2 ഒബ്ജക്റ്റ് ഓറിയന്റഡ് പ്രോഗ്രാമിങ്ങിന്റെ തത്ത്വങ്ങൾ

പ്രധാന പഠനനേട്ടങ്ങൾ

ഈ അധ്യായത്തിന്റെ പൂർത്തികരണത്തോടെ പഠിതാവിന്

- വിവിധ പ്രോഗ്രാമിങ്ങ് മാതൃകകളെ താരതമ്യം ചെയ്യാൻ സാധിക്കുന്നു.
- പ്രൊസിജർ ഓറിയന്റഡ് പ്രോഗ്രാമിങ്ങ് മാത്രയുടെ വിവിധ സവിശേഷതകൾ സൂചിപ്പിക്കാൻ സാധിക്കുന്നു.
- ഒബ്ജക്ട് ഓറിയന്റഡ് പ്രോഗ്രാമിങ്ങ് മാതൃകയുടെ ഗുണങ്ങൾ സൂചിപ്പിക്കാൻ കഴിയുന്നു.
- ഡാറ്റ അബ്സ്ട്രാക്ഷൻ, ഡാറ്റ എൻക്യാപ്സുലേഷൻ എന്നീ ആശയങ്ങൾ ഉദാഹരണസഹിതം വിശദീകരിക്കാൻ സാധിക്കുന്നു.
- ഇൻഹെറിറ്റൻസ്, പോളിമോർഫിസം എന്നിവ യഥാർഥ ജീവിത ഉദാഹരണങ്ങൾ ഉപയോഗിച്ച് വിശദീകരിക്കാൻ സാധിക്കുന്നു.

കമ്പ്യൂട്ടറുകളെ കൂടുതൽ ഉപയോഗ പ്രദമാക്കുന്നതിനായി നമ്മൾ പ്രോഗ്രാമുകൾ (Software) വികസിപ്പിക്കുന്നു.അതിനായി വിവിധ പ്രോഗ്രാമിങ് ഭാഷകൾ നമ്മൾ ഉപയോഗിക്കുന്നു. പ്രോഗ്രാമിന്റെ വലിപ്പം കൂടുന്തോറും അത് നിർമിക്കുവാനുള്ള ബുദ്ധിമുട്ടും കൂടുന്നു. ഈ ബുദ്ധിമുട്ട് ഒഴിവാക്കുന്നതിനായി IDE, ഡീബഗ്ഗർ, കംപൈലർ എന്നിങ്ങനെ വിവിധ ഉപാധികൾ നമുക്ക് ഉപയോഗിക്കാം.കൂടാതെ വിവിധ നിർമ്മാണ സമീപന രീതികളായ മോഡുലാർ, സ്ട്രക്ച്ചർഡ്, പ്രൊസീജറൽ, ഒബ്ജക്റ്റ് ഓറിയന്റഡ് എന്നിവ സോഫ്റ്റ്വെയർ നിർമ്മാണത്തിന് നമുക്ക് പിന്തുടരാവുന്നതാണ്.ഈ ഉപാധികളും സമീപനങ്ങളും സോഫ്റ്റ്വെയർ നിർമ്മാണ സമയത്തെ സോഫ്റ്റ്വെയർ ഉൽപ്പന്നങ്ങളുടെ പരിപാലനം (Maintainabiltiy), പുനരുപയോഗം (Reusabiltiy), വഹനീയത, (Portabiltiy), സുരക്ഷിതത്ത്വം, സമഗ്രത (Integrtiy), ഉപയോക്തസൗഹൃദം മുതലായ പ്രശ്നങ്ങളെ അഭിമുഖീകരിക്കാൻ സഹായിക്കുന്നു.

മുൻ അധ്യായത്തിൽ വിവിധ പ്രോഗ്രാമിങ് ആശയങ്ങളെ പറ്റിയും C++ ഭാഷ ഉപയോഗിച്ച് പ്രശ്ന പരിഹാരത്തിനായി പ്രോഗ്രാമുകൾ നിർമ്മിക്കുന്നതിനെ പറ്റിയും നമ്മൾ പഠിച്ചതാണ്. ഇൻപുട്ടുകളെ പ്രോസസ്സ് ചെയ്ത് ഔട്ട്പുട്ട് ലഭ്യമാകുക എന്നതായിരുന്നു അവിടെയെല്ലാം നമ്മുടെ ലക്ഷ്യം. തയാറാക്കിയ പ്രോഗ്രാമുകൾ ചെറുതും ലളിതവും ആയിരുന്നതിനാൽ കൈക്കൊണ്ട സമീപനങ്ങളെ പറ്റിയോ ഡാറ്റയുടെ സുരക്ഷിതത്വത്തെ പറ്റിയോ നമ്മൾ ഒരിക്കലും ചിന്തിച്ചിരുന്നില്ല.

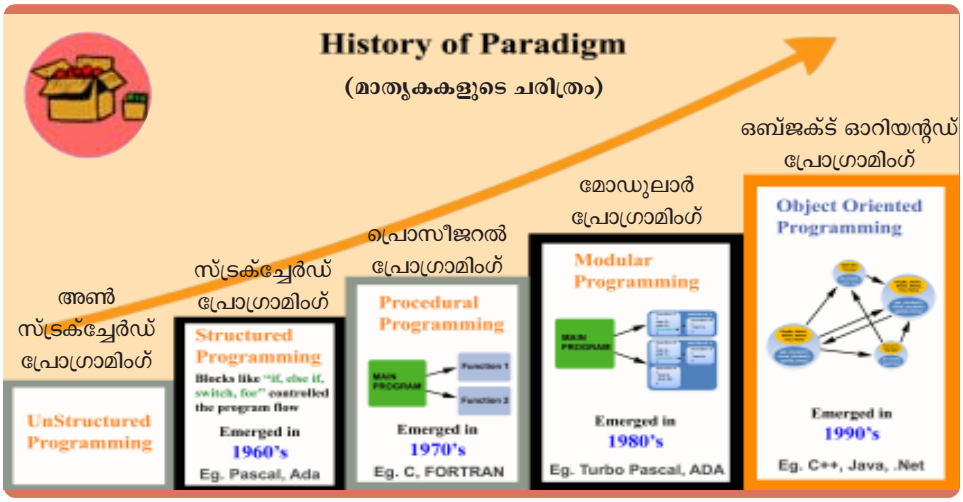
എന്നാലും പ്രശ്ന പരിഹാരത്തിനായി പ്രോഗ്രാമുകൾ തയ്യാറാക്കുമ്പോൾ അറിഞ്ഞോ അറിയാതയോ നാം ഒരു സമീപനം പിന്തുടർന്നു കൊണ്ടിരുന്നു. പ്രോഗ്രാം വികസനത്തിനായി ഏറ്റവും ഉചിതമായ സമീപനം തിരഞ്ഞെടുക്കാൻ നമ്മെ പ്രാപ്തരാക്കുന്നതിനായി ഈ അധ്യായത്തിൽ, നാം ഇതു വരെ ഉപയോഗിച്ച് പോന്ന പ്രോഗ്രാമിങ് സമീപനത്തെപ്പറ്റി ചർച്ച ചെയ്യുകയും പുതിയ സമീപനം അവതരിപ്പിക്കുകയും ചെയ്യുന്നു.

2.1 പ്രോഗ്രാമിങ് മാതൃക (Programming paradigm)

ഒരു പ്രോഗ്രാം എങ്ങനെ ചിട്ടപ്പെടുത്തിയിരിക്കുന്നു എന്നതിനെ സൂചിപ്പിക്കുന്നതാണ് പ്രോഗ്രാമിങ്മാതൃക. പ്രോഗ്രാം വളരെ ചെറുതാണെങ്കിൽ പ്രോഗ്രാം ചിട്ടപ്പെടുത്താൻ പ്രത്യേക തത്ത്വമൊന്നും പാലിക്കേണ്ട കാര്യമില്ല. എന്നാൽ പ്രോഗ്രാം വലുതാകുംതോറും അതിന്റെ സങ്കീർണത കുറയ്ക്കുവാനും അതിനെ പരിപാലിക്കുന്നതിനും ചില മുൻകരുതലുകൾ എടുക്കേണ്ടി വരുന്നു.

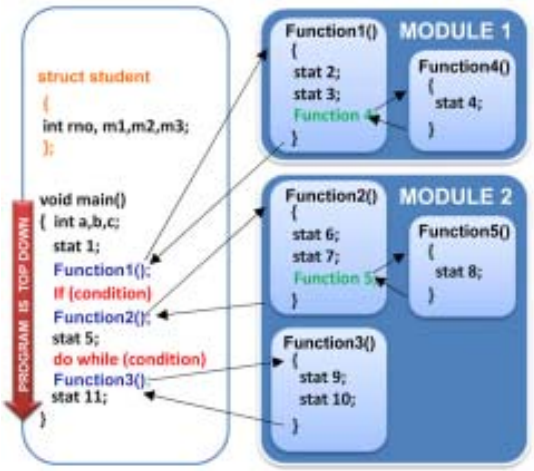
ചില മാതൃകകൾ നടപടിക്രമങ്ങൾക്കു കൂടുതൽ പ്രാധാന്യം കൊടുക്കുമ്പോൾ മറ്റു ചിലത് ഡാറ്റയ്ക്ക് കൂടുതൽ പ്രാധാന്യം കൊടുക്കുന്നു. വിവിധ പ്രോഗ്രാമിങ് ഭാഷകളുടെ കാര്യക്ഷമതയും ശൈലിയുമെല്ലാം അവ അവലംബിക്കുന്ന പ്രോഗ്രാം മാതൃകകളാണ് നിർവചിക്കുന്നത്. മോഡുലാർ പ്രോഗ്രാമിങ്, ടോപ് ഡൗൺപ്രോഗ്രാമിങ്, ബോട്ടം അപ്പ് പ്രോഗ്രാമിങ്, സ്ട്രക്ച്ചർഡ് പ്രോഗ്രാമിങ് മുതലായ രീതികൾ നേരത്തെ പരീക്ഷിക്കപ്പെട്ടവയാണ്. ഇവ ഓരോന്നും പ്രോഗ്രാമിങ്ങിന്റെ സങ്കീർണത കുറയ്ക്കുവാനും വിശ്വസ്തവും പരിപാലന യോഗ്യവുമായ പ്രോഗ്രാമുകൾ നിർമ്മിക്കുവാനും വേണ്ടിയാണ് ഉപയോഗിച്ചത്.

ഒരു പ്രോഗ്രാമിങ് മാതൃക മാത്രം പിന്തുടരുന്ന രീതിയിലാണ് ചില പ്രോഗ്രാമിങ് ഭാഷകൾ രൂപകൽപ്പന ചെയ്തിരിക്കുന്നത്. എന്നാൽ മറ്റു ചിലത് ഒന്നിലധികം മാതൃകകൾ പിന്തുടരുന്നു. C++ ഒന്നിലധികം പ്രോഗ്രാമിങ് മാതൃകകൾ പ്രയോജനപ്പെടുത്തുന്ന ഭാഷയാണ്. C++ ഉപയോഗിച്ച് ഏറ്റവും പ്രധാനപ്പെട്ട രണ്ടു മാതൃകകളായ പ്രോസിജറൽ മാതൃകയും (Procedural paradigm) ഒബ്ജക്റ്റ് ഓറിയന്റഡ് മാതൃകയും (Object Oriented Paradigm) നമുക്ക് പ്രാവർത്തികമാക്കാവുന്നതാണ്. ഇവ ഓരോന്നിനെയും പറ്റി നമുക്ക് വിശദമായി ചർച്ച ചെയ്യാം.



2.1.1 പ്രോസിജർ ഓറിയന്റഡ് പ്രോഗ്രാമിങ് മാതൃക (Procedure-Oriented Programming paradigm)

പ്രോസിജർ ഓറിയന്റഡ് പ്രോഗ്രാമിങ്, ഒരു പ്രോഗ്രാം നിർമ്മിക്കാൻ ആവശ്യമായ നല്ല രീതിയിൽ ക്രമീകരിച്ച ഘട്ടങ്ങളുടെ ശ്രേണിയെയും പ്രവർത്തനങ്ങളെയും സൂചിപ്പിക്കുന്നു. ഒരു പ്രവൃത്തിയെ അല്ലെങ്കിൽ പ്രോഗ്രാമിനെ പൂർത്തീകരിക്കുന്നതിനായുള്ള ചിട്ടയായ ക്രമീകരണങ്ങൾ ഉള്ള പ്രസ്താവനകളും നടപടികളും നിർദേശങ്ങളും ഇവയിൽ അടങ്ങിയിരിക്കും. ഈ നിർദേശങ്ങൾ ഇൻപുട്ട് സ്വീകരിക്കാനുള്ളതോ, ഗണിതപരമോ യുക്തിപരമോ ആയ ക്രിയകൾ നിർവഹിക്കാനുള്ളതോ അതുമല്ലെങ്കിൽ പരിണിതഫലം പ്രദർശിപ്പിക്കാനുള്ളതോ ആവാം. ഈ സമീപനത്തിൽ കാര്യങ്ങൾ ചെയ്യുന്നതിനാണ് ഊന്നൽ നൽകിയിരിക്കുന്നത്. ഈ മാതൃകയിൽ പ്രോഗ്രാം സങ്കീർണവും വലുപ്പം കൂടിയതുമാകുമ്പോൾ നിർദേശങ്ങളുടെ പട്ടികയെ വിഭജിച്ചു നിർദേശങ്ങളെ ഫംഗ്ഷനുകളായി രൂപപ്പെടുന്നു. പ്രോഗ്രാമിലെ മറ്റു ഫംഗ്ഷനുകളുമായുള്ള സമ്പർക്ക മാർഗ്ഗത്തെയും അതിന്റെ ആവശ്യകതയെയും ഫംഗ്ഷൻ വ്യക്തമായി നിർവചിക്കുന്നു. അതു വഴി പ്രോഗ്രാമിന്റെ സങ്കീർണത കുറക്കാൻ ഫംഗ്ഷൻ സഹായിക്കുന്നു. സങ്കീർണത വീണ്ടും കുറയ്ക്കുന്നതിനായി പൊതുവായ ക്രിയകളുമായി ബന്ധപ്പെട്ട ഫംഗ്ഷനുകളെ മോഡ്യൂളുകളായി രൂപപ്പെടുത്തുന്നു. ഒരു വലിയ പ്രോഗ്രാമിനെ അഞ്ചു വ്യത്യസ്ത ഫംഗ്ഷനുകളായി വിഭജിക്കുകയും പൊതുവായ പ്രവർത്തനങ്ങൾ ഉൾപ്പെടുന്ന ഫംഗ്ഷനുകളെ കൂട്ടിച്ചേർത്തു രണ്ട് മൊഡ്യൂളുകളാക്കുകയും ചെയ്യുന്ന പ്രോസിജർ ഓറിയന്റഡ് മാതൃകയാണ് ചിത്രം 2.1 ൽ കാണിച്ചിരിക്കുന്നത്.



ചിത്രം 2.1 പ്രോസിജറൽ മാതൃക

C, Pascal, FORTRAN, BASIC മുതലായവ പ്രോസിജർ ഓറിയന്റഡ് പ്രോഗ്രാമിങ് ഭാഷകളാണ്. C++ പ്രോഗ്രാമിങ് പഠന പ്രക്രിയയിൽ നമ്മൾ ഇത് വരെ പിൻതുടർന്നത് പ്രോസിജർ ഓറിയന്റഡ് മാതൃകയാണ്. പ്രോസിജർ ഓറിയന്റഡ് പ്രോഗ്രാമിങ് ഭാഷകളെ ടോപ് ഡൗൺ പ്രോഗ്രാമിങ് ഭാഷകൾ എന്നും വിളിക്കുന്നു.

പക്ഷെ പ്രോഗ്രാം വലുപ്പം കൂടുകയും സങ്കീർണത വർധിക്കുകയും ചെയ്യുന്നതോടെ അതിന്റെ പരിമിതികൾ വെളിപ്പെട്ടു തുടങ്ങുന്നു. ഈ സമീപനം എത്ര നല്ല രീതിയിൽ പ്രാവർത്തികമാക്കിയാലും വലിയ പ്രോഗ്രാമുകൾ സങ്കീർണമായിത്തന്നെ തുടരുന്നു. പ്രോസിജർ ഓറിയന്റഡ് ഭാഷകളുടെ സങ്കീർണത വർധിക്കുവാനുള്ള പ്രധാന കാരണങ്ങൾ താഴെ പറയുന്നവയാണ്.

a. ഡാറ്റയുടെ പ്രാധാന്യം കുറച്ചു കാണുന്നു.

b. പുതിയ ഡാറ്റ അംഗത്തെ കൂട്ടിച്ചേർക്കുമ്പോൾ എല്ലാ ഫങ്ഷനുകൾക്കോ അല്ലെങ്കിൽ ചില ഫങ്ഷനുകൾക്കോ മാറ്റം വേണ്ടി വരുന്നു.

c. പുതിയ ഡാറ്റ തരങ്ങൾ നിർമ്മിക്കുക പ്രയാസകരമാണ്.

d. യഥാർഥ ജീവിത മാതൃകക്കു യോജിച്ചതല്ല.

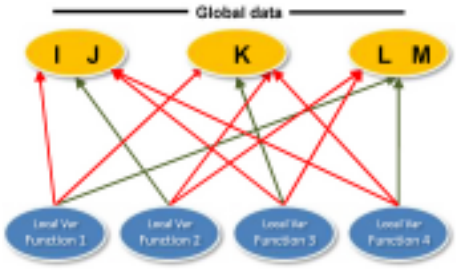
നമുക്ക് ഇത് ഓരോന്നിനെക്കുറിച്ചും ഇവിടെ ചർച്ച ചെയ്യാം.

a. ഡാറ്റയുടെ പ്രാധാന്യം കുറച്ചു കാണുന്നു

പ്രോസിജർ ഓറിയന്റഡ് ഭാഷകൾ, കാര്യങ്ങൾ ചെയ്യുന്നതിനാണ് പ്രാമുഖ്യം നൽകുന്നത്. ഡാറ്റയ്ക്ക് പ്രാധാന്യം കുറവാണ്. ഒരു ഉദാഹരണത്തിന്റെ സഹായത്തോടെ ഇത് വിശദീകരിക്കാം. സ്കൂളിലെ പ്രവർത്തനങ്ങളെ യന്ത്രവൽകരിക്കുന്നതിനായി ഒരു സോഫ്റ്റ്വെയർ വികസിപ്പിക്കുന്നു എന്ന് കരുതുക. ഒരു പുതിയ വിദ്യാർഥിയെ ഉൾപ്പെടുത്തുക, ഒരു വിദ്യാർഥിയെ നീക്കം ചെയ്യുക, ഫീ ശേഖരണ വിവരങ്ങൾ രേഖപ്പെടുത്തുക മുതലായവയാകാം ഇതിൽ ഉൾപ്പെടുന്ന പ്രവർത്തനങ്ങൾ. സോഫ്റ്റ്വെയറിൽ ഈ പ്രവർത്തനങ്ങൾക്കായി പ്രത്യേക ഫംഗ്ഷനുകൾ ഉപയോഗിച്ചിരിക്കുന്നു എന്ന് അനുമാനിക്കുക. വിദ്യാർഥികളുടെ വിവരങ്ങൾ അടങ്ങിയ സ്ട്രക്ച്ചറുകളുടെ (സ്റ്റുഡൻറ്) അറേ ഉപയോഗിച്ച് ഈ ഫംഗ്ഷനുകളുടെ പ്രവർത്തനത്തിനായുള്ള ഡാറ്റ സംഭരിക്കാവുന്നതാണ്. ഈ ഡാറ്റ എല്ലാ ഫംഗ്ഷനുകൾക്കും ലഭ്യമാക്കണമെങ്കിൽ അറേയെ ഗ്ലോബൽ ആയി നിർവചിക്കേണ്ടി വരും (ഒന്നാം വർഷ പാഠപുസ്തകത്തിന്റെ ഭാഗം 10.5 നോക്കുക). ഇപ്പോൾ ഈ ഫംഗ്ഷനുകളെ കൂടാതെ പ്രോഗ്രാമിലുള്ള മറ്റ് ഏതു ഫങ്ഷനുകൾക്കു വേണമെങ്കിലും ഡാറ്റ ഉപയോഗിക്കാവുന്ന അവസ്ഥയായി. തന്മൂലം അറിഞ്ഞോ അറിയാതെയോ ഡാറ്റയ്ക്ക് മാറ്റം സംഭവിക്കാനുള്ള കളമൊരുങ്ങുന്നു.ഇത് തടയുവാൻ നമുക്ക് യാതൊരു തരത്തിലുമുള്ള നിയന്ത്രണവും ഏർപ്പെടുത്താൻ സാധിക്കില്ല.

അടുത്ത ദിവസം സമർപ്പിക്കേണ്ട അസൈൻമെന്റ് തീൻ മേശപ്പുറത്തു വെച്ചിട്ടു പോകുന്നത് പോലെയാണിത്. ഈ അസൈൻമെന്റ് നാശമാകാനുള്ള സാധ്യത ഇവിടെ വളരെ കൂടുതലാണ്. ചെറിയ കുട്ടികളുണ്ടെങ്കിൽ അവർ അത് കീറുകയോ അല്ലെങ്കിൽ അതിൽ ചിത്രങ്ങൾ വരയ്ക്കുകയോ ചെയ്യാം അതുമല്ലെങ്കിൽ മേശപ്പുറത്തു വച്ചിരിക്കുന്ന ചായ അബദ്ധത്തിൽ അതിന്റെ മേൽ മറിഞ്ഞു വീഴാം. ആർക്കു വേണമെങ്കിലും എടുക്കാവുന്ന രീതിയിൽ അസൈൻമെന്റ് വെക്കുന്നത് കൊണ്ടാണ് ഇത് സംഭവിക്കുന്നത്.

ഒരു പ്രോസീജറൽ ഓറിയന്റഡ് പ്രോഗ്രാമിങ് മാതൃകയിലെ ലോക്കൽ വേരിയബിളുകൾ, ഗ്ലോബൽ വേരിയബിളുകൾ, ഫംഗ്ഷനുകൾ മുതലായവയുടെ ക്രമീകരണമാണ് ചിത്രം 2.2 ൽ കാണിച്ചിരിക്കുന്നത്. ഡാറ്റ ആവശ്യമായ ഫങ്ഷനുകളെ പച്ച വരകൾ സൂചിപ്പിക്കുന്നു. ഡാറ്റയുടെ അനധികൃതമായ ഉപയോഗത്തെ ചുവപ്പു വരകൾ സൂചിപ്പിക്കുന്നു.



ചിത്രം 2.2: ഡാറ്റ ആക്സസ് ചെയ്യുന്ന ഫങ്ഷനുകൾ

b. പുതിയ ഡാറ്റ അംഗത്തെ കൂട്ടിച്ചേർക്കുമ്പോൾ എല്ലാ ഫങ്ഷനുകൾക്കോ അല്ലെങ്കിൽ ചില ഫംഗ്ഷനുകൾക്കോ മാറ്റം വേണ്ടി വരുന്നു

പല ഫംഗ്ഷനുകളും ഗ്ലോബൽ ഡാറ്റ ഉപയോഗിക്കുന്നതിനാൽ ഡാറ്റ എങ്ങനെ സംഭരിച്ചിരിക്കുന്നു എന്നത് ഏറെ പ്രാധാന്യമുള്ളതാണ്. ഡാറ്റ ഉപയോഗിക്കുന്ന

ഫങ്ഷനുകളെ പരിഷ്കരിക്കാതെ ഡാറ്റയുടെ ക്രമീകരണത്തിൽ മാറ്റം വരുത്തുക സാധ്യമല്ല. നമ്മൾ പുതിയ ഡാറ്റ അംഗത്തെ കൂട്ടി ചേർക്കുമ്പോൾ ആ ഡാറ്റ ഉപയോഗിക്കുന്ന എല്ലാ ഫങ്ഷനുകളെയും പരിഷ്ക്കരിക്കേണ്ടതുണ്ട്. എന്നാൽ മാത്രമേ ഈ ഫംഗ്ഷനുകൾക്കു പുതിയ ഡാറ്റ അംഗം ലഭ്യമാകൂ. ഇത്തരത്തിൽ ഒരു ഡാറ്റയുമായി ബന്ധപ്പെട്ട എല്ലാ ഫംഗ്ഷനുകളെയും കണ്ടെത്തുക എന്നത് വളരെ ശ്രമകരമായ കാര്യമാണ് എന്ന് മാത്രമല്ല പുതിയ ഡാറ്റ ലഭ്യമാകത്തക്ക രീതിയിൽ അവയെ കൃത്യമായി പരിഷ്ക്കരിക്കുക എന്നത് അതിലും ബുദ്ധിമുട്ടേറിയതാണ്.

നമ്മുടെ സ്കൂൾ സോഫ്റ്റ്വെയറിലെ സ്റ്റുഡൻറ് സ്ട്രക്ച്ചറിലേക്കു 'age' എന്നൊരു പുതിയ ഇനം കൂട്ടിച്ചേർക്കണം എന്നിരിക്കട്ടെ. സോഫ്റ്റ്വെയറിന്റെ സുഗമമായ പ്രവർത്തനത്തിന് സ്റ്റുഡൻറ് ഡാറ്റ ഉപയോഗിക്കുന്ന എല്ലാ ഫംഗ്ഷനുകളെയും കണ്ടെത്തി അവയിൽ ആവശ്യമായ മാറ്റം വരുത്തൽ അനിവാര്യമാണ്.

c. പുതിയ ഡാറ്റ ടൈപ്പുകൾ നിർമ്മിക്കുന്നത് ബുദ്ധിമുട്ടാണ്

ഇൻറിജർ, ഫ്ളോട്ട്, ക്യാരക്ടർ മുതലായ ഡാറ്റ ടൈപ്പുകൾ ആണ് കമ്പ്യൂട്ടർ ഭാഷകളിൽ സാധാരണയായി ഉപയോഗിക്കാറുള്ളത്. ചില പ്രോഗ്രാമിങ് ഭാഷകൾ അടിസ്ഥാന ഡാറ്റാതരങ്ങളെ കൂടാതെ മറ്റു ഡാറ്റാതരങ്ങൾ നിർമ്മിക്കാൻ അനുവാദം നൽകിയിരിക്കുന്നു. അതിനർത്ഥം അവ വിപുലീകരിക്കാവുന്നവയാണ് എന്നാണ്. അടിസ്ഥാന രൂപകൽപ്പനകളിൽ മാറ്റം വരുത്തുകയോ അല്ലെങ്കിൽ പ്രോഗ്രാം കോഡിൽ കാര്യമായ മാറ്റം വരുത്താതെ പ്രോഗ്രാമിന്റെ കഴിവും കാര്യക്ഷമതയും കാര്യമായ വർധന വരുത്തുകയോ ചെയ്യാനുള്ള കഴിവിനെയാണ് വിപുലീകരണ സാധ്യത എന്ന് പറയുന്നത്. ഈ സവിശേഷത പ്രോഗ്രാമുകളുടെ സങ്കീർണത കുറക്കുവാൻ സഹായിക്കുന്നതിനോടൊപ്പം അതിന്റെ കഴിവുകൾ വർധിപ്പിക്കുകയും ചെയ്യുന്നു. പ്രോസിജറൽ പ്രോഗ്രാമുകൾ വിപുലീകരണ സാധ്യതയില്ലാത്തവയാണ്.

d. യഥാർഥ ജീവിത മാതൃകയ്ക്കു യോജിച്ചതല്ല

പ്രോസിജറൽ പ്രോഗ്രാമിങ് മാതൃകയിൽ ഉപയോഗിക്കുന്ന ഫങ്ഷനുകളെയും പ്രോഗ്രാമുകളെയും ഒറ്റ ഘടകമായി പരിഗണിക്കുകയില്ല. മറിച്ചു അവ ഓരോന്നും സ്വതന്ത്രമായി നിലകൊള്ളുന്നു. അതിനാൽ ഈ മാതൃകയിൽ ഉപയോഗിക്കുന്ന ഡാറ്റയും ഫംഗ്ഷനുകളും ഒരു യഥാർത്ഥ ജീവിത മാതൃകയായി പറയാനാവില്ല.

ഉദാഹരണത്തിന് നമ്മുടെ സ്കൂൾ സോഫ്റ്റ്വെയറിൽ നമ്മൾ വിദ്യാർഥികളുടെ ഡാറ്റയും അവയെ കൈകാര്യം ചെയ്യുന്ന ഫംഗ്ഷനുകളും ഉപയോഗിക്കുന്നു. അതുപോലെ അധ്യാപകരുടെ ഡാറ്റയും അവ കൈകാര്യം ചെയ്യുന്ന ഫംഗ്ഷനുകളും ഉണ്ടായിരിക്കും. പ്രോസിജറൽ മാതൃകയാണെങ്കിൽ വിദ്യാർഥികളുടെ ഡാറ്റയും അവയുടെ ഫങ്ഷനുകളും ഒറ്റ ഘടകമാക്കാനും അതുപോലെ അധ്യാപകരുടെ ഡാറ്റയും ഫങ്ഷനുകളും മറ്റൊരു ഘടകമാക്കി മാറ്റാനും സാധിക്കില്ല.

നിത്യജീവിതത്തിലെ ഓരോ വസ്തുവും പരിശോധിച്ചാൽ, അവയെല്ലാം അവയുടെ സവിശേഷതകളും സ്വഭാവ ഗുണങ്ങളും ഒത്തു ചേർന്ന ഒറ്റ ഘടകമാണെന്ന് നമുക്ക് കാണാം. ഉദാഹരണത്തിന് മനുഷ്യനെ പരിഗണിച്ചാൽ, പേര്, പൗരത്വം, ലിംഗം മുതലായവ സവിശേഷതകളും സംസാരം, ചിരി മുതലായവ സ്വഭാവ ഗുണങ്ങളും ആവാം. പ്രോസിജറൽ മാതൃകയിൽ, സ്വഭാവ ഗുണങ്ങളെ ഫംഗ്ഷനുകളായും സവിശേഷതകളെ ഡാറ്റയായും സൂചിപ്പിക്കാമെങ്കിലും അവയെ ഒറ്റ ഘടകമാക്കി മാറാൻ സാധിക്കാത്തതിനാൽ, ഈ മാതൃക യഥാർഥ ജീവിത മാതൃകയുമായി സാമ്യമില്ലാത്തതാണ്.

അടുത്തതായി ഒബ്ജക്റ്റ് ഓറിയന്റഡ് മാതൃകയെപ്പറ്റി ചർച്ച ചെയ്യുകയും അത് പ്രോസിജറൽ മാതൃകയുടെ പരിമിതികൾ എങ്ങനെ മറികടക്കുന്നു എന്ന് കാണുകയും ചെയ്യാം.

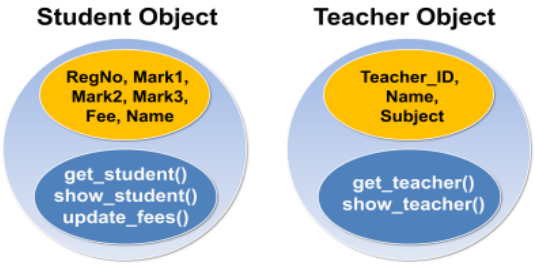
നിങ്ങളുടെ പുരോഗതി അറിയുക

1. താഴെ പറയുന്ന മൂന്നു പ്രസ്താവനകൾ ശരിയോ തെറ്റോ എന്ന് പറയുക:
   a. ഗ്ലോബൽ വേരിയബിളുകൾ ഒന്നിലധികം ഫങ്ക്ഷനുകളിൽ ഉപയോഗിക്കാൻ സാധിക്കില്ല.
   b. യഥാർഥ ലോകവുമായി വളരെ സാമ്യമുള്ളതാണ് പ്രോസിജറൽ മാതൃക.
   c. പ്രോസിജറൽ മാതൃകയിൽ ഡാറ്റയും ഫംഗ്ഷനുകളും ഒറ്റ ഘടകമായി കൂട്ടിച്ചേർക്കാൻ സാധിക്കില്ല.
2. താഴെ പറയുന്നവയിൽ നിന്ന് പ്രോസിജറൽ ഭാഷകൾ തിരഞ്ഞെടുക്കുക.
   C, C++, ഫോർട്രാൻ, ജാവ, പാസ്കൽ

2.1.2 ഒബ്ജക്റ്റ് ഓറിയന്റഡ് പ്രോഗ്രാമിങ് മാതൃക (Object-Oriented Programming (OOP) paradigm)

ഒബ്ജക്റ്റ് ഓറിയന്റഡ് പ്രോഗ്രാമിങ് മാതൃകയിൽ പ്രോസിജറൽ മാതൃകയിലെ പരിമിതികൾ പരിഹരിച്ചു കൊണ്ട് ഡാറ്റയെയും അവയുടെ മേൽ പ്രവർത്തിക്കുന്ന ഫംഗ്ഷനുകളെയും കൂട്ടിച്ചേർത്ത് ഒറ്റ ഘടകമാക്കി മാറ്റുന്നു. ഈ ഘടകത്തെ ഒബ്ജക്റ്റ് (**object**) എന്ന് വിളിക്കുന്നു.

ഉദാഹരണത്തിന് സ്കൂൾ സോഫ്റ്റ്വെയറിൽ OOP നടപ്പിലാക്കിയാൽ വിദ്യാർത്ഥികളുടെ ഡാറ്റയും അവ ഉപയോഗപ്പെടുത്തുന്ന ഫംഗ്ഷനുകളും കൂട്ടിയോജിപ്പിച്ചു സ്റ്റുഡൻറ് (`Student`) എന്ന ഒബ്ജക്റ്റ് നമുക്ക് നിർമിക്കാം. അത് പോലെ അധ്യാപകരുടെ ഡാറ്റയും ഫംഗ്ഷനുകളും കൂട്ടിച്ചേർത്തു ടീച്ചർ (`Teacher`) എന്നൊരു ഒബ്ജക്റ്റും നിർമ്മിക്കാം. ഒരു ഒബ്ജെക്ടിലുള്ള ഫംഗ്ഷനുകൾക്ക് അനുവാദം കൂടാതെ മറ്റൊരു ഒബ്ജെക്ടിലെ ഡാറ്റയെ ഉപയോഗിക്കുവാൻ സാധിക്കുകയില്ല.



ചിത്രം 2.3 ഡാറ്റകളും ഫങ്ഷനുകളും അടങ്ങുന്ന ഒബ്ജക്റ്റുകൾ

OOP ഉപയോഗിക്കുന്നത് കൊണ്ടുള്ള മേന്മകൾ :

a. സ്പഷ്ടമായ ഘടകങ്ങൾ അടങ്ങിയ ഒരു പ്രോഗ്രാമിങ് ഘടന OOP പ്രദാനം ചെയ്യുന്നു.

b. അബ്സ്ട്രാക്ട് ഡാറ്റടൈപ്പുകൾ നിർവചിക്കാൻ ഇത് വളരെ നല്ലതാണ്.

c. എങ്ങനെയാണ് പ്രയോഗത്തിൽ വരുത്തിയിരിക്കുന്നത് എന്നുള്ള വിവരങ്ങൾ മറ്റു ഘടകങ്ങളിൽ നിന്നും മറച്ചു വയ്ക്കാനും വ്യക്തമായ ഒരു സമ്പർക്കമുഖം പ്രദാനം ചെയ്യാനും ഇത് സഹായിക്കുന്നു.

d. നിലവിലുള്ളവയെ ബാധിക്കാത്ത വിധത്തിൽ പുതിയ ഒബ്ജക്റ്റുകൾ നിർമ്മിക്കാവുന്നതിനാൽ നിലവിലുള്ള കോഡിനെ മാറ്റാനും പരിപാലിക്കാനും മറ്റും എളുപ്പമാണ്.

e. യഥാർഥ ജീവിത സാഹചര്യങ്ങളെ പ്രയോഗത്തിൽ വരുത്താൻ സഹായിക്കുന്നു.

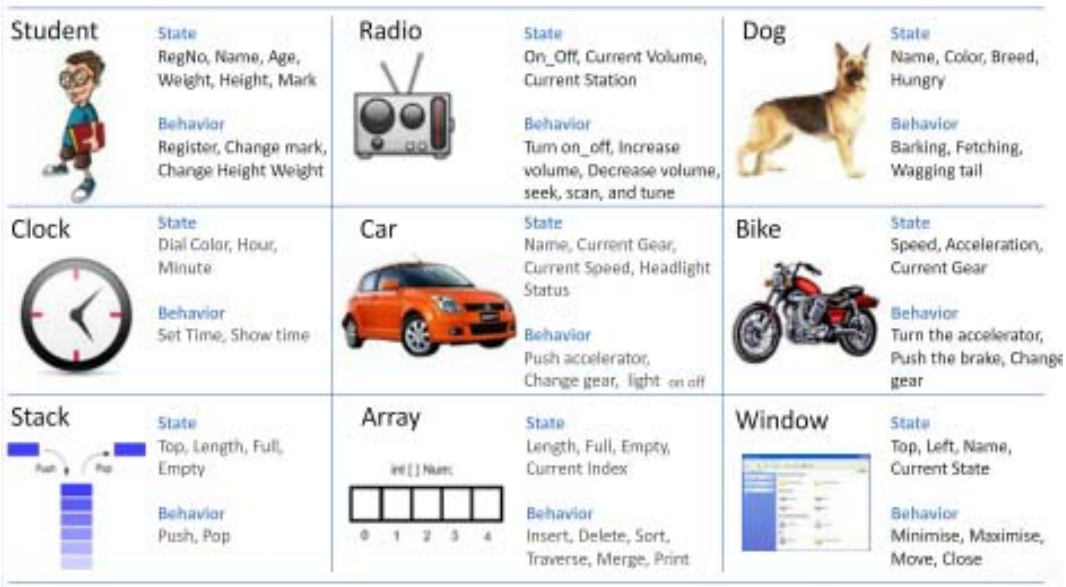
f. ഓപ്പറേറ്ററുകൾക്കു പുതിയ ഡാറ്റാടൈപ്പുകളും പുതിയ പ്രവർത്തനങ്ങളും നിർവചിക്കാൻ സഹായിക്കുന്നു.

2.2 OOP ന്റെ അടിസ്ഥാന തത്വങ്ങൾ (Basic concepts of OOP)

ഒബ്ജക്റ്റുകൾ, ക്ലാസ്സുകൾ, ഡാറ്റ അബ്സ്ട്രാക്ഷൻ (Data Abstraction), ഡാറ്റ എൻക്യാപ്സുലേഷൻ (Data Encapsulation), മോഡുലാരിറ്റി (Modulartiy), ഇൻഹെറിറ്റൻസ് (Inheritance), പോളിമോർഫിസം (Polymorphism) മുതലായ ആശയങ്ങൾ നൽകുന്നതിലൂടെ ഒബ്ജക്റ്റ് ഓറിയന്റഡ് പ്രോഗ്രാമിങ് സോഫ്റ്റ്വെയർ വികസനവും പരിപാലനവും ലളിതമാക്കുന്നു.ഈ ആശയങ്ങൾ നമുക്ക് വിശദമായി ചർച്ച ചെയ്യാം.

2.2.1 ഒബ്ജക്റ്റുകൾ (Objects)

നമുക്ക് ചുറ്റുമുള്ള ഏതൊരു വസ്തുവിനെയും ഒബ്ജക്റ്റായി പരിഗണിക്കാം. എല്ലാ ഒബ്ജക്റ്റുകൾക്കും ഗുണവിശേഷങ്ങളും (ഡാറ്റ/അംഗം/അവസ്ഥ) പ്രവർത്തന രീതികളും (ഫംഗ്ഷനുകൾ/മെത്തേഡുകൾ) ഉണ്ടായിരിക്കും. ചിത്രം 2.4 ൽ ഒബ്ജക്റ്റുകളും അവയുടെ ഗുണവിശേഷങ്ങളും മെത്തേഡുകളും ഉദാഹരണസഹിതം പട്ടികപ്പെടുത്തിരിക്കുന്നു.



ചിത്രം 2.4: നിത്യ ജീവിതത്തിലെ ഒബ്ജക്റ്റുകൾ അവയുടെ സവിശേഷതകളും (അവസ്ഥ) മെത്തേഡുകളോടും (സ്വഭാവ ഗുണങ്ങൾ) കൂടി

നമുക്ക് ചെയ്യാം

നിങ്ങൾക്ക് ചുറ്റുമുള്ള വസ്തുക്കൾ നിരീക്ഷിക്കുകയും അവ ഓരോന്നിന്റെയും സവിശേഷതകളും സ്വഭാവഗുണങ്ങളും തിരിച്ചറിയുക. നിങ്ങളുടെ കണ്ടെത്തലുകൾ താഴെ പറയുന്ന പട്ടികയിൽ എഴുതുക:

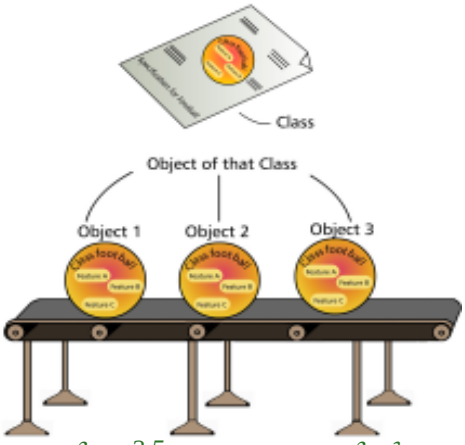
വസ്തുവിന്റെ പേര്	സവിശേഷതകൾ	സ്വഭാവ ഗുണങ്ങൾ

പ്രോഗ്രാമിങ് ഉപയോഗിച്ച് പ്രശ്ന പരിഹാരത്തിനായി OOP പ്രയോജനപ്പെടുത്തുകയാണെങ്കിൽ, പ്രശ്നത്തിനെ ഫംഗ്ഷനുകളായി വിഭജിക്കുന്നതിനു പകരം ഒബ്ജക്റ്റുകളായി വിഭജിക്കുന്നതായി പരിഗണിക്കുക. ഫങ്ഷനുകളായി പരിഗണിക്കാതെ ഒബ്ജക്റ്റുകളായി പരിഗണിക്കുമ്പോൾ പ്രോഗ്രാമിന്റെ രൂപകൽപ്പന ലളിതമായി മാറുന്നു. എന്ത് കൊണ്ടെന്നാൽ പ്രോഗ്രാമുകളിലെ ഒബ്ജക്റ്റുകളും യഥാർഥ ജീവിതത്തിലെ ഒബ്ജക്റ്റുകളും തമ്മിൽ വളരെ സാമ്യമുണ്ട്.

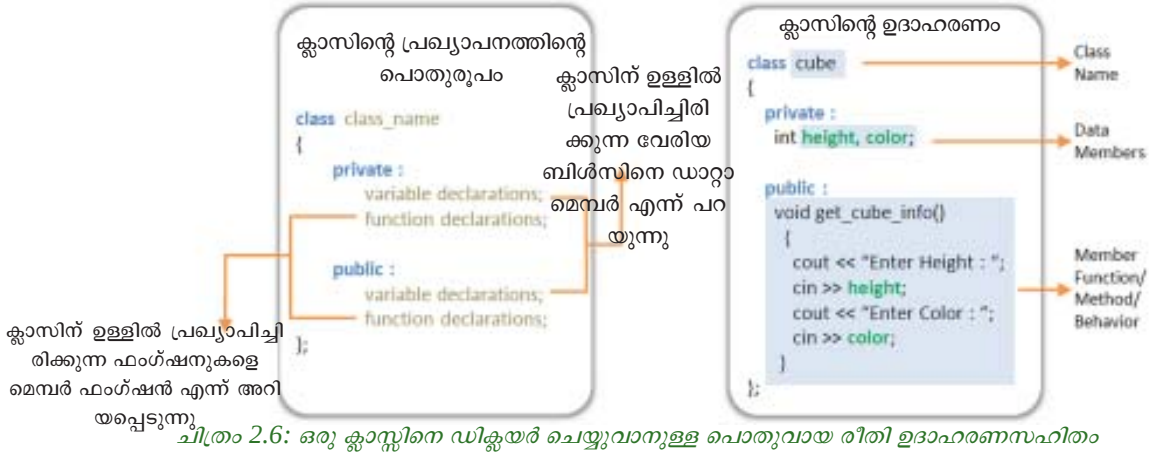
ഡാറ്റയും അവയുടെ മേൽ പ്രവർത്തിക്കുന്ന ഫങ്ഷനുകളും സംയോജിപ്പിച്ചു ഒറ്റ ഘടകമാക്കിയാണ് OOP ൽ ഒബ്ജക്റ്റ് ലഭ്യമാക്കുന്നത്. സംയോജനത്തിനു ശേഷം ഒബ്ജക്റ്റിലുള്ള ഫങ്ഷനുകളെ മെമ്പർ ഫങ്ഷൻ എന്നും ഡാറ്റയെ മെമ്പർ എന്നും വിളിക്കുന്നു.

2.2.2 ക്ലാസുകൾ (Classes)

ഒരു ഒബ്ജെക്റ്റിനെപ്പറ്റിയുള്ള എല്ലാം നിർണയിക്കുന്ന ക്ലാസ് ഉപയോഗിച്ചാണ് ഒരു ഒബ്ജെക്ടിനെ നിർവചിക്കുന്നത്. ഒരു പ്രത്യേക തരത്തിൽപെട്ട എല്ലാ വസ്തുക്കൾക്കും പൊതുവായ ഗുണങ്ങളെ നിർവചിക്കുന്ന മാതൃകയാണ് ക്ലാസ്. ഈ നിർദേശങ്ങളിൽ ഡാറ്റയെ കുറിച്ചും അവയുടെ മേൽ പ്രവർത്തിക്കുന്ന ഫങ്ഷനുകളെ കുറിച്ചും വിവരങ്ങൾ അടങ്ങിയിരിക്കും. ഒരു ക്ലാസ്സിന്റെ ഒബ്ജെക്റ്റിനെ ആ ക്ലാസ്സിന്റെ ഇൻസ്റ്റൻസ് എന്ന് വിളിക്കുന്നു.മാത്രമല്ല ഒരു ക്ലാസ്സിൽ നിന്നും എത്ര ഒബ്ജക്റ്റ് വേണമെങ്കിലും നിർമിക്കാം. (ചിത്രം 2.5 കാണുക).



ചിത്രം 2.5: ക്ലാസുകളും അതിന്റെ ഒബ്ജക്റ്റുകളും

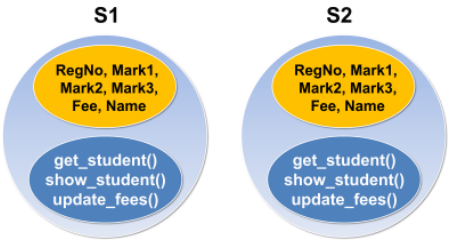


ചിത്രം 2.6: ഒരു ക്ലാസ്സിനെ ഡിക്ലയർ ചെയ്യുവാനുള്ള പൊതുവായ രീതി ഉദാഹരണസഹിതം

ഒരു ക്ലാസ്സിന്റെ നിർവചനവും ഉപയോഗവും സ്ട്രക്ച്ചറിന് സമാനമാണ്. ഒരു സ്ട്രക്ച്ചറിൽ ഡാറ്റയെ കുറിച്ചുള്ള വിവരങ്ങളാണുള്ളതെങ്കിൽ ക്ലാസ്സിൽ ഡാറ്റയെയും അവയുടെ മേൽ പ്രവർത്തിക്കുന്ന ഫങ്ഷനുകളെയും പറ്റിയുള്ള വിവരങ്ങളാണ് അടങ്ങിയിരിക്കുന്നത് (ചിത്രം 2.6 കാണുക). 'struct' എന്ന കീവേർഡ് ഉപയോഗിച്ചാണ് സ്ട്രക്ച്ചർ നിർമിക്കുന്നതെങ്കിൽ 'class' എന്ന കീവേർഡ് ഉപയോഗിച്ചാണ് ക്ലാസ് നിർമിക്കുന്നത്.

Student എന്ന ക്ലാസ് ഉപയോഗിച്ച് 'S1', 'S2' എന്ന് പേരുള്ള രണ്ടു ഒബ്ജക്റ്റുകൾ നിർമ്മിക്കാനുള്ള പ്രസ്താവന താഴെ കൊടുത്തിരിക്കുന്നു. (ചിത്രം 2.7 ത്തിലുള്ളത് പോലെ).

```
Student S1,S2;
```



ചിത്രം 2.7: സ്റ്റുഡന്റ് ക്ലാസ്സിന്റെ ഒബ്ജക്റ്റുകൾ

താഴെ പറയുന്ന പ്രസ്താവന ഉപയോഗിച്ച് ഒരു ക്ലാസ്സിൽ നിന്നും ഒബ്ജക്റ്റുകളെ നിർമ്മിക്കാം.

Cube S; അല്ലെങ്കിൽ **Cube *C; C = new Cube;**

ഇവിടെ Cube എന്നത് ക്ലാസ്സിന്റെ പേരും C എന്നത് ഒബ്ജക്ടിന്റെ പേരും ആണ്.

രണ്ടു ഒബ്ജക്റ്റുകൾ തമ്മിൽ ആശയവിനിമയം നടത്തുന്നത് സന്ദേശം (മെസ്സേജ്) കൈമാറിക്കൊണ്ടാണ്. ഇത് നിത്യ ജീവിതത്തിൽ ആളുകൾ പരസ്പരം സന്ദേശങ്ങൾ കൈമാറുന്നതിന് സമാനമാണ്. യഥാർത്ഥ ജീവിതത്തെ അനുകരിക്കുന്ന തരത്തിലുള്ള സംവിധാനം നിർമിക്കുവാൻ ഇത് സഹായിക്കുന്നു. ഒരു ഒബ്ജെക്റ്റിന്റെ മെമ്പർ ഫങ്ക്ഷനെ മറ്റൊരു ഒബ്ജക്റ്റ് വിളിക്കുന്നതിനെ മെസ്സേജ് പാസിംഗ് എന്ന് പറയുന്നു. മെസ്സേജ് പാസിംഗ് പ്രക്രിയയിൽ ഒബ്ജെക്ടിന്റെ പേര്, മെമ്പർ ഫങ്ഷന്റെ പേര്, അയയ്ക്കുവാനുള്ള വിവരം എന്നിവ സൂചിപ്പിക്കുന്നു.

ചിത്രം 2.8: ഒബ്ജക്റ്റുകൾ സന്ദേശം കൈമാറുന്നു

ഉദാഹരണത്തിന് നമ്മുടെ സ്കൂൾ സോഫ്റ്റ്വെയറിൽ Teacher ഒബ്ജെക്റ്റിനു ഒരു വിദ്യാർഥിയുടെ ഫീസ് പരിഷ്കരിക്കണമെങ്കിൽ Student ഒബ്ജെക്ടിലേക്കു ഡാറ്റ ആവശ്യപ്പെട്ടു കൊണ്ട് ഒരു സന്ദേശം അയക്കുവാൻ update_fee() എന്ന മെമ്പർ ഫങ്ഷനെ വിളിക്കുന്നു (ചിത്രം 2.8 കാണുക).

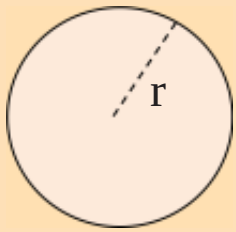
```
S1.update_fees("Rahul",  1000);
```

ഇവിടെ S1 എന്നത് സ്റ്റുഡന്റിന്റെ ഒബ്ജക്റ്റ് ആണ്. സന്ദേശം ലഭിക്കുന്ന മുറയ്ക്ക് Student ഒബ്ജെക്ടിനു Teacher ഒബ്ജക്റ്റ് നൽകുന്ന ഡാറ്റ ഉപയോഗിച്ച് Student ന്റെ ഫീസ് പരിഷ്കരിക്കുന്നു.

നിങ്ങളുടെ പുരോഗതി അറിയുക

1. OOP എന്നാൽ _____.
2. OOP ൽ ഒരു ഒബ്ജെക്റ്റിനുള്ള ബ്ലൂ പ്രിന്റിനെ _____.
3. ക്ലാസ്സുമായി ബന്ധപ്പെട്ട ഫംഗ്ഷനുകളെ _____ എന്ന് വിളിക്കുന്നു.
4. ക്ലാസ്സിന്റെ അകത്തു നിർവചിക്കുന്ന വേരിയബിളുകളെ _____ എന്ന് പറയുന്നു.
5. ക്ലാസ്സും സ്ട്രക്ച്ചറും തമ്മിലുള്ള വ്യത്യാസം എന്ത്?

താഴെ പറയുന്ന പ്രോഗ്രാം വൃത്തത്തെ ഒബ്ജക്റ്റായി ഉപയോഗിക്കുന്നു 'Circle' എന്ന ക്ലാസ്സ് ആവശ്യമായ ഏക ഡാറ്റയായ radius പ്രസ്താവിക്കുകയും radius നെ സ്വീകരിക്കാനും വിസ്തീർണം പ്രദർശിപ്പിക്കാനുമുള്ള മെമ്പർ ഫംഗ്ഷനുകളെ പ്രസ്താവിക്കുകയും ചെയ്യുന്നു.




```
using namespace std;
#include<ciostream>
class Circle
 {
   private:
      float r;
   public:
   void get_radius()
   {
     cout << "Enter Radius :";
     cin >> r;
   }
   void display_area()
   {
     cout<< "Area:"<< 3.14*r*r;
   }
   };
   int main()
   {
   Circle C1;
   C1.get_radius(); //സന്ദേശം അയയ്ക്കുന്നു
   C1.display_area(); //സന്ദേശം അയയ്ക്കുന്നു
   }
```

Member (മെമ്പർ)

Class Declaration (ക്ലാസ് നിർവചനം)

Member functions (മെമ്പർ ഫങ്ഷൻ)

Class (ക്ലാസ്)

Object (ഒബ്ജക്ട്)

മെയിൻ പ്രോഗ്രാം (Main program)

Output :

Enter Radius : 2.0

Area: 12.56

2.2.3 ഡാറ്റ അബ്സ്ട്രാക്ഷൻ (Data Abstraction)

ആവശ്യമായ വിവരങ്ങൾ മാത്രം പുറം ലോകത്തിനു വെളിപ്പെടുത്തിക്കൊണ്ട്, ബാക്കി വിവരങ്ങൾ പിന്നണിയിൽ മറച്ചു വെക്കുകയും ചെയ്യുന്നതിനെയാണ് ഡാറ്റ അബ്സ്ട്രാക്ഷൻ എന്നത് കൊണ്ട് ഉദ്ദേശിക്കുന്നത്.

നിത്യ ജീവിതത്തിലെ ഒരു ഉദാഹരണം ടിവിയുടെ പ്രവർത്തന രീതി പരിഗണിക്കുക. സ്വിച്ച് ഓൺ ചെയ്യുക പവർ ഓഫ് ചെയ്യുക, ചാനൽ മാറ്റുക, ശബ്ദം നിയന്ത്രിക്കുക, VCR പ്ലേയർ, DVD മുതലായ ഉപകരണങ്ങൾ കൂട്ടിച്ചേർക്കുക തുടങ്ങിയ കാര്യങ്ങൾ നമുക്ക് ചെയ്യാവുന്നതാണ്. എന്നിരുന്നാലും നമുക്കതിന്റെ ആന്തരിക പ്രവർത്തനങ്ങളെ കുറിച്ച് അറിവില്ല. ടിവിയിലേക്ക് എങ്ങനെ സിഗ്നലുകൾ ലഭിക്കുന്നു, അതെങ്ങനെ പരിവർത്തനം ചെയ്യപ്പെടുന്നു, എങ്ങനെയത് സ്ക്രീനിൽ പ്രദർശിപ്പിക്കുന്നു എന്നിവയെക്കുറിച്ചു നമുക്കു അറിയില്ല.

അപ്പോൾ ടെലിവിഷൻ അതിന്റെ ആന്തരികമായ പ്രവർത്തനത്തലത്തെയും ബാഹ്യമായ സമ്പർക്കമുഖത്തെയും വേർതിരിക്കുന്നതായി ഇതിൽ നിന്നും നമുക്ക് മനസ്സിലാക്കാം. ആന്തരിക പ്രവർത്തനത്തെക്കുറിച്ചുള്ള പരിജ്ഞാനം കൂടാതെ തന്നെ പവർ ബട്ടൺ, ചാനൽ മാറ്റുവാനുള്ള ബട്ടൺ, ശബ്ദം നിയന്ത്രിക്കാനുള്ള ബട്ടൺ മുതലായ നമുക്ക് ഉപയോഗിക്കാവുന്നതാണ്.

ഇതു പോലെ C++ ക്ലാസുകളും ഡാറ്റ അബ്സ്ട്രാക്ഷൻ പ്രദാനം ചെയ്യുന്നു. ഒരു ഒബ്ജെക്റ്റിന്റെ പ്രവർത്തനത്തെ ഉപയോഗപ്പെടുത്തുവാനും അതിലെ ഡാറ്റ ലഭ്യമാക്കാനും പുറം ലോകത്തിനു ഉപയോഗിക്കാവുന്ന തരത്തിൽ പൊതുവായ മെത്തേഡുകൾ അവ പ്രദാനം ചെയ്യുന്നു. ക്ലാസിന്റെ ആന്തരിക ഘടനയെ കുറിച്ച് അറിയാതെ തന്നെ ഈ മെത്തേഡുകൾ ഉപയോഗപ്പെടുത്തി ഒബ്ജക്റ്റുകളെ കൈകാര്യം ചെയ്യാവുന്നതാണ്.

ഉദാഹരണമായി, വിദ്യാർഥികളുടെ വിശദാംശങ്ങൾ പ്രദർശിപ്പിക്കുന്നതിനായി `Student` ഒബ്ജെക്റ്റിനോടുകൂടി `show_student()` എന്ന മെമ്പർ ഫങ്ക്ഷനെ വിളിക്കാവുന്നതാണ്. ഏതു സമയത്തു വേണമെങ്കിലും നമുക്ക് `show_student()`ന്റെ പ്രവർത്തന രീതി മാറ്റാവുന്നതാണ്. എന്നിരുന്നാലും ഫങ്ക്ഷന്റെ പേരിന് മാറ്റം വരാത്തിടത്തോളം കാലം ഫങ്ക്ഷനെ വിളിക്കുന്നതിന് ഒരു മാറ്റവും വരുന്നില്ല.

ഡാറ്റ അബ്സ്ട്രാക്ഷന്റെ രണ്ടു പ്രധാനപ്പെട്ട പ്രയോജനങ്ങൾ:

- ഉപയോക്ത തലത്തിൽ അപ്രതീക്ഷിതമായി വരാവുന്ന തെറ്റുകളിൽ നിന്നും ക്ലാസ്സിന്റെ ആന്തരിക ഘടനയെ സംരക്ഷിക്കുന്നു. ഈ തെറ്റുകൾ ഒബ്ജെക്ടിന്റെ അവസ്ഥയെ പ്രതികൂലമായി ബാധിക്കുന്നത് ഒഴിവാക്കുവാൻ സാധിക്കുന്നു.
- ആവശ്യകതയനുസരിച്ചു ഉപയോക്ത തലത്തിലുള്ള കോഡിന് മാറ്റം വരുത്താതെ തന്നെ ലാസ്സിന്റെ പ്രവർത്തനത്തിനു മാറ്റം വരുത്താവുന്നതാണ്. കാലാന്തരത്തിലെ ആവശ്യകത അനുസരിച്ച് ക്ലാസിന്റെ നടപ്പിലാക്കലിൽ മാറ്റം ഉണ്ടാക്കുവാൻ, ക്ലാസിനുള്ളിലെ പ്രവർത്തന നിർദേശങ്ങൾ മാറ്റം വരുത്തേണ്ടതായി വരുന്നില്ല.

2.2.4 ഡാറ്റ എൻക്യാപ്സൂലേഷൻ (Data Encapsulation)

ഫംഗ്ഷനുകൾ, ഡാറ്റ എന്നീ രണ്ടു അടിസ്ഥാന ഘടകങ്ങൾ അടങ്ങിയതാണ് C ++ ലെ എല്ലാ പ്രോഗ്രാമുകളും. ഡാറ്റയെയും അവയുടെ മേൽ പ്രവർത്തിക്കുന്ന ഫംഗ്ഷനുകളെയും പരസ്പരം ബന്ധിപ്പിക്കുകയും അവയെ ബാഹ്യമായ ഇടപെടലുകളിൽ നിന്നും ദുരുപയോഗത്തിൽ നിന്നും സംരക്ഷിക്കുകയും ചെയ്യുന്ന OOP തത്വമാണ് ഡാറ്റ എൻക്യാപ്സൂലേഷൻ.

ക്ലാസ് എന്ന് പറയുന്ന ഉപയോക്ത നിർവചിതമായ ഡാറ്റ ടൈപ്പ് ഉപയോഗിച്ചാണ് C++ ഡാറ്റ എൻക്യാപ്സൂലേഷൻ എന്നതു പ്രാവർത്തികമാക്കുന്നത്. `private`, `protected`, `public` എന്നീ മൂന്നുതരം അംഗങ്ങൾ ഒരു ക്ലാസ്സിൽ ഉണ്ടാകാം(ചിത്രം 2.6 കാണുക). ക്ലാസ്സിൽ നിർവചിച്ചിരിക്കുന്ന എല്ലാ അംഗങ്ങളും സ്വാഭാവികമായി `private` ആയിരിക്കും. `private` ആയി നിർവചിക്കപ്പെട്ട അംഗങ്ങൾ ക്ലാസ്സിനു പുറത്തു ദൃശ്യമാകുകയില്ല. `protected` ആയി നിർവചിക്കപ്പെട്ട അംഗങ്ങൾ ബേസ് ക്ലാസ്സിന് പുറമെ ഡിറൈവ്ഡ് ക്ലാസ്സിലും ഉപയോഗിക്കാം (ഭാഗം 2.2.6 ൽ വിശദീകരിച്ചിരിക്കുന്നു). പക്ഷേ അവ ക്ലാസിനു പുറത്തു ദൃശ്യമാകുകയില്ല.

`Student` ക്ലാസിലെ അംഗങ്ങളായ `Regno`, `Name`, `Mark1`, `Mark2`, `Mark3`, `Fee` എന്നിവ `private` ആയിട്ടാണ് നിർവചിച്ചിരിക്കുന്നത്. അതായത് `Student` ക്ലാസ്സിലെ മെമ്പർ ഫങ്ക്ഷനുകൾക്കു മാത്രമേ അവയെ ഉപയോഗിക്കാൻ സാധിക്കൂ, പ്രോഗ്രാമിലെ മറ്റു ഭാഗങ്ങളിൽ ഇവ ലഭ്യമാകുകയില്ല. ഇവിടെ ഡാറ്റ മറയ്ക്കുക വഴി എൻക്യാപ്സൂലേഷൻ സാധ്യമാക്കുകയും ചെയ്യുന്നു.

ക്ലാസിലെ അംഗങ്ങൾ പ്രോഗ്രാമിലെ മറ്റു ഭാഗങ്ങളിലേക്ക് ലഭ്യമാകണമെങ്കിൽ അവയെ `public` എന്ന കീവേർഡ് ഉപയോഗിച്ച് നിർവചിക്കണം. `public` എന്ന വാക്കിനു ശേഷം നിർവചിക്കപ്പെടുന്ന എല്ലാ ഡാറ്റയും ഫംഗ്ഷനുകളും പ്രോഗ്രാമിന്റെ മറ്റു ഭാഗങ്ങൾക്കു ലഭ്യമാകുന്നതാണ്. ഉദാഹരണത്തിന് `Student` എന്ന ക്ലാസ്സിലെ അംഗങ്ങൾ പ്രോഗ്രാമിന്റെ മറ്റു ഭാഗങ്ങൾക്കു ലഭ്യമാകണമെങ്കിൽ `public` ആയിട്ട് അവയെ പ്രസ്താവിക്കണം.

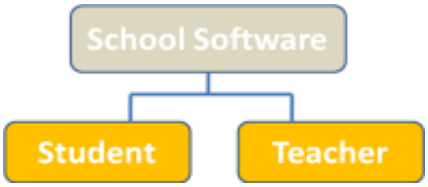
2.2.5 മോഡുലാരിറ്റി (Modularity)

ഒരു പ്രശ്നപരിഹാരണത്തിനായി കമ്പ്യൂട്ടർ പ്രോഗ്രാം എഴുതുമ്പോൾ അതിനെ ഉപ പ്രോഗ്രാമുകളായി വിഭജിച്ച് അവ ഓരോന്നിനും പ്രതിവിധി (ഫലം) കണ്ടെത്തുന്നു. സമ്പർക്കമുഖവും (interface), പ്രവർത്തനതലവും (implementation), വിവരണങ്ങളും (specification) അടങ്ങിയ വ്യത്യസ്ത ഘടകങ്ങളായിരിക്കും ഓരോ ഉപപ്രോഗ്രാമുകളുടെയും പരിഹാരമായി ലഭിക്കുക.

മോഡുലാരിറ്റി എന്ന ആശയം ഉപയോഗിച്ചാണ് തന്നിരിക്കുന്ന പ്രശ്നത്തെ സ്വതന്ത്രമായ ഘടകങ്ങളായി വിഭജിക്കുന്നത്. ഇത്തരത്തിലുള്ള ഓരോ മൊഡ്യൂളും മറ്റു മൊഡ്യൂളുകളിൽ നിന്ന് സ്വതന്ത്രമായി എഴുതപ്പെടുന്നവയായിരിക്കും. പിന്നീട് ഈ മൊഡ്യൂളുകൾ കോർത്തിണക്കി പൂർണമായ ഒരു സോഫ്റ്റ്വെയർ നിർമിക്കുന്നു. ഇത്തരം മൊഡ്യൂളുകൾ ആശയവിനിമയം ചെയ്യുന്നത് പരസ്പരം സന്ദേശങ്ങൾ അയക്കുന്നതിലൂടെ (മെസ്സേജ് പാസിംഗ്) ആണ്.

ഫംഗ്ഷനുകൾ ഉപയോഗിച്ച് മോഡുലാരിറ്റി പ്രാവർത്തികമാക്കുന്നത് എങ്ങനെയാണ് എന്ന് നമ്മൾ മുമ്പ് പഠിച്ചിട്ടുണ്ട് (പതിനൊന്നാം തരത്തിലെ പാഠപുസ്തത്തിൽ അധ്യായം

10 ഫംഗ്ഷനുകൾ കാണുക). മോഡുലാരിറ്റി പ്രവർത്തികമാക്കിയിരിക്കുന്നത് ഒബ്ജക്റ്റ് ഓറിയന്റഡ് പ്രോഗ്രാമിങ്ങിൽ ക്ലാസിന്റെ സഹായത്തോടുകൂടിയാണ്. ഉദാഹരണത്തിന് നമ്മുടെ സ്കൂൾ സോഫ്റ്റ്വെയറിൽ ക്ലാസ് എന്ന ആശയത്തിന്റെ സഹായത്തോടെ വിദ്യാർഥികളുമായി ബന്ധപ്പെട്ടതും അധ്യാപകരുമായി ബന്ധപ്പെട്ടതുമായ കാര്യങ്ങൾ പ്രത്യേകം മൊഡ്യൂളുകളായി തരം തിരിച്ചിരിക്കുന്നു (ചിത്രം 2.9 കാണുക).



ചിത്രം 2.9: മോഡുലാരിറ്റി

2.2.6 ഇൻഹെറിറ്റൻസ് (Inheritance)

ഒരു ക്ലാസ്സിലെ ഒബ്ജക്റ്റുകൾ മറ്റൊരു ക്ലാസ്സിലെ സവിശേഷതകളും പ്രവർത്തനങ്ങളും ആർജിക്കുന്ന പ്രക്രിയയാണ് ഇൻഹെറിറ്റൻസ് എന്ന് പറയുന്നത്. ക്രമപ്രകാരമുള്ള തരം തിരിക്കൽ (Hierarchial Classification), പുനരുപയോഗം (Reusabiltiy) എന്നീ തത്വങ്ങളെ ഇൻഹെറിറ്റൻസ് പിന്തുടരുന്നു.

നിത്യ ജീവിതത്തിലെ ഒരു ഉദാഹരണത്തിലൂടെ നമുക്കീ സാഹചര്യം വിവരിക്കാം. ചിത്രം 2.10 ൽ ഭൂതല വാഹനത്തിനും (land vehicle) ജല വാഹനത്തിനും (water vehicle) വാഹനത്തിന്റെ (Vehicle) സവിശേഷതകൾ (അതായത് ഡാറ്റ അംഗങ്ങളും അംഗങ്ങളായ ഫംഗ്ഷനുകളും) കൈവരുന്നു. കാറിനും ട്രക്കിനും ഭൂതല വാഹനത്തിന്റെ സവിശേഷതകൾ ലഭ്യമാകുന്നു (അതായത് കാർ/ട്രക്ക് = വാഹനം + ഭൂതല വാഹനം). അത് പോലെ ബോട്ടിന് ജല വാഹനത്തിന്റെ സവിഷേതകൾ (അതായത് ബോട്ട്=വാഹനം+ജല വാഹനം) ലഭിക്കുന്നു. ജലത്തിലും ഭൂതലത്തിലും സഞ്ചരിക്കുന്ന ഒരു ഹോവർ ക്രാഫ്റ്റിനു (hovercraft) ഭൂതല വാഹനത്തിന്റെയും ജല വാഹനത്തിന്റെയും സവിശേഷതകൾ (അതായത് ഹോവർ ക്രാഫ്റ്റ്=വാഹനം+ഭൂതല വാഹനം+ജല വാഹനം) ലഭിക്കുന്നു. അതുപോലെ കാറിനെ വീണ്ടും ഹാച്ച്ബാക്ക് (Hatchback), സെഡാൻ (Sedan), എസ് യൂ വി (SUV) മുതലായവയായി വീണ്ടും തരം തിരിക്കാവുന്നതാണ്. അവ ഓരോന്നും വാഹനം, ഭൂതല വാഹനം, കാർ എന്നിവയുടെ സവിശേഷതകൾ ആർജിക്കുന്നു, അതേ സമയം തന്നെ അവയുടേതായിട്ടുള്ള സവിശേഷതകൾ നിലനിർത്തപ്പെടുകയും ചെയ്യുന്നു. ഇത്തരത്തിലുള്ള ക്രമീകരണം ഏതു തലത്തിലേക്കും വ്യാപിപ്പിക്കാവുന്നതാണ്.

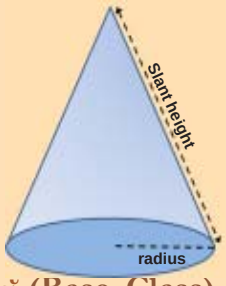
ചിത്രം 2.10: യഥാർത്ഥ ജീവിതത്തിലെ ഇൻഹെറിറ്റൻസ്

ഈ ഉദാഹരണത്തിൽ അഥവാ ഭൂതല വാഹനത്തിന്റെയും ജല വാഹനത്തിന്റെയും പൊതുവായ സവിശേഷതകളും പ്രവർത്തനങ്ങളും വേർപിരിച്ച് വാഹനം എന്ന ക്ലാസ്സിൽ തന്നെ വച്ചിരുന്നില്ലെങ്കിൽ ആ രണ്ടു ക്ലാസ്സുകളിലും അവ ആവർത്തിക്കേണ്ടി വന്നേനെ. ഇത് പ്രോഗ്രാമിങ്ങിന്റെ വലിപ്പവും കോഡ് ചെയ്യാനും തെറ്റ് കണ്ടെത്തി തിരുത്താനുമുള്ള സമയവും വർധിപ്പിക്കുന്നു. തന്നിരിക്കുന്ന ചാർട്ടിന്റെ തുടക്കം മുതൽ അവസാനം വരെ നോക്കിയാൽ പ്രോഗ്രാമിന്റെ സങ്കീർണത എത്രത്തോളം കുറയ്ക്കുവാൻ ഇൻഹെറിറ്റൻസ് കൊണ്ട് സാധിക്കുന്നു എന്ന് നമുക്ക് കാണാം.

ക്ലാസ് നിർമിക്കപ്പെടുകയും തെറ്റ് തിരുത്തപ്പെടുകയും ചെയ്തു കഴിഞ്ഞാൽ ആവശ്യമെങ്കിൽ മറ്റു പ്രോഗ്രാമുകൾക്ക് അതിനെ വിതരണം ചെയ്യാവുന്നതാണ്. ഇതിനെ പുനരുപയോഗം (റീയൂസബിലിറ്റി) എന്ന് വിളിക്കുന്നു. OOP ൽ ഇൻഹെറിറ്റൻസ് തത്വം പുനരുപയോഗം എന്ന ആശയത്തെ ഒന്നുകൂടി വിപുലീകരിക്കുന്നു. പുനരുപയോഗത്തിലൂടെ നിലവിലുള്ള ക്ലാസിൽ മാറ്റം വരുത്താതെ തന്നെ കൂടുതൽ വിശേഷഗുണങ്ങൾ നമുക്ക് കൂട്ടി ചേർക്കാവുന്നതാണ്. നിലവിലുള്ള ക്ലാസ്സിൽ നിന്നും പുതിയ ഒരു ക്ലാസ്സിനെ ഉത്പാദിപ്പിച്ചിട്ടാണ് ഇത് സാധ്യമാകുന്നത്. പുതിയ ക്ലാസിനു അതിന്റേതായ സവിശേഷതകൾ കൈവരുന്നതിനോടൊപ്പം നിലവിലുള്ള ക്ലാസ്സിന്റെ സവിശേഷതകൾ പൈതൃകമായി ആർജിക്കുകയും ചെയ്യുന്നു. നിലവിലുള്ള ക്ലാസ്സിനെ ബേസ് ക്ലാസ് (Base Class) എന്നും പുതിയ ക്ലാസ്സിനെ ഡിറൈവ്ഡ് ക്ലാസ് (Derived Class) എന്നും വിളിക്കുന്നു. ഡിറൈവ്ഡ് ക്ലാസിനു രണ്ടിന്റെയും സംയുക്തമായ സവിശേഷതകൾ ഉണ്ടായിരിക്കും. നിലവിലുള്ള ക്ലാസ്സിൽ നിന്നും എത്ര ഡിറൈവ്ഡ് ക്ലാസ് വേണമെങ്കിലും ഉൽപ്പാദിപ്പിക്കാം. ചിത്രം 2.11 ൽ പുതിയ ക്ലാസ്സിന്റെ ഉൽപ്പത്തി കാണിച്ചിരിക്കുന്നു.

ചിത്രം 2.11: OOP ലെ ഇൻഹെറിറ്റൻസ്

താഴെ പറയുന്ന പ്രോഗ്രാം ഇൻഹെറിറ്റൻസ് വഴി 'Cone' ക്ലാസ്സിലെ ഒബ്ജെക്ടിൽ നിന്നും 'Circle' ക്ലാസിലെ ഒരു ഒബ്ജെക്ടിനെ ഉൽപ്പാദിപ്പിക്കുന്നു.ആരം (r) നേരത്തെ തന്നെ 'Circle', ക്ലാസ്സിൽ ഡിക്ലയർ ചെയ്തിരിക്കുന്നതിനാൽ ഡിറൈവ്ഡ് ക്ലാസ്സായ 'Cone' ൽ സ്ലാൻറ് ഹൈറ്റ് (s). മാത്രം ഡിക്ലയർ ചെയ്താൽ മതിയാകും.ഡാറ്റ സ്വീകരിക്കാനും വിസ്തീർണം പ്രദർശിപ്പിക്കാനും രണ്ടു മെമ്പർ ഫംഗ്ഷനുകൾ ഉപയോഗിച്ചിരിക്കുന്നു. 'Circle' ലെ ഡാറ്റ അംഗങ്ങൾ 'protected' ആയി ഡിക്ലയർ ചെയ്തിരിക്കുന്നതിനാൽ, 'Cone' ക്ലാസ്സിനു അവ ലഭ്യമാകുന്നു.



ബേസ് ക്ലാസ് (Base Class)

```
#include <iostream>
using namespace std;
class Circle
{ protected:
    float r;
  public:
    void get_radius(){
      cout << "Enter Radius :";
      cin >> r;
    }
    void display_area(){
      cout<< "Area:"<< 3.14*r*r;
    }
};
class Cone : public Circle
{ private:
    float s;
  public:
    void get_cone_data() {
      get_radius();
      cout << "Enter slant height:";
      cin >> s;
    }
    void display_cone_area(){
    cout << "Area :" << 3.14*r*(s+r);
    }
};
int main()     {
    Cone C1;
    C1.get_cone_data(); //Sending message
    C1.display_cone_area(); //Sending message
}
```

പുതിയ മെമ്പർ (New member)

പുതിയ മെമ്പർ ഫങ്ഷനുകൾ (New member functions)

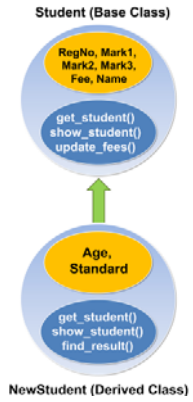
ഡിറൈവ്ഡ് ക്ലാസ് (Derived class)

മെയിൻ ഫങ്ഷൻ (Main function)

Output :
```
Enter Radius : 2.0
Enter slant height: 5.0
Area :43.96
```

സിംഗിൾ ഇൻഹെറിറ്റൻസ്, മൾട്ടിപ്പിൾ ഇൻഹെറിറ്റൻസ്, ഹൈയറാർക്കിയൽ ഇൻഹെറിറ്റൻസ്, മൾട്ടി ലെവൽ ഇൻഹെറിറ്റൻസ്, ഹൈബ്രിഡ് ഇൻഹെറിറ്റൻസ് എന്നിവയാണ് വിവിധ തരത്തിലുള്ള ഇൻഹെറിറ്റൻസുകൾ (ചിത്രം 2.12 കാണുക).

ചിത്രം 2.12: വിവിധ തരത്തിലുള്ള ഇൻഹെറിറ്റൻസ്



ചിത്രം 2.13: ഇൻഹെറിറ്റൻസിന്റെ ഉദാഹരണം

സ്കൂൾ സോഫ്റ്റ്വെയറിൽ ഇൻഹെറിറ്റൻസ് നടപ്പിലാക്കുന്നത് എങ്ങനെയാണെന്ന് നോക്കാം. നിലവിലുള്ള ഡാറ്റയും ഫങ്ഷനും കൂടാതെ Student ക്ലാസിലേക്ക് വയസ്സ്, പഠിക്കുന്ന ക്ലാസ് എന്നീ ഡാറ്റയും പരീക്ഷാഫലം കണ്ടു പിടിക്കാനുള്ള ഫങ്ഷനും കൂട്ടിച്ചേർക്കണം എന്ന് കരുതുക. Student ക്ലാസിനു മാറ്റം വരുത്താതെ NewStudent എന്ന ഒരു പുതിയ ക്ലാസിനെ നമ്മൾ Student ക്ലാസിൽ നിന്നും ഉൽപ്പാദിപ്പിക്കുന്നു. Student ക്ലാസ് അതേപടി തുടരുന്നു. ഇവിടെ Student ക്ലാസ് ബേസ് ക്ലാസും, NewStudent ക്ലാസ് ഡിറൈവ്ഡ് ക്ലാസും ആണ് (ചിത്രം 2.13 കാണുക).

ഡിറൈവ്ഡ് ക്ലാസിനെ പ്രഖ്യാപിക്കാനുള്ള ഘടന താഴെ പറയുന്നത് പോലെയാണ്.

```
class derived_class: AccessSpecifier base_class
{
        //മെമ്പറുകളുടെയും മെമ്പർ ഫങ്ഷനുകളുടെയും പ്രഖ്യാപനം
(declaration of members and memberfunctions)
};
```

ഇവിടെ derived_class എന്നത് ഡിറൈവ്ഡ് ക്ലാസ്സിനെയും base_class എന്നത് ഏതു ക്ലാസ്സിൽ നിന്നാണോ ഡിറൈവ്ഡ് ക്ലാസ്സിനെ ഉൽപാദിപ്പിക്കുന്നത് ആ ക്ലാസ്സിനെയുമാണ് സൂചിപ്പിക്കുന്നത്. AccessSpecifier എന്നത് public, protected, private ഇവയിൽ ഏതെങ്കിലും ആവാം. ഇത് ബേസ് ക്ലാസ്സിൽ നിന്നും പൈതൃകമായി ലഭിച്ച അംഗങ്ങളുടെ ലഭ്യതയെ സൂചിപ്പിക്കുന്നു.

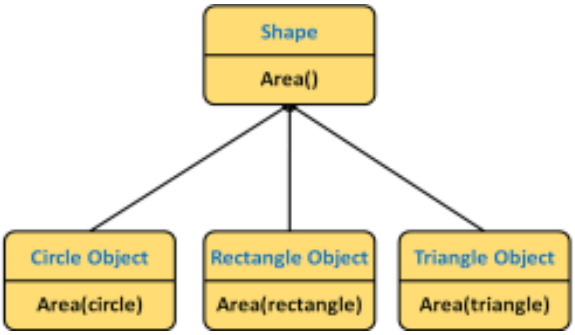
2.2.7 പോളിമോർഫിസം (Polymorphism)

'പോളി' എന്നാൽ ഒന്നിലധികം എന്നും 'മോർഫ്' എന്നാൽ ആകൃതികൾ എന്നുമാണ് അർഥം. വിവിധ രൂപത്തിൽ പ്രകടിപ്പിക്കാനുള്ള കഴിവിനെ പോളിമോർഫിസം എന്ന് നിർവചിക്കാം. ചിത്രം 2.14 ൽ ഇത് വിശദീകരിച്ചിരിക്കുന്നു. ഇവിടെ 'Now Speak' എന്ന ഒരേ നിർദേശമാണ് വിവിധ ജീവികൾക്ക് നൽകുന്നത്, പക്ഷെ അവ ഓരോന്നും വ്യത്യസ്ത രീതികളിലാണ് ഈ നിർദേശത്തോട് പ്രതികരിക്കുന്നത്.

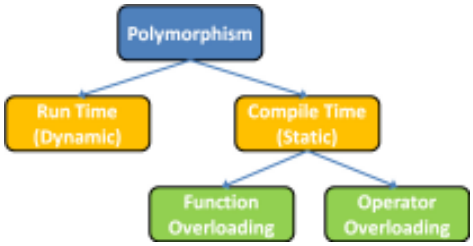
ഒബ്ജക്റ്റുകളുടെ ഡാറ്റ ടൈപ്പ് അല്ലെങ്കിൽ ക്ലാസ് അനുസരിച്ച് അവയെ വ്യത്യസ്തമായി പ്രോസസ്സ് ചെയ്യാനുള്ള പ്രോഗ്രാമിങ് ഭാഷയുടെ കഴിവിനെയാണ് ഒബ്ജക്റ്റ് ഓറിയന്റഡ് പ്രോഗ്രാമിങ്ങിൽ പോളിമോർഫിസം എന്ന് പറയുന്നത്. ഒന്നുകൂടി വ്യക്തമായി പറഞ്ഞാൽ ഡിറൈവ്ഡ് ക്ലാസിന്റെ രീതികളെ പുനർ നിർവചിക്കാനുള്ള കഴിവാണ് പോളിമോർഫിസം.

ചിത്രം 2.14: പോളിമോർഫിസത്തിന്റെ വിശദീകരണം

ഉദാഹരണത്തിന് `Shape` എന്ന ബേസ് ക്ലാസ്സിൽ നിന്നും ഉൽപാദിപ്പിച്ചിരിക്കുന്ന `circle`, `rectangle`, `triangle` മുതലായ ഡിറൈവ്ഡ് ക്ലാസ്സുകൾക്ക് `area` എന്ന പേരിൽ വിവിധ മെത്തേഡുകൾ നിർവചിക്കുവാനുള്ള സൗകര്യം പോളിമോർഫിസം പ്രോഗ്രാമർക്കു നൽകുന്നു (ചിത്രം 2.15 കാണുക). ഒബ്ജക്റ്റ് ഏതായാലും `area` എന്ന മെത്തേഡ് പ്രയോഗിക്കുമ്പോൾ കൃത്യമായ ഫലം ലഭിക്കുന്നു.



ചിത്രം 2.15: പോളിമോർഫിസത്തിന്റെ ഉദാഹരണം



ചിത്രം 2.16: പോളിമോർഫിസത്തിന്റെ വർഗീകരണം

പോളിമോർഫിസം രണ്ടു തരത്തിലുണ്ട് (ചിത്രം 2.16 കാണുക)

a. കംപയിൽ ടൈം പോളിമോർഫിസം (ഏർലി ബൈൻഡിങ്ങ്/സ്റ്റാറ്റിക്) (early binding/static)

b. റൺ ടൈം പോളിമോർഫിസം (ലേറ്റ് ബൈൻഡിങ്ങ്/ഡൈനാമിക്) (late binding/dynamic)

a. കംപയിൽ ടൈം പോളിമോർഫിസം/ഏർലി ബൈൻഡിങ് (Compile time polymorphism)

കംപൈൽ സമയത്തു തന്നെ ഫംഗ്ഷൻ വിളിക്കുന്നതിനെ ഫംഗ്ഷൻ നിർവചനവുമായി ബന്ധിപ്പിക്കാനുള്ള കമ്പലെറിന്റെ കഴിവിനെയാണ് കംപയിൽ ടൈം പോളിമോർഫിസം സൂചിപ്പിക്കുന്നത്. ഇതിന്റെ കീഴിൽ വരുന്നതാണ് ഫങ്ഷൻ ഓവർലോഡിങ്, ഓപ്പറേറ്റർ ഓവർലോഡിങ് എന്നിവ.

ഫങ്ഷൻ ഓവർലോഡിങ് (Function Overloading): ഒരേ പേരും വ്യത്യസ്തങ്ങളായ സിഗ്നേച്ചറുകളും (ഫംഗ്ഷൻ ആർഗ്യുമെന്റുകളുടെ എണ്ണവും തരവും) ഉള്ള ഫങ്ഷനുകൾ വ്യത്യസ്തങ്ങളായി പ്രവർത്തിക്കുന്നു. ഉദാഹരണത്തിന് `area(int)` എന്നത് ഒരു സമചതുരത്തിന്റെ വിസ്തീർണം കണ്ടു പിടിക്കാൻ ഉപയോഗിക്കുന്നു, അതേ സമയം `area(int, int)` എന്നത് ഒരു ചതുരത്തിൻറെ വിസ്തീർണം കണ്ടുപിടിക്കാൻ ഉപയോഗിക്കുന്നു. ഇപ്രകാരം `area()` എന്ന ഒരേ ഫങ്ഷൻ വ്യത്യസ്ത സിഗ്നേച്ചറുകൾക്കനുസരിച്ച് രണ്ടു രീതിയിൽ പ്രവർത്തിക്കുന്നു. ഒന്നിൽ കൂടുതൽ ഫങ്ഷനുകൾ ഒരേ പേരും വ്യത്യസ്ത സിഗ്നേച്ചറുകളോടും കൂടി നിർവചിക്കുന്നതിനെ ഫങ്ഷൻ ഓവർലോഡിങ് എന്ന് പറയുന്നു.

ഓപ്പറേറ്റർ ഓവർലോഡിങ് (Operator overloading): നിലവിലുള്ള C++ ഓപ്പറേറ്ററുകൾക്ക് (+, -, =, * മുതലായവ) പുതിയ നിർവചനം നൽകുന്ന ആശയമാണ് ഓപ്പറേറ്റർ ഓവർലോഡിങ്. ലഭ്യമാകുന്ന ഓപെറന്റുകൾക്ക് അനുസരിച്ച് ഒരു ക്ലാസ്സിലെ വ്യത്യസ്ത ഒബ്ജക്റ്റുകളുടെ മേൽ വ്യത്യസ്തമായി പ്രവർത്തിക്കാൻ സാധാരണ ഓപ്പറേറ്ററുകളെ പ്രാപ്തരാക്കുന്ന പ്രക്രിയയാണിത്. ഒരു ഓപ്പറേറ്ററിനെ ഓവർലോഡ് ചെയ്യണമെങ്കിൽ ഓവർലോഡ് ചെയ്യുന്ന ഓപ്പറേറ്ററിനായി ഒരു മെമ്പർ ഫങ്ഷൻ നിർവചിക്കേണ്ടതാണ്.

ഉദാഹരണത്തിന് C++ ലെ + (പ്ലസ്) എന്ന ഓപ്പറേറ്റർ നിലവിൽ തന്നെ ഓവർലോഡ് ചെയ്യപ്പെട്ടതാണ്. ഇതിന് പൂർണ സംഖ്യകളെയും (`4 + 5`) അസ്ഥിര ദശാംശ സംഖ്യകളെയും കൂട്ടുവാൻ (`3.14 + 2.6`) സാധിക്കുന്നു. ആവശ്യമെങ്കിൽ രണ്ടു ഒബ്ജക്റ്റുകൾ തമ്മിലുള്ള സങ്കലനത്തിനും ഇതേ ഓപ്പറേറ്റർ ഉപയോഗിക്കാം (അതിനുള്ള കോഡ് ക്ലാസ്സിൽ ചേർക്കണം). ഉദാഹരണത്തിന്, `T1 = T2 + T3`. ഇവിടെ `T1`, `T2`, `T3` എന്നത് 'time' എന്ന ക്ലാസ്സിന്റെ ഒബ്ജക്റ്റുകളാണ് . '+' എന്ന ഓപ്പറേറ്റർ ഉപയോഗിച്ച് HH:MM:SS മാതൃകയിലുള്ള രണ്ടു സമയങ്ങളുടെ തുക കണ്ടുപിടിക്കാവുന്നതാണ്.

b. റൺ ടൈം പോളിമോർഫിസം/ലേറ്റ് ബൈൻഡിങ് (Run time polymorphism)

റൺ ടൈമിൽ ഫംഗ്ഷൻ ഉപയോഗിക്കുമ്പോൾ ഫങ്ഷൻ നിർവചനവുമായി ബന്ധിപ്പിക്കുന്നതിനെ റൺ ടൈം പോളിമോർഫിസം എന്ന് പറയുന്നു. ഇൻഹെറിറ്റൻസ്, പോയിന്ററുകൾ എന്നീ ആശയങ്ങളാണ് ഇതിനായി ഉപയോഗിക്കുന്നത്.

താഴെ പറയുന്ന പ്രോഗ്രാം ഫംഗ്ഷൻ ഓവർലോഡിങ് ഉപയോഗിച്ച് സമചതുരത്തിന്റെയും ദീർഘചതുരത്തിന്റെയും വിസ്തീർണം കണ്ടുപിടിക്കുന്നു. സമചതുരത്തിന്റെ വിസ്തീർണം കണ്ടുപിടിക്കാൻ വേണ്ടിയും ദീർഘചതുരത്തിന്റെ വിസ്തീർണം കണ്ടെത്താൻ വേണ്ടിയും രണ്ടു ഫംഗ്ഷനുകൾ ഇത് നിർവചിക്കുന്നു.അവ രണ്ടിനും 'area' എന്ന പിതാവായ പേര് നൽകിയിട്ടുണ്ടെങ്കിലും അവ രണ്ടിലേക്കും നൽകുന്ന വിലകൾ വ്യത്യസ്തമാണ്.

ഫങ്ഷൻ പേരുകൾ ഒരുപോലെയാണ് (Function names are same)

```
#include<ciostream>
using namespace std;
int area(int s){ //സമചതുരത്തിന്റെ വിസ്തീർണം കണ്ടുപിടിക്കാൻ
return s * s;
}
int area(int s1, int s2){ //ചതുരത്തിന്റെ വിസ്തീർണം
return (s1 * s2);          //കണ്ടുപിടിക്കാൻ
}
int main()
{
cout << "Area of Square:" << area(5); << endl;
cout << "Area of Rectangle:"<< area (7,2);
}
```

സിഗ്നേച്ചറുകൾ വ്യത്യസ്തമാണ് (Signatures are different)

Output:

```
Area of Square: 25
Area of Rectangle: 14
```

നിങ്ങളുടെ പുരോഗതി അറിയുക

1. ഡാറ്റയെയും ഫങ്ക്ഷനുകളെയും ഒറ്റ ഘടകമാക്കി മാറ്റുന്നതിനെ _____ എന്ന് വിളിക്കുന്നു.
2. ഡാറ്റയിലേക്കുള്ള അനുമതി നിയന്ത്രിക്കുന്ന സവിശേഷത _____ എന്നറിയപ്പെടുന്നു.
3. ഒബ്ജക്റ്റുകൾ സാധാരണയായി _____ ബൈൻഡിങ് ഉപയോഗിക്കുന്നു.
4. C++ _____, _____ ബൈൻഡിങ് ഉപയോഗിക്കുന്നു
5. ഏർലി ബൈൻഡിങ്ങിനെ _____ വിളിക്കുന്നു.
6. ലേറ്റ് ബൈൻഡിങ്ങിനെ _____ വിളിക്കുന്നു.
7. വിവിധ തരത്തിലുള്ള ഇൻഹെറിറ്റൻസ് ഏതൊക്കെയാണ്?

നമുക്ക് സംഗ്രഹിക്കാം

സോഫ്റ്റ്വെയർ കമ്പ്യൂട്ടറിനെ ഉപയോഗപ്രദമായ ഒരു ഉപകരണമാക്കി മാറ്റുന്നതിനാൽ, അതിന്റെ നിർമ്മാണത്തിനും പരിപാലനത്തിലും പ്രത്യേക പരിഗണന ആവശ്യമാണ്. മറ്റൊരു തരത്തിൽ പറഞ്ഞാൽ സോഫ്റ്റ്വെയർ വികസനം ഫലപ്രദമാക്കാനും പരിപാലനത്തിനുള്ള ചെലവ് കുറക്കാനും പല തരത്തിലുള്ള മാതൃകകൾ പരീക്ഷിക്കപ്പെട്ടിട്ടുണ്ട്. സ്ട്രക്ച്ചർഡ് മാതൃക, പ്രോസിജറൽ ഓറിയന്റഡ് മാതൃക, ഒബ്ജക്റ്റ് ഓറിയന്റെഡ് മാതൃക (OOP) മുതലായവ അവയ്ക്കുദാഹരണമാണ്. നൂതനവും പ്രചാരത്തിലുള്ളതുമായ പ്രോഗ്രാമിങ് ഭാഷകൾ OOP മാതൃക പിന്തുടരുന്നു. പരസ്പരം ആശയവിനിമയം നടത്തുന്ന വസ്തുക്കളുടെ (ഒബ്ജക്റ്റുകളുടെ) സഹായത്തോടെയാണ് OOP പ്രാവർത്തികമാക്കിയിരിക്കുന്നത്. ഇവിടെ ഡാറ്റക്ക് കൂടുതൽ പ്രാധാന്യം നൽകിയിരിക്കുന്നു. ആധികാരികമല്ലാത്ത ഡാറ്റയുടെ ഉപയോഗത്തിൽ നിന്നും സംരക്ഷണം ലഭിക്കാൻ വിവിധ ആക്സസ് സ്പെസിഫയറുകൾ ഫംഗ്ഷനുകളുടെ കൂടെ ഉൾപ്പെടുത്തുകയും ചെയ്യുന്നു. പുനരുപയോഗം മെച്ചപ്പെടുത്തുന്നതിനോടൊപ്പം കോഡ് വിപുലീകരണത്തിനും ഫലപ്രദമായ ഘടനാ ക്രമീകരണത്തിനും സ്റ്റാറ്റിക് ഡൈനാമിക് പോളിമോർഫിസം പ്രാവർത്തികമാക്കാനും OOP സഹായിക്കുന്നു.

നമുക്ക് വിലയിരുത്താം

1. അധികാരികത ഇല്ലാത്ത ഫങ്ക്ഷനുകളിൽ നിന്നും ഡാറ്റയെ സംരക്ഷിക്കുന്നതാണ് _____.
   a. പോളിമോർഫിസം b. എൻക്യാപ്സൂലേഷൻ
   c. ഡാറ്റ അബ്സ്ട്രാക്ഷൻ d. ഇൻഹെറിറ്റൻസ്
2. ബേസ് ക്ലാസിനെ _____ എന്നും വിളിക്കുന്നു.
   a. ചൈൽഡ് ക്ലാസ്സ് b. സബ് ക്ലാസ്സ് c. ഡിറൈവ്ഡ് ക്ലാസ് d.പേരന്റ് ക്ലാസ്
3. താഴെ പറയുന്നവയിൽ ഇൻഹെറിറ്റൻസ് അല്ലാത്തത് ഏത്?
   a. ഹൈബ്രിഡ് b. മൾട്ടിപ്പിൾ c. മൾട്ടിലെവൽ d. മൾട്ടിക്ലാസ്സ്
4. സബ് ക്ലാസിനു തുല്യമാണ്
   a. ഡിറൈവ്ഡ് ക്ലാസ് b. സൂപ്പർ ക്ലാസ്
   c. ബേസ് ക്ലാസ് d. ഇതൊന്നും അല്ല
5. സ്വാഭാവികമായിട്ടുള്ള ആക്സസ് സ്പെസിഫിയർ ആണ്
   a. പബ്ലിക് b. പ്രൈവറ്റ് c. പ്രൊട്ടക്റ്റഡ് d. ഇവ ഒന്നുമല്ല
6. താഴെ പറയുന്നവയിൽ OOP ആശയം അല്ലാത്തതേത്
   a. ഓവർലോഡിങ് b. പ്രോസിജറൽ ഓറിയന്റഡ് പ്രോഗ്രാമിങ്
   c. ഡാറ്റ അബ്സ്ട്രാക്ഷൻ d. ഇൻഹെറിറ്റൻസ്

7. ഒരു ഡാറ്റക്ക് അല്ലെങ്കിൽ മെസ്സേജിന് ഒന്നിലധികം രൂപത്തിൽ പ്രോസസ്സ് ചെയ്യപ്പെടാനുള്ള കഴിവാണ്.
   a. പോളിമോർഫിസം b. എൻക്യാപ്സുലേഷൻ
   c. ഡാറ്റ ഹൈഡിങ് d. ഇൻഹെറിറ്റൻസ്
8. C++ ഒരു _____ ഭാഷയാണ്.
   a. ഒബ്ജക്ട് b. നോൺ പ്രൊസിജറൽ
   c. ഒബ്ജക്റ്റ് ഓറിയന്റെഡ് d. പ്രൊസിജറൽ
9. താഴെ പറയുന്നവയിൽ OOP ന്റെ പ്രത്യേകത അല്ലാത്തതേത്?
   a. ഡാറ്റയെക്കാൾ നടപടികൾക്ക് പ്രാധാന്യം നൽകുന്നു
   b. യഥാർത്ഥ ലോകത്തിന്റെ മാതൃകയിലുള്ളത്
   c. ഡാറ്റയെയും ബന്ധപ്പെട്ട ഫങ്ക്ഷനുകളെയും ഒറ്റ ഘടകമായി കൂട്ടിയിണക്കുന്നു.
   d. ഇവ ഒന്നുമല്ല
10. താഴെ പറയുന്നവയിൽ OOP നെ കുറിച്ച് ശരിയായതേത്?
   a. ഡാറ്റ അബ്സ്ട്രാക്ഷനെ പിന്തുണക്കുന്നു
   b. പോളിമോർഫിസത്തെ പിന്തുണക്കുന്നു
   c. ഘടനാപരമായ പ്രോഗ്രാമിങ്ങിനെ പിന്തുണക്കുന്നു
   d. ഇവയെയെല്ലാം പിന്തുണക്കുന്നു
11. പ്രോഗ്രാമിങ് മാതൃക എന്നാലെന്ത്? പ്രോഗ്രാമിങ് മാതൃകകളുടെ പേരെഴുതുക.
12. പ്രോസിജറൽ പ്രോഗ്രാമിങ് സമീപനത്തിന്റെ പരിമിതികൾ എന്തെല്ലാം?
13. ഒബ്ജക്റ്റ് ഓറിയന്റഡ് പ്രോഗ്രാമിങ് മാതൃക എന്നാലെന്ത്? OOP ന്റെ അടിസ്ഥാന തത്വങ്ങൾ പട്ടികപ്പെടുത്തുക.
14. C++ ൽ എങ്ങനെയാണു OOP പ്രയോഗിച്ചിരിക്കുന്നത്?
15. എൻക്യാപ്സുലേഷൻ എന്നാലെന്ത്?
16. ക്ലാസ്സും ഒബ്ജക്റ്റും തമ്മിലുള്ള വ്യത്യാസം കണ്ടെത്തുക.
17. ബേസ് ക്ലാസ്, സബ് ക്ലാസ്സ് എന്നാലെന്ത്? ബേസ് ക്ലാസും സബ് ക്ലാസും തമ്മിലുള്ള ബന്ധം എന്താണ്?
18. ഡാറ്റ അബ്സ്ട്രാക്ഷൻ എന്ന ആശയം വിവരിക്കുക. ഒരു ഉദാഹരണം നൽകുക.
19. ഇൻഹെറിറ്റൻസിനെ പറ്റി ഒരു ലഘു വിവരണം എഴുതുക.
20. ഒരു കാർ പ്രവർത്തിപ്പിക്കുന്നതിനായി സ്റ്റിയറിംഗ്, ബ്രേക്ക്, ആക്സിലറേറ്റർ മുതലായവ നമ്മൾ ഉപയോഗിക്കുന്നു. ഇവ ഉപയോഗിക്കുമ്പോൾ ആന്തരികമായി എന്താണ് സംഭവിക്കുന്നത് എന്നു നമുക്കറിയേണ്ടതില്ല. ഏതെങ്കിലും OOP ആശയവുമായി നിങ്ങൾക്കിതിനെ ബന്ധപ്പെടുത്താമോ? വിശദീകരിക്കുക.
21. ഇൻഹെറിറ്റൻസ് എന്നാലെന്ത്? പുനരുപയോഗത്തിന് ഇത് എങ്ങനെ സഹായകമാകുന്നു?

22. പോളിമോർഫിസം എന്നാലെന്ത്? ഇത് വിവരിക്കുവാൻ ഒരു ഉദാഹരണം നൽകുക.
23. OOP എന്ന ആശയം ഉദാഹരണ സഹിതം വിവരിക്കുക.
24. സ്വിച്ചോടു കൂടിയ ഒരു പ്ലഗ് പോയിൻറ് പരിഗണിക്കുക. സ്വിച്ചിന്റെ പ്രവർത്തനം എങ്ങനെയായിരിക്കും എന്നത്, ഏതു സാഹചര്യത്തിലാണത് ഉപയോഗിക്കുന്നത് എന്നതിന്റെയും പ്ലഗ് പോയിന്റിൽ എന്താണ് ഘടിപ്പിച്ചിരിക്കുന്നത് എന്നതിന്റെയും അടിസ്ഥാനത്തിയായിരിക്കും. ഏതെങ്കിലും OOP ആശയവുമായി നിങ്ങൾക്കിതിനെ ബന്ധപ്പെടുത്താമോ? വിശദീകരിക്കുക.
25. 'Horse', 'Fish', 'Bird' എന്നീ ഉപക്ലാസുകളെ ഉൽപ്പാദിപ്പിച്ചിരിക്കുന്ന 'LivingBeings' എന്നൊരു ബേസ് ക്ലാസ് പരിഗണിക്കുക. പരാമർശിച്ചിരിക്കുന്ന എല്ലാ ക്ലാസുകളും ഇൻഹെറിറ്റ് ചെയ്യുന്ന 'Move' എന്നൊരു ഫംഗ്ഷൻ LivingBeings ക്ലാസിലുണ്ടെന്ന് അനുമാനിക്കുക. Horse ക്ലാസിന്റെ ഒബ്ജക്ട് 'Move' ഫങ്ഷനെ വിളിക്കുമ്പോൾ, കുതിക്കുന്നു എന്നായിരിക്കാം സ്ക്രീനിൽ പ്രദർശിപ്പിക്കുന്നത്. അതേസമയം Fish ക്ലാസിന്റെ ഒബ്ജക്ട് അതേ ഫംഗ്ഷനെ വിളിക്കുമ്പോൾ നീന്തുന്നു എന്നായിരിക്കാം സ്ക്രീനിൽ പ്രദർശിപ്പിക്കുക. Bird ന്റെ ഒബ്ജക്ടിന്റെ കാര്യത്തിൽ പറക്കുന്നു എന്നായിരിക്കാം. ഏതെങ്കിലും OOP ആശയവുമായി നിങ്ങൾക്കിതിനെ ബന്ധപ്പെടുത്താമോ? വിശദീകരിക്കുക.