചറിലെ രണ്ടാമത്തെ അംഗം അതേ സ്ട്രക്ചറിന്റെ തരത്തിലുള്ള ഒരു പോയിന്റർ ആണെന്നും നമുക്ക് കാണാം. താഴെ പറയുന്ന പ്രസ്താവന ഉപയോഗിച്ച് ആദ്യത്തെ നോഡിനെ സൂചിപ്പിക്കുന്ന **Start** എന്ന പ്രത്യേക പോയിന്റർ നിർമിക്കാവുന്നതാണ്.

```
struct Node * Start;        or        Node *Start;
```

ചിത്രം 3.15 ൽ സ്ട്രിങ്ങുകളുടെ ഒരു ലിങ്ക്ഡ് ലിസ്റ്റ് ആണ് കാണിച്ചിരിക്കുന്നത്. ഇതിലെ നോഡുകളുടെ ഡാറ്റാസ്ട്രിങ്ങുകൾ ഉപയോഗിച്ചും ലിങ്കുകൾ മറ്റു നോഡുകളുടെ അഡ്രസ് ഉപയോഗിച്ചും നിറച്ചിരിക്കും. ശ്രദ്ധിക്കേണ്ട കാര്യം ഇവിടെ അഡ്രസ്സുകൾ എന്നത് അനുമാനിച്ചിരിക്കുന്ന ചില സംഖ്യകൾ ആണ്. ഈ നോഡുകളെ താഴെ പറയുന്ന സ്ട്രക്ചർ ഉപയോഗിച്ച് പ്രതിനിധീകരിക്കാവുന്നതാണ്.



ചിത്രം 3.15: ലിങ്ക്ഡ് ലിസ്റ്റ് പ്രാവർത്തികമാക്കൽ

```
struct Node
{
    char data[10];
    Node *link;
};
```

പ്രവർത്തന സമയത്ത് ആദ്യത്തെ നോഡിന് 1000 എന്ന അഡ്രസ്സുള്ള മെമ്മറി സ്ഥാനം നീക്കിവയ്ക്കുന്നതായി അനുമാനിക്കുക. അതിനാൽ Start പോയിന്ററിന്റെ വില 1000 ആയിരിക്കും. ഒന്നാമത്തെ നോഡിന്റെ ഡാറ്റ ഭാഗത്തു "Sonu" എന്ന സ്ട്രിങ് സംഭരിച്ചിരിക്കുന്നു. രണ്ടാമത്തെ നോഡിന് 1200 എന്ന മെമ്മറി അഡ്രസ് ആണ് നൽകിയിരിക്കുന്നത്. അതിന്റെ ഡാറ്റാഭാഗത്തു "Nived" എന്ന വില സംഭരിച്ചിരിക്കുന്നു. രണ്ടാമത്തെ നോഡ് ആയതിനാൽ ആദ്യത്തെ നോഡിന്റെ ലിങ്ക് ഭാഗത്ത് ഈ നോഡിന്റെ അഡ്രസ് സംഭരിച്ചിരിക്കുന്നു. ചിത്രത്തിൽ കാണിച്ചിരിക്കുന്നത് പോലെ അവസാനത്തെ നോഡിൽ ലിങ്കിന്റെ സ്ഥാനത്ത് NULL പോയിൻറർ ആയിരിക്കും ഉണ്ടായിരിക്കുക.

3.4.2 ലിങ്ക്ഡ് ലിസ്റ്റിലെ പ്രവർത്തനങ്ങൾ (Operations on linked list)

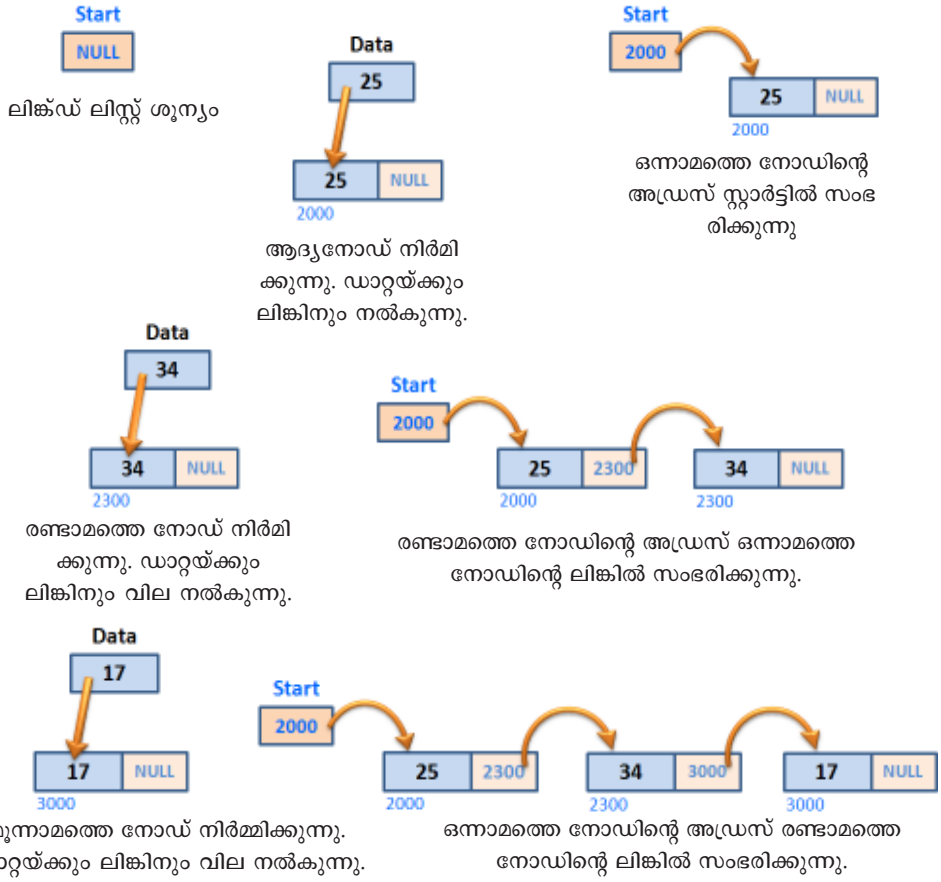
ഭാഗം 3.1.2 ൽ പ്രതിപാദിച്ചിരിക്കുന്ന എല്ലാ പ്രവർത്തനങ്ങളും ഉപാധികളില്ലാതെ ലിങ്ക്ഡ് ലിസ്റ്റിൽ ചെയ്യുവാൻ സാധിക്കുന്നതാണ്. പക്ഷേ നിർമാണം, കടന്നുപോകൽ, ഉൾപ്പെടുത്തൽ, നീക്കം ചെയ്യൽ എന്നീ പ്രവർത്തനങ്ങളെപ്പറ്റി മാത്രമേ നമ്മൾ ഇവിടെ ചർച്ച ചെയ്യുന്നുള്ളൂ. ഉപരിപഠന സമയത്തു ബാക്കി പ്രവർത്തനങ്ങളെപ്പറ്റി നിങ്ങൾ പഠിക്കുന്നതാണ്.

a. ലിങ്ക്ഡ് ലിസ്റ്റിന്റെ നിർമാണം (Creation of linked list)

തുടക്കത്തിൽ നമുക്ക് സ്വയം സൂചകമായി ഒരു സ്ട്രക്ചർ നിർവചിക്കേണ്ടതുണ്ട്. Start അല്ലെങ്കിൽ Header എന്ന ഒരു പോയിന്റർ വേരിയബിളിനെ നിർവചിച്ച് നൾ (NULL) എന്ന വില നൽകുന്നു. ഇനി ആവശ്യകതയനുസരിച്ച് നോഡുകൾക്കുള്ള മെമ്മറി അസ്ഥിരമായി നീക്കിവച്ചുകൊണ്ട് ലിങ്ക്ഡ് ലിസ്റ്റ് നിർമിക്കാം.

നമുക്ക് ചെയ്യാം

ലിങ്ക്ഡ് ലിസ്റ്റ് നിർമാണ സമയത്തെ അവസ്ഥയാണ് ചിത്രം 3.16 ൽ കാണിച്ചിരിക്കുന്നത്. Node എന്ന പേരുള്ള ഒരു സ്വയം സൂചിത സ്ട്രക്ചർ നിർമിച്ചതായും Start എന്ന NULL തരത്തിൽപ്പെട്ട പോയിന്ററിനു Node എന്ന വില നൽകിയതായും നമുക്ക് അനുമാനിക്കാം. ഇനി താഴെ പറയുന്ന ചിത്രങ്ങൾ നിരീക്ഷിച്ച് ലിങ്ക്ഡ് ലിസ്റ്റ് നിർമാണത്തിനാവശ്യമായ നടപടിക്രമം തയാറാക്കുക.



ചിത്രം 3.16: ലിങ്ക്ഡ് ലിസ്റ്റ് നിർമാണത്തിന്റെ തുടർച്ചയായ പ്രവർത്തനങ്ങൾ

താഴെ പറയുന്ന ഘട്ടങ്ങൾ വികസിപ്പിക്കുവാൻ സാധിക്കുമോ എന്ന് പരിശോധിക്കുക.

ഘട്ടം 1: ഒരു നോഡ് നിർമിച്ച് അതിന്റെ മെമ്മറി അഡ്രസ് ലഭ്യമാക്കുക.

ഘട്ടം 2: ഡാറ്റയും NULL എന്ന വിലയും നോഡിൽ സംഭരിക്കുക.

ഘട്ടം 3: അത് ആദ്യത്തെ നോഡ് ആണെങ്കിൽ അതിന്റെ അഡ്രസ് node ൽ സംഭരിക്കുക.

ഘട്ടം 4: അത് ആദ്യത്തെ നോഡ് അല്ലെങ്കിൽ അതിന്റെ അഡ്രസ് തൊട്ടു മുമ്പത്തെ നോഡിന്റെ ലിങ്കിൽ സംഭരിക്കുക.

ഘട്ടം 5: ഉപയോക്താവിന്റെ ആവശ്യാനുസരണം ഘട്ടം 1 മുതൽ 4 വരെ ആവർത്തിക്കുക.

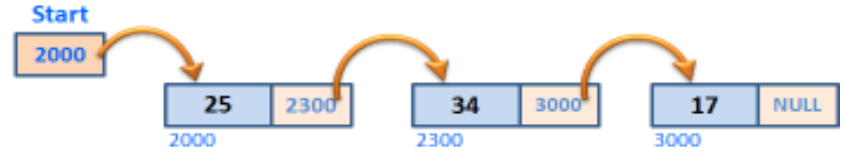
യഥാർഥത്തിൽ ലിങ്ക്ഡ് ലിസ്റ്റിന്റെ നിർമാണം എന്നത് ലിങ്ക്ഡ് ലിസ്റ്റിന്റെ അവസാന ഭാഗത്ത് ആവർത്തിച്ചു നടത്തുന്ന ഉൾപ്പെടുത്തൽ പ്രവർത്തനം ആണ്. രണ്ടാമത്തെ നോഡ് മുതൽ ഉൾപ്പെടുത്തൽ പ്രവർത്തനം ചെയ്യണമെങ്കിൽ നിലവിലുള്ള ലിസ്റ്റിലെ അവസാനത്തെ അംഗത്തിന്റെ അഡ്രസ് ലഭിക്കുന്നതിനാവശ്യമായ കടന്നുപോകൽ പ്രവർത്തനം നടത്തണം. അതിനാൽ കടന്നുപോകൽ പ്രവർത്തനത്തെപ്പറ്റി നമുക്ക് വിശദമായി ചർച്ച ചെയ്യാം.

b. ലിങ്ക്ഡ് ലിസ്റ്റിൽ കടന്നുപോകൽ പ്രവർത്തനം (Traversing a linked list)

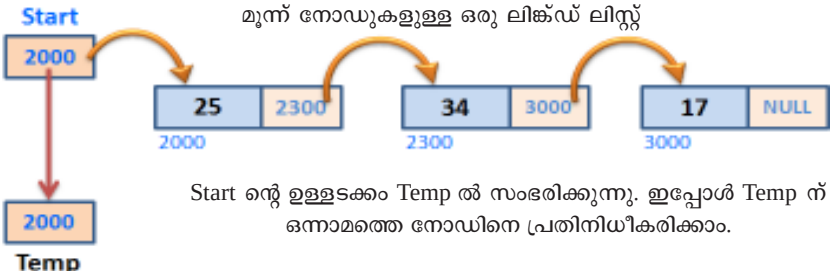
നമുക്കറിയാം ഒരു ഡാറ്റ സ്ട്രക്ചറിൽ കടന്നുപോകൽ എന്നാൽ അതിലുള്ള എല്ലാ അംഗങ്ങളെയും സന്ദർശിക്കലാണ്. ലിങ്ക്ഡ് ലിസ്റ്റുകളുടെ കാര്യത്തിൽ യാത്ര ചെയ്യൽ ആരംഭിക്കുന്നത് ആദ്യത്തെ നോഡ് മുതലാണ്. ആദ്യത്തെ നോഡിന്റെ അഡ്രസ് *arrow* എന്ന പോയിന്ററിൽ നിന്നും ലഭിക്കുന്നു. ഇതുപയോഗിച്ച് ആരോ (**->**) ഓപ്പറേറ്ററിന്റെ സഹായത്തോടെ നമുക്ക് ഡാറ്റ ഭാഗത്തെ ലഭ്യമാക്കാം. ശേഷം രണ്ടാമത്തെ നോഡിന്റെ അഡ്രസ് അടങ്ങിയ ആദ്യത്തെ നോഡിന്റെ ലിങ്ക് ഭാഗത്തെ ഉപയോഗപ്പെടുത്തുന്നു. ഈ അഡ്രസ് ഉപയോഗിച്ച് രണ്ടാമത്തെ നോഡിലെ ഡാറ്റയും ലിങ്കും ലഭ്യമാക്കുന്നു. നോഡിന്റെ ലിങ്കിൽ NULL ലഭിക്കുന്നത് വരെ ഈ പ്രക്രിയ തുടരുന്നു.

നമുക്ക് ചെയ്യാം

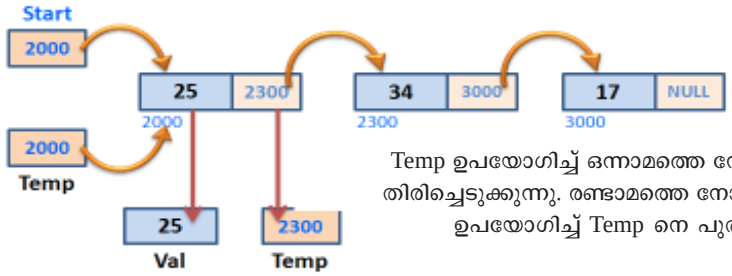
ചിത്രം 3.17 ൽ നൽകിയിരിക്കുന്ന ലിങ്ക്ഡ് ലിസ്റ്റ് നിരീക്ഷിച്ച് യാത്ര ചെയ്യൽ പ്രവർത്തനത്തിന് വേണ്ട ഘട്ടങ്ങൾ നമുക്ക് നിർവചിക്കാം. Node തരത്തിൽപ്പെട്ട ഒരു പോയിന്റർ ആണ് TEMP എന്നും VAL എന്നത് നോഡിൽ നിന്നുമുള്ള ഡാറ്റ സംഭരിക്കാനാവശ്യമായ ഒരു വേരിയബിൾ ആണെന്നും കരുതുക.



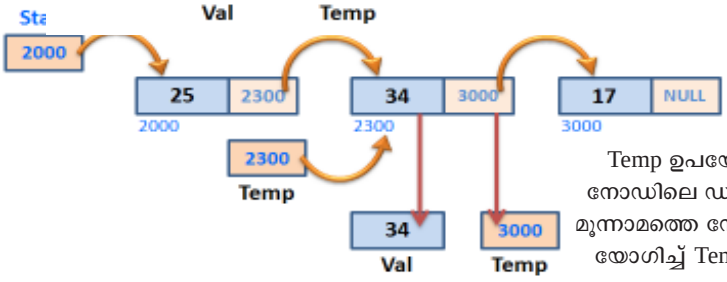
മൂന്ന് നോഡുകളുള്ള ഒരു ലിങ്ക്ഡ് ലിസ്റ്റ്



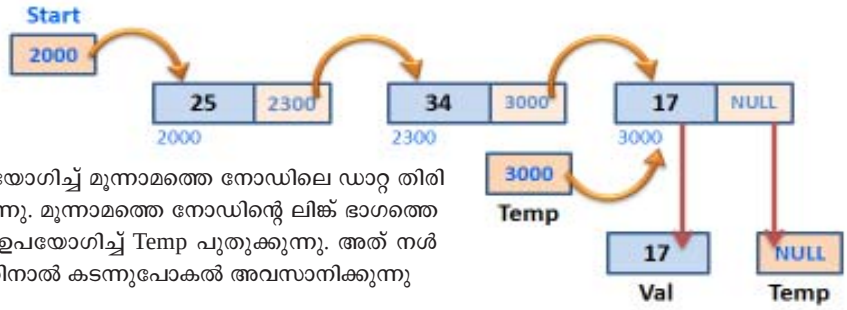
Start ന്റെ ഉള്ളടക്കം Temp ൽ സംഭരിക്കുന്നു. ഇപ്പോൾ Temp ന് ഒന്നാമത്തെ നോഡിനെ പ്രതിനിധീകരിക്കാം.



Temp ഉപയോഗിച്ച് ഒന്നാമത്തെ നോഡിലെ ഡാറ്റാ തിരിച്ചെടുക്കുന്നു. രണ്ടാമത്തെ നോഡിന്റെ അഡ്രസ്സ് ഉപയോഗിച്ച് Temp നെ പുതുക്കുന്നു.



Temp ഉപയോഗിച്ച് രണ്ടാമത്തെ നോഡിലെ ഡാറ്റ തിരിച്ചെടുക്കുന്നു. മൂന്നാമത്തെ നോഡിന്റെ അഡ്രസ് ഉപയോഗിച്ച് Temp നെ പുതുക്കുന്നു.



Temp ഉപയോഗിച്ച് മൂന്നാമത്തെ നോഡിലെ ഡാറ്റ തിരിച്ചെടുക്കുന്നു. മൂന്നാമത്തെ നോഡിന്റെ ലിങ്ക് ഭാഗത്തെ അഡ്രസ് ഉപയോഗിച്ച് Temp പുതുക്കുന്നു. അത് നൾ ആയതിനാൽ കടന്നുപോകൽ അവസാനിക്കുന്നു

ചിത്രം 3.17: ലിങ്ക്ഡ് ലിസ്റ്റിലെ കടന്നു പോകൽ പ്രവർത്തനം

താഴെ പറയുന്ന ഘട്ടങ്ങൾ നിങ്ങൾക്ക് നിർവചിക്കാൻ കഴിയുന്നുണ്ടോ എന്ന് പരിശോധിക്കുക.

ഘട്ടം 1: ആദ്യത്തെ നോഡിന്റെ അഡ്രസ് Start ൽ നിന്നും Temp ലേക്ക് സംഭരിക്കുക.

ഘട്ടം 2: Temp ലെ അഡ്രസ് ഉപയോഗിച്ച് ആദ്യത്തെ നോഡിന്റെ ഡാറ്റ Val ലിൽ സംഭരിക്കുക.

ഘട്ടം 3: ഈ നോഡിന്റെ ലിങ്ക് ഭാഗത്തെ അഡ്രസ് (അതായത് അടുത്ത നോഡിന്റെ അഡ്രസ്) Temp ലേക്ക് സ്വീകരിക്കുക.

ഘട്ടം 4: Temp ലെ അഡ്രസ് NULL അല്ലെങ്കിൽ ഘട്ടം 2 ലേക്ക് പോകുക അല്ലെങ്കിൽ അവസാനിപ്പിക്കുക.

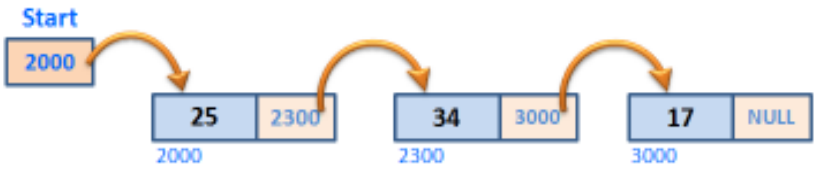
Start ന്റെ വില NULL അല്ലാത്ത സാഹചര്യത്തിൽ ഒരു ലിങ്ക്ഡ് ലിസ്റ്റിനെ നിർമിക്കാൻ മേൽപ്പറഞ്ഞ ഘട്ടങ്ങൾ ഉപയോഗിക്കുന്നു. ഇത്തരം സാഹചര്യത്തിൽ അവസാനത്തെ നോഡിന്റെ അഡ്രസ് കണ്ടുപിടിക്കേണ്ടതുണ്ട്. എന്തിനെന്നാൽ പുതിയ നോഡിന്റെ അഡ്രസ് അതിന്റെ ലിങ്ക് ഭാഗത്താണ് സംഭരിക്കേണ്ടത്. Start ൽ നിന്നും ആദ്യത്തെ നോഡിന്റെ അഡ്രസ് സ്വീകരിച്ചുകൊണ്ടാണ് യാത്രചെയ്യൽ പ്രവർത്തനം ആരംഭിക്കുന്നത്. ഈ അഡ്രസ് താൽക്കാലിക പോയിന്റർ വേരിയബിളിലേക്ക് പകർത്തുകയും (ചിത്രത്തിൽ Temp) പിന്നീട് Temp ചൂണ്ടുന്ന നോഡിലെ ലിങ്ക് ഭാഗത്തുള്ള അഡ്രസ് പകർത്തിയെടുത്തു. താൽക്കാലിക പോയിൻറ്റർ വേരിയബിൾ പുതുക്കപ്പെടുകയും ചെയ്യുന്നു. Temp ചൂണ്ടുന്ന നോഡിന്റെ ലിങ്ക് ഭാഗത്തു നൾ വില ലഭിക്കുന്നത് വരെ ഈ സന്ദർശനം തുടർന്ന് കൊണ്ടേയിരിക്കുന്നു.

c. ലിങ്ക്ഡ് ലിസ്റ്റിലെ ഉൾപ്പെടുത്തൽ പ്രവർത്തനം (Insertion in a linked list)

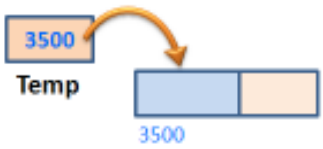
ലിങ്ക്ഡ് ലിസ്റ്റിൽ ഒരു അംഗത്തെ ഉൾപ്പെടുത്തൽ എന്നാൽ ആ അംഗം ഉൾപ്പെടുന്ന നോഡിനെ നിശ്ചിതസ്ഥാനത്തു സ്ഥാപിക്കുന്ന പ്രക്രിയയാണ്. അറെയിലേതു പോലെ

നമുക്ക് ചെയ്യാം

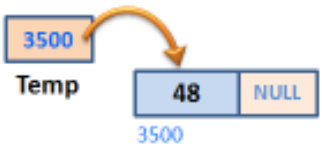
ആരംഭത്തിൽ മൂന്ന് നോഡുകളുള്ള ഒരു ലിങ്ക്ഡ് ലിസ്റ്റിന്റെ മൂന്നാം സ്ഥാനത്തേക്കു നോഡിനെ ഉൾപ്പെടുത്താനുള്ള പ്രവർത്തനങ്ങളാണ് ചിത്രം 3.18 ൽ കാണിച്ചിരിക്കുന്നത്. ചിത്രം നിരീക്ഷിച്ചു ലിങ്ക്ഡ് ലിസ്റ്റിലേക്ക് ഒരു നോഡിനെ ഉൾപ്പെടുത്താനുള്ള പ്രവർത്തനത്തിന്റെ ഘട്ടങ്ങൾ നമുക്ക് നിർവചിക്കാം. Node തരത്തിൽപ്പെട്ട പോയിന്ററുകളാണ് Temp, PreNode, PostNode എന്നിവ എന്നും POS നോഡിനെ ഉൾപ്പെടുത്തേണ്ട സ്ഥാനം സംഭരിച്ചിരിക്കുന്ന വേരിയബിൾ ആണെന്നും നമുക്ക് അനുമാനിക്കാം.



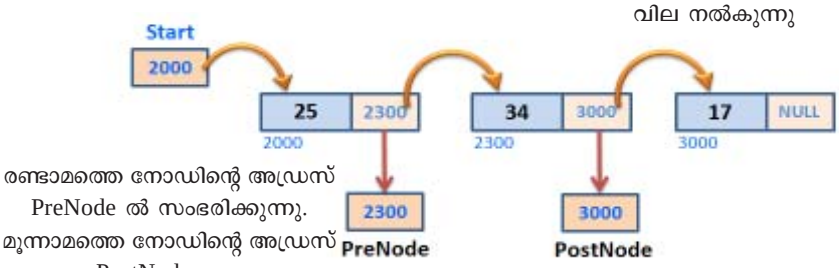
മൂന്ന് നോഡുള്ള ഒരു ലിങ്ക്ഡ് ലിസ്റ്റ്



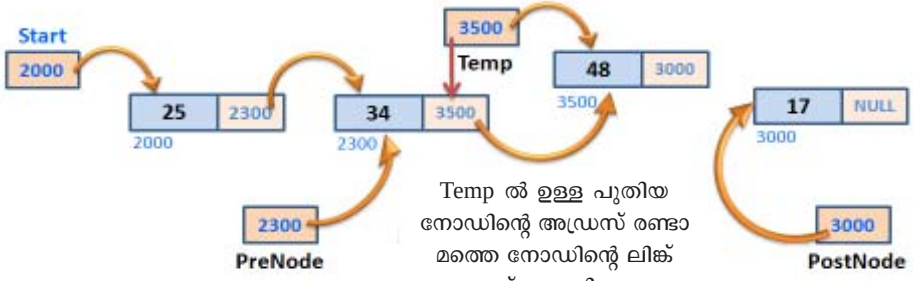
പുതിയ നോഡ് നിർമിച്ച് അഡ്രസ് Temp ൽ സംഭരിക്കുന്നു



പുതിയ നോഡിന്റെ ഡാറ്റാ ഭാഗത്തിനും ലിങ്ക് ഭാഗത്തിനും വില നൽകുന്നു

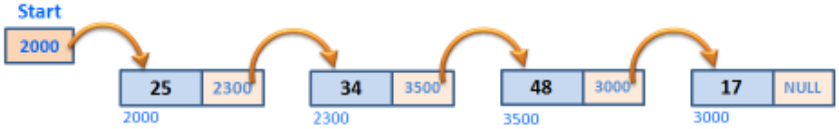


രണ്ടാമത്തെ നോഡിന്റെ അഡ്രസ് PreNode ൽ സംഭരിക്കുന്നു. മൂന്നാമത്തെ നോഡിന്റെ അഡ്രസ് PostNode ലും



Temp ൽ ഉള്ള പുതിയ നോഡിന്റെ അഡ്രസ് രണ്ടാമത്തെ നോഡിന്റെ ലിങ്ക് ഭാഗത്ത് സംഭരിക്കുന്നു.

PreNode ൽ ഉള്ള നാലാമത്തെ നോഡിന്റെ അഡ്രസ് പുതിയ നോഡിന്റെ ലിങ്ക് ഭാഗത്തും സംഭരിക്കുന്നു.



മൂന്നാമത്തെ സ്ഥാനത്ത് പുതിയ നോഡ് ഉൾക്കൊള്ളിച്ച ശേഷം ഉള്ള അവസ്ഥ

ചിത്രം 3.18: ലിങ്ക്ഡ് ലിസ്റ്റിലെ ഉൾപ്പെടുത്തൽ പ്രവർത്തനം

ലിങ്ക്ഡ് ലിസ്റ്റിൽ നോഡിനെ എവിടെ വേണമെങ്കിലും ഉൾപ്പെടുത്താവുന്നതാണ്. തുടക്കത്തിലോ, അവസാനത്തിലോ അല്ലെങ്കിൽ രണ്ടു നോഡുകൾക്കിടയിലോ ആകാം.

ഉൾപ്പെടുത്തൽ പ്രവർത്തനത്തിന് താഴെ പറയുന്ന ഘട്ടങ്ങൾ നിർവചിക്കാം:

ഘട്ടം 1: നോഡ് നിർമിച്ച് അതിന്റെ അഡ്രസ് Temp ൽ സംഭരിക്കുന്നു.

ഘട്ടം 2: Temp ഉപയോഗിച്ച് ഈ നോഡിന്റെ ഡാറ്റാഭാഗവും ലിങ്ക് ഭാഗവും സംഭരിക്കുക.

ഘട്ടം 3: യാത്ര ചെയ്യൽ പ്രവർത്തനത്തിന്റെ സഹായത്തോടെ (POS-1), (POS+1) എന്നീ സ്ഥാനങ്ങളിലുള്ള നോഡുകളുടെ അഡ്രസ് യഥാക്രമം PreNode, PostNode എന്നീ പോയിന്ററുകളിലേക്കു സ്വീകരിക്കുക.

ഘട്ടം 4: Temp (POS-1) സ്ഥാനത്തുള്ള നോഡ്) ന്റെ link ഭാഗത്തിലേക്കു PreNode (പുതിയ നോഡ്) ന്റെ അഡ്രസ് സംഭരിക്കുക.

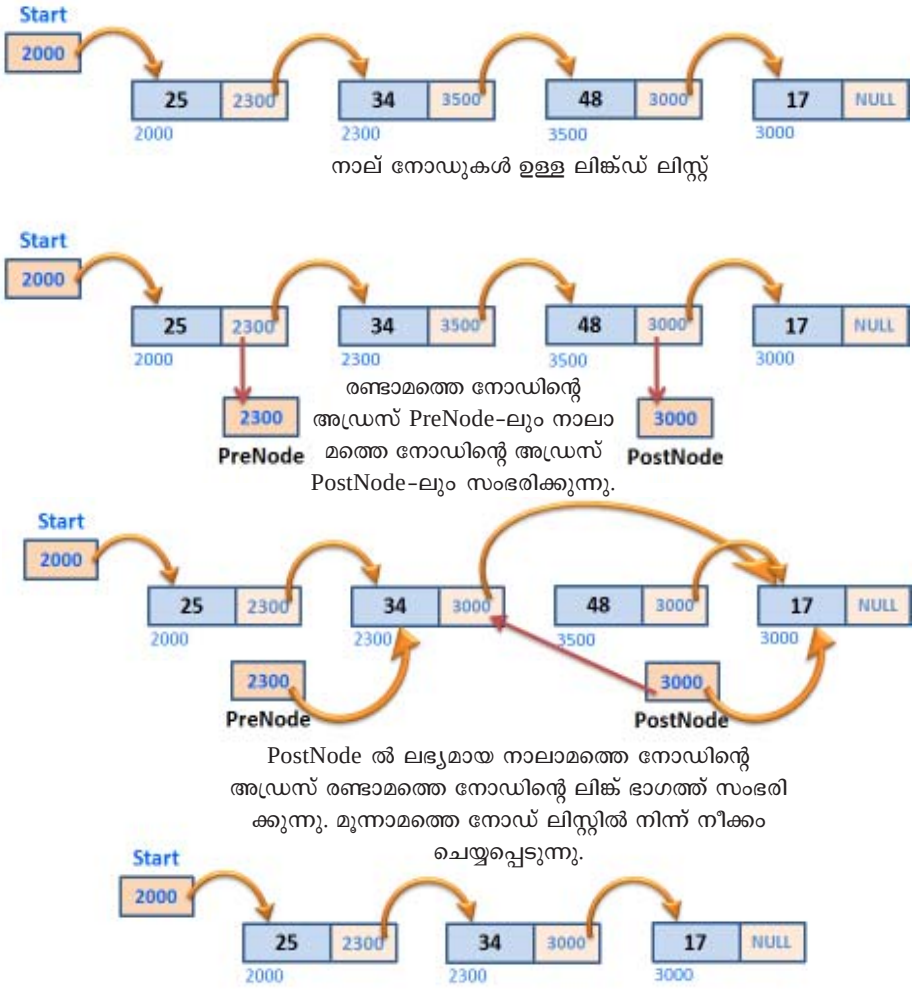
ഘട്ടം 5: Temp (പുതിയ നോഡ്) ന്റെ link ഭാഗത്തിലേക്കു PostNode (POS+1) സ്ഥാനത്തുള്ള നോഡ്) ന്റെ അഡ്രസ് സംഭരിക്കുക.

d. ലിങ്ക്ഡ് ലിസ്റ്റിൽ നിന്നും നീക്കം ചെയ്യൽ (Deletion from a linked list)

ലിങ്ക്ഡ് ലിസ്റ്റിലുള്ള ഒരു അംഗത്തെ നീക്കം ചെയ്യൽ എന്നാൽ ലിസ്റ്റിൽ നിന്നും ഒരു നോഡിനെ ഒഴിവാക്കുന്ന പ്രക്രിയയാണ്. നീക്കം ചെയ്യാനുള്ള നോഡിന്റെ സ്ഥാനം നൽകിയിട്ടുണ്ടായിരിക്കും. അതിനു പകരം ഡാറ്റയാണ് നൽകിയിരിക്കുന്നതെങ്കിൽ, ആ ഡാറ്റ അടങ്ങിയ നോഡ് തിരഞ്ഞു കണ്ടുപിടിക്കുകയും അതിന്റെ സ്ഥാനം കണ്ടെത്തേണ്ടതുമാണ്. അതിനു ശേഷം നീക്കം ചെയ്യൽ പ്രവർത്തനത്തിനുള്ള നടപടിക്രമങ്ങൾ നടപ്പിലാക്കേണ്ടതാണ്. ലിസ്റ്റിലെ ഏതു നോഡിനെയും നീക്കം ചെയ്യുന്നതിന് സാധ്യമാണ്. ഒന്നാമത്തെ നോഡാണ് നീക്കം ചെയ്യേണ്ടതെങ്കിൽ ഒന്നാമത്തെ നോഡിലെ link ലെ അഡ്രസ് START ലേക്ക് സംഭരിക്കണം. അതുപോലെ അവസാനത്തെ നോഡാണ് നീക്കം ചെയ്യേണ്ടതെങ്കിൽ അവസാനത്തേതിന്റെ തൊട്ടു മുമ്പുള്ള നോഡിലെ link ന്റെ വില NULL ആയി സംഭരിക്കണം. ഒരു പ്രത്യേക സ്ഥാനത്തുള്ള നോഡിനെ നീക്കം ചെയ്യുന്നതിന് ആവശ്യമായ നടപടിക്രമങ്ങൾ നമുക്ക് ചർച്ച ചെയ്യാം.

നമുക്ക് ചെയ്യാം

തുടക്കത്തിൽ നാല് നോഡുകളുള്ള ലിങ്ക്ഡ് ലിസ്റ്റിൽ നിന്നും മൂന്നാമത്തെ നോഡിനെ നീക്കം ചെയ്യാനുള്ള നടപടികളാണ് ചിത്രം 3.19 ൽ പ്രതിപാദിച്ചിരിക്കുന്നത്. ചിത്രം നിരീക്ഷിച്ച് ലിങ്ക്ഡ് ലിസ്റ്റിൽ നിന്നും ഒരു നോഡിനെ നീക്കം ചെയ്യാനുള്ള പ്രവർത്തനത്തിന്റെ ഘട്ടങ്ങൾ നമുക്ക് നിർവചിക്കാം. Node തരത്തിൽപ്പെട്ട പോയിന്ററുകളാണ് PreNode, PostNode എന്നിവ എന്നും POS നോഡിനെ നീക്കം ചെയ്യേണ്ട സ്ഥാനം സംഭരിച്ചിരിക്കുന്ന വേരിയബിൾ ആണെന്നും നമുക്ക് അനുമാനിക്കാം.



ചിത്രം 3.19: ലിങ്ക്ഡ് ലിസ്റ്റിലെ നീക്കം ചെയ്യൽ പ്രവർത്തനം

നീക്കം ചെയ്യൽ പ്രവർത്തനത്തെ നിർവചിക്കാൻ താഴെ പറയുന്ന നടപടികൾ നിങ്ങളെ സഹായിക്കുന്നു.

ഘട്ടം 1: കടന്നുപോകൽ പ്രവർത്തനത്തിന്റെ സഹായത്തോടെ (POS-1), (POS+1) എന്നീ സ്ഥാനങ്ങളിലുള്ള നോഡുകളുടെ അഡ്രസ് യഥാക്രമം PreNode, PostNode എന്നീ പോയിന്ററുകളിലേക്കു സ്വീകരിക്കുക.

ഘട്ടം 2: PostNode ((POS+1) സ്ഥാനത്തുള്ള നോഡ്) ന്റെ link ഭാഗത്തിലേക്കു PreNode ((POS-1) സ്ഥാനത്തുള്ള നോഡിന്റെ അഡ്രസ് സംഭരിക്കുക.

ഘട്ടം 3: POS സ്ഥാനത്തുള്ള നോഡിനെ മെമ്മറിയിൽ നിന്നും ഒഴിവാക്കുക.

ചിത്രം 3.19 ൽ സൂചിപ്പിച്ചിരിക്കുന്ന ആദ്യത്തെ രണ്ടു ഘട്ടങ്ങൾ പാലിച്ച് മൂന്നാമത്തെ നോഡിനെ രണ്ടാമത്തേതിൽ നിന്നും വേർപ്പെടുത്തിയാലും, മൂന്നാമത്തെ നോഡ്

നാലാമത്തേതിലേക്കു ചൂണ്ടിക്കൊണ്ട് മെമ്മറിയിൽത്തന്നെ അവശേഷിക്കുന്നു. അതിനാൽ പ്രോഗ്രാമിങ് ഭാഷകളിലെ മെമ്മറി സ്വാതന്ത്ര്യമാക്കൽ സംവിധാനങ്ങൾ ഉപയോഗപ്പെടുത്തി ആ നോഡിനെ നീക്കം ചെയ്യേണ്ടതാണ്. അതുപോലെ ലിങ്ക്ഡ് ലിസ്റ്റുകളിൽ പ്രവർത്തനങ്ങൾ നടത്തുമ്പോൾ പ്രവർത്തനത്തിന് ശേഷം TEMP, PreNode, PostNode മുതലായ താൽക്കാലിക പോയിന്ററുകളും സ്വതന്ത്രമാക്കപ്പെടേണ്ടവയാണ്.

പുരോഗതി ഇതുവരെ

1 .ഡൈനാമിക് ഡാറ്റാസ്ട്രക്ചറിന് ഒരു ഉദാഹരണം പറയുക.

2. ലിങ്ക്ഡ് ലിസ്റ്റ് എന്നാലെന്ത്?

3. ഒരു ലിങ്ക്ഡ് ലിസ്റ്റിന്റെ നോഡിൽ _____ ഉം _____ ഉം അടങ്ങിയിരിക്കും.

4. ഒരു ലിങ്ക്ഡ് ലിസ്റ്റിലെ നോഡിനെ നിർവചിക്കാൻ ഉപയോഗിക്കുന്ന പ്രോഗ്രാമിങ് ഭാഷയിലെ സംവിധാനം ഏതാണ്?

5. ഒരു ലിങ്ക്ഡ് ലിസ്റ്റിൽ Start അല്ലെങ്കിൽ Header ന്റെ ഉള്ളടക്കം എന്തായിരിക്കും?

സ്റ്റാക്കും ക്യൂ വും ലിങ്ക്ഡ് ലിസ്റ്റുകൾ ഉപയോഗിച്ചും നിർമിക്കാവുന്നതാണ്. അവയെ ഡൈനാമിക് സ്റ്റാക് ഡൈനാമിക് ക്യൂ എന്ന് വിളിക്കുന്നു. അടുത്ത നോഡിലേക്കു മാത്രം ചൂണ്ടുന്നതാകയാൽ നമ്മൾ ചർച്ച ചെയ്ത ലിങ്ക്ഡ് ലിസ്റ്റുകളെ സിംഗിളിലിങ്ക്ഡ് ലിസ്റ്റ് എന്ന് വിളിക്കുന്നു. എന്നാൽ അടുത്ത നോഡിലേക്കും തൊട്ടു മുമ്പുള്ള നോഡിലേക്കും ചൂണ്ടുന്ന ഡബിളി ലിങ്ക്ഡ് ലിസ്റ്റും ഉണ്ട്. സ്വയം സൂചിതസ്ട്രക്ചറിൽ രണ്ടു പോയിന്ററുകൾ ഉപയോഗിച്ചാണ് ഇത് സാധ്യമാക്കുന്നത്. ട്രീ (TREE) പോലുള്ള സങ്കീർണമായ ഡാറ്റ സ്ട്രക്ചറുകൾ ഡബിളി ലിങ്ക്ഡ് ലിസ്റ്റുകൾ ഉപയോഗിച്ചാണ് നിർമ്മിക്കുന്നത്.

നമുക്ക് സംഗ്രഹിക്കാം

ഡാറ്റാസ്ട്രക്ചറുകൾ എന്ന ആശയത്തെയും അവയുടെ മേൽ നടപ്പിലാക്കാവുന്ന പ്രവർത്തനങ്ങളെയും നമ്മൾ പരിചയപ്പെട്ടു. ഏതു തരത്തിലുള്ള ഡാറ്റയെയും പ്രതിനിധീകരിക്കാൻ അനുയോജ്യമായ വ്യത്യസ്തങ്ങളായ ഡാറ്റാസ്ട്രക്ചറുകളുണ്ട്. ഇവയിൽ ഉപയോഗിക്കുന്ന പ്രവർത്തനങ്ങളും അംഗങ്ങളെ ഗണങ്ങളാക്കാൻ ഉപയോഗിക്കുന്ന തത്ത്വത്തിനനുസരിച്ചു വ്യത്യാസപ്പെട്ടിരിക്കും. ചില ഡാറ്റാ സ്ട്രക്ചറുകൾ യുക്തിപരമായ ആശയങ്ങൾ മാത്രമാണെങ്കിലും അവയുടെ നടപ്പിലാക്കലിനെക്കുറിച്ചും നമ്മൾ ചർച്ച ചെയ്തു. അറെയും ലിങ്ക്ഡ് ലിസ്റ്റും തമ്മിലുള്ള വ്യത്യാസങ്ങളെപ്പറ്റിയും ചർച്ചയിൽ പരാമർശിച്ചിരുന്നു. കമ്പ്യൂട്ടർ സയൻസിന്റെ ഉപരിപഠനത്തിനു ഈ അധ്യായത്തിൽ പറഞ്ഞ ആശയങ്ങൾ വളരെ വേണ്ടപ്പെട്ടവയാണ്.

നമുക്ക് പരിശോധിക്കാം

1 . താഴെ പറയുന്ന പ്രസ്താവനകൾ വായിക്കുക:

(i) ഒറ്റ ഘടകമായി പ്രോസസ് ചെയ്യപ്പെടുന്ന ഡാറ്റയുടെ ശേഖരം.

(i i) അറെ ഉപയോഗിച്ചാണ് എല്ലാ ഡാറ്റാസ്ട്രക്ചറുകളും പ്രാവർത്തികമാക്കിയിരിക്കുന്നത്.

(i i i) സ്റ്റാക്ക്, ക്യൂ എന്നിവ യുക്തിസഹമായ ആശയങ്ങളും അറെ, ലിങ്ക്ഡ് ലിസ്റ്റ് എന്നിവ ഉപയോഗിച്ച് പ്രാവർത്തികമാക്കിയവയുമാണ്.

(i v) സ്റ്റാറ്റിക് ഡാറ്റാസ്ട്രക്ചറുകളുടെ കാര്യത്തിൽ ഓവർഫ്ളോ സംഭവിക്കുന്നു.

മുകളിൽ കൊടുത്തിരിക്കുന്ന പ്രസ്താവനകളിൽ ഏതാണ് ശരിയായിട്ടുള്ളത്? ഏറ്റവും ഉചിതമായത് തിരഞ്ഞെടുക്കുക.

a. പ്രസ്താവനകൾ (i), (iii) മാത്രം b. പ്രസ്താവനകൾ (i), (ii), (iii) മാത്രം

c. പ്രസ്താവനകൾ (i), (iii), (iv) മാത്രം d. പ്രസ്താവനകൾ (i), (ii), (iv) മാത്രം

2 . ഡാറ്റാസ്ട്രക്ചറുകൾ സ്റ്റാറ്റിക്കോ ഡൈനാമിക്കോ ആവാം.

a. സ്റ്റാറ്റിക് ഡാറ്റാസ്ട്രക്ചറുകൾക്കു രണ്ടു ഉദാഹരണം നൽകുക.

b. സ്റ്റാറ്റിക് ഡാറ്റാസ്ട്രക്ചറുകൾ ഓവർ ഫ്ളോ സാഹചര്യം നേരിടേണ്ടി വരാം. എന്തുകൊണ്ട്?

c. ലിങ്ക്ഡ് ലിസ്റ്റ് ഒരു ഡൈനാമിക് ഡാറ്റാസ്ട്രക്ചർ ആണ്. ഈ പ്രസ്താവനയെ ന്യായീകരിക്കുക.

3. സ്റ്റാക്കിലേക്കു ഒരു അംഗത്തെ ഉൾപ്പെടുത്താനുള്ള അൽഗോരിതം എഴുതുക.

4. പുഷ്, പോപ്പ് പ്രവർത്തനങ്ങൾ എന്നാലെന്ത്?

5. സ്റ്റാക്കിൽ നിന്ന് ഒരു അംഗത്തെ നീക്കം ചെയ്യാനുള്ള അൽഗോരിതം എഴുതുക.

6 . രേഖീയക്യൂവിൽ ഉൾപ്പെടുത്തൽ, നീക്കംചെയ്യൽ പ്രവർത്തനങ്ങൾ പ്രാവർത്തികമാക്കാനുള്ള അൽഗോരിതം എഴുതുക.

7 . രേഖീയക്യൂവിന്റെ പരിമിതികൾ വൃത്താകൃതി ക്യൂ എങ്ങനെ മറികടക്കുന്നു.

8 . ഡാറ്റാസ്ട്രക്ചറുകളിൽ നടപ്പാക്കാവുന്ന ചില പ്രവർത്തനങ്ങൾ താഴെ പറയുന്നു

i. ഉൾപ്പെടുത്തൽii. നീക്കംചെയ്യൽ iii.തിരയൽ iv.ക്രമപ്പെടുത്തൽ

a .ഇവയിൽ ഏതു പ്രവർത്തനത്തിനാണ് അണ്ടർ ഫ്ളോ സാഹചര്യം നേരിടേണ്ടി വരിക.

b. ഈ സാഹചര്യം ഉചിതമായ ഡാറ്റാസ്ട്രക്ചറിന്റെ പശ്ചാത്തലത്തിൽ വിവരിക്കുക.

9 . ചേരുംപടി ചേർക്കുക

A	B	C
a. അറെ	i. സ്റ്റാർട്ട്	1 . വ്യത്യസ്ത ഭാഗങ്ങളിൽ ഉൾപ്പെടുത്തലും നീക്കം ചെയ്യലും സംഭവിക്കുന്നു.
b. സ്റ്റാക്ക്	ii. സൂചിക	2 . ഒരേ ഭാഗത്തു തന്നെ ഉൾപ്പെടുത്തലും നീക്കം ചെയ്യലും സംഭവിക്കുന്നു
c. ക്യൂ	iii. റിയർ	3 . സ്വയം സൂചിത സ്ട്രക്ചറുകൾ ഉപയോഗപ്പെടുത്തുന്നു
d. ലിങ്ക്ഡ് ലിസ്റ്റ്	iv. ടോപ്പ്	4 . അംഗത്തിന്റെ സ്ഥാനം ഉപയോഗിച്ചിട്ടാണ് അതിനെ ആക്സസ് ചെയ്യുന്നത്

10 .അറെ കേന്ദ്രീകൃത ഡാറ്റാസ്ട്രക്ചറുകളുടേതു പോലെ ലിങ്ക്ഡ് ലിസ്റ്റുകൾക്ക് ഓവർഫ്ളോ സാഹചര്യം നേരിടേണ്ടി വരാത്തത് എന്തുകൊണ്ടെന്നു വിശദീകരിക്കുക.