C++

පරිගණක ක්‍රමලේඛණය

මුල සිට සරලව

C++ Computer Programming

(1 කොටස)

පෙරවදන

මෙම පොත මූලිකවම ලියා ඇත්තේ, ප්‍රධාන වශයෙන් Windows පරිගණක මෙහෙයුම් පද්ධතිය තුල, **C++** පරිගණක භාශාව මගින් පරිගණක ක්‍රමලේඛණයේ (Computer programming) යෙදීම ඉගෙනීමට ඇරඹීමේ කැමැත්තෙන් සිටින අය සඳහාය. මේ හෙයින් මෙහිදී කරුණු හැකිතාක් සරල ලෙස දැක්වීමට උත්සාහ ගෙන ඇත. මෙම පොත මගින් ඔබට C++ මගින් පරිගණක ක්‍රමලේඛණයෙහි යෙදීමේදී භාවිතා වන මූලික සංකල්ප හඳුන්වා දෙනු ලැබේ. එසේම, එම සංකල්ප ප්‍රායෝගිකව යොදමින් පරිගණක වැඩසටහනක් (සරල Media Player වැඩසටහනක්) ලියන ආකාරයද පෙන්වා දෙනු ලැබේ. මෙමගින් ඔබට C++ භාවිතා කර මෘදුකාංග ලිවීම පිලිබඳ මුල්ම වැටහීමක් ලැබිය හැක.

සුභ **C++** අනාගතයක්**!**

C++ පරිගණක භාෂාව පිලිබඳ කෙටි හැඳින්වීමක්

C++ යනු **C** පරිගණක භාෂාවෙහි කුඩා සොහොයුරාය. එනම්, C++ සහ C ගොඩනගා ඇත්තේ එකම පදනමක් මතයි. මේ නිසා සිදුවිය යුතු පරිදිම C++ සතුව ඇති ලක්ෂණ බොහොමයක් ම C භාෂාවෙහද තිබෙන ලක්ෂණයි. C යනු ඉතා ප්‍රභල භාෂාවක් බව නොරහසකි. මෙම ප්‍රභලතාවය දැක්වීමේ හොඳම උදාහරණයක් නම් **Unix** පරිගණක මෙහෙයුම් පද්ධතියය. **Unix** සම්පූර්ණයෙන්ම ලියා ඇත්තේ C භාවිතයෙනි. එමෙන්ම **Windows 98** සහ **Windows NT** ද ලියා ඇත්තේ වැඩිවශයෙන්ම C පාවිච්චි කිරීමෙනි. ඒ අනුව, C පදනම් කොටගෙන, C හි වැඩි දියුණුවක් ලෙස නිපදවා ඇති C++ පරිගණක භාෂාව වඩාත් ප්‍රභල භාෂාවක් වී ඇත.

C භාෂාව මෙලොව එලිය දකිනු ලැබුයේ 1970 වසරේදීය. එහි නිර්මාතෘ වූයේ AT&T Bell පර්යේෂණාගාරයේ සේවය කල Dennis Ritchie නම් පරිගණක ක්‍රමලේඛකයාය. C පදනම් කරගෙන C++ නිර්මාණය කරනු ලැබූයේද AT&T Bell හිම සේවය කල දොස්තර Stroustrup නැමැත්තාය. ඔහු තම නිර්මාණය බිහි කිරීම ඇරඹූයේ 1980 ගණන්වලදීයි.

C මෙන්ම C++ භාෂාවද මේවන විට විශාල ජනප්‍රියතාවක් දිනාගෙන ඇති අතර මෙයට සාක්ෂි නම් අප භාවිතා කරන බොහොමයක් ම පරිගණක මෘදුකාංග, සහ පරිගණක ක්‍රීඩා ආදිය C හෝ C++ මගින් ලියා තිබීමය. පරිගණක ක්‍රමලේඛකයන් පැරණි C වෙනුවට C++ භාවිතා කිරීමට හේතුව නම්, C++ මගින් පරිගණක වැඩසටහන් ලිවීම C මගින් වැඩසටහන් ලිවීමට වඩා පහසු වීමය. එමෙන්ම C හි අන්තර්ගත නොවන බොහෝ අංගෝපාංගද C++ තුල දැකිය හැක. මේනිසා, C++ ඉගෙනීමට ඔබ යොදන කාලය ඉතා හොඳ ආයෝජනයක් වනු ඇත.

1. ඔබේ පලමු පියවර

පරිගණක වැඩසටහනක් සහ පරිගණක භාෂාවක් යනු කුමක්ද?

මිනිසෙකු හට පරිගණක යන්ත්‍රයක් ලවා කිසියම් දෙයක් සිදුකරවා ගැනීමට අවශ්‍ය වූ විට ප්‍රථමයෙන් ඔහු කල යුත්තේ කුමක්ද? ඔහු කල යුත්තේ ඔහුට අවශ්‍ය කාර්යය කුමක්දැයි එම යන්ත්‍රයට පැහැදිලි කර දීමයි. උදාහරණයක් ලෙස කිසිවෙකුට පරිගණකය ලවා 68975 යන සංඛ්‍යාව 123 න් ගුණකරගැනීමට අවශ්‍යනම් ඔහු මෙය පරිගණකයට තෙරුම් කර දිය යුතුය. නමුත් ඔබ හිටි අඩියේ පරිගණකය ඉදිරියට ගොස් එයට ඇසෙන ලෙස “කරුණාකර 68975, 123 න් වැඩිකර දෙන්න” වැනි දෙයක් එයට කිවහොත් පරිගණකය සිදුකරන එකම දෙය වනු ඇත්තේ, තමා තුල ඇති විදුලි පංකාවෙන් මුග්ධ ශබ්දයක් නගමින් නොසෙල්වී සිටීම පමණයි. මක්නිසාදයත් පරිගණකය කොතරම් සුපිරි යන්ත්‍රයක් වුවත් එයට ඔබ භාවිතා කරණ මිනිස් බස නොවැටහීමය. ඔබත් පරිගණකයත් මෙහිදී වෙනස් භාෂා දෙකක් භාවිතා කරන, දේශයන් දෙකකට අයත් මිනිසුන් දෙදෙනෙකු වැනිය. එසේනම් ඔබට අවශ්‍ය දෙය පරිගණකය නැමැති මෙම “විදේශිකයාට” පැහැදිලි කර දීමේ ක්‍රමය කුමක්ද? එම ක්‍රමය නම් අප සැබෑ විදේශිකයකුට යමක් තෙරුම් කර දීමේදී භාවිතා කරන ක්‍රමයමයි. එනම් භාෂා පරිවර්ථකයෙකුගේ සහය ලබා ගැනීමය. මෙහිදී ඔබේ පිහිටට එන භාෂා පරිවර්ථකයා නම් ඔබේ

පරිගණකය තුල අන්තර්ගත Calculator වැඩසටහනය. එය පණගන්වා එය හරහා පරිගණකය මගින් ඔබට අවශ්‍ය ගණිත කර්මය සිදුකර ගැනීමට හැකිය. Calculator මෘදුකාංගයට ඔබ ඉහත සංඛ්‍යා යුගල සහ ගුණ කිරීමේ විධානය සැපයූ විට, එය ඔබේ දත්ත සහ විධානය පරිගණකය භාවිතා කරනු ලබන භාශාවට, එනම් ද්විමය සංඛ්‍යා නොහොත් දෙකේ පාදයේ සංඛ්‍යා වලින් සැදි පරිගණක කේත බවට පරිවර්ථනය කර පරිගණකයේ මයික්‍රො ප්‍රොසෙසරයට සපයනු ලබයි. මෙම පරිගණක කේත දෙකේ පාදයෙන් යුතු නිසා, ඒවා binary codes ලෙසින් හැඳන්විය හැකිය. පරිගණකයට දැන් නම් කල යුතු දෙය වැටහෙන නිසා එය ඔබේ සංඛ්‍යා යුගලය ගුණ කොට, ලැබෙන පිලිතුර Calculator වැඩසටහනට නැවත binary codes ලෙසම ලබාදෙයි. Calculator වැඩසටහන මෙම ද්විමය කේත ලෙසින් ඇති දත්තය මිනිසාට හඳුනාගත පහසු දශමය හෙවත් දසවන පාදයේ සංඛ්‍යා වලට හරවා ඔබට දිස්වීමට සලස්වයි. මෙලෙස Calculator නැමැති පරිගණක වැඩසටහන භාශා පරිවර්ථකයෙකු ලෙස ක්‍රියා කරයි. එසේනම්, පරිගණක වැඩසටහනක ක්‍රියාව නම්, මෙසේ මිනිසා සහ පරිගණකය අතර සම්බන්ධයක්, බුද්ධි පාලමක් බැඳීමය. මේනිසා, C++ පරිගණක භාශාව ඉගෙනීමෙන් ඔබ පරිගණක ක්‍රමලේඛකයෙකු (Computer programmer), හෙවත් පරිගණක වැඩසටහන් නිර්මාපකයෙකු වීමට අදහස් කරන්නේ නම්, ඔබ අතගැසිය යුත්තේ පරිගණකය හා මිනිසා අතර මෙම සම්බන්ධය ඇතිකිරීමේ කාර්යයටය. ඔබ මෙම භාශා පරිවර්ථන කාර්යය භාර ගත යුතුය.

ඔබ භාශා පරිවර්ථකයෙකුගේ ක්‍රියාව ඉටු කිරීමට නම්, ඔබට භාශා දෙකක් ගැන අවබෝධය අවශ්‍ය වේ. එනම්, ඔබට මිනිස් බස මෙන්ම පරිගණකයේ බස ගැනද වැටහීමක් තිබිය යුතුය. නමුත් දැන් මෙහිදී ගැටලුවක් මතුවේ. එයනම්, පරිගණකය භාවිතා කරණ ද්විමය සංඛ්‍යා (දෙකේ පාදයේ සංඛ්‍යා) වලින් සැදි භාශා ක්‍රමය සාමාන්‍ය මිනිසෙකුට වටහා ගැනීමට ඉතා දුශ්කරවීමය. අද භාවිතා වන පරිගණක (bit 32 පරිගණක) වටහා ගන්නා භාශාවේ වචනයක් (word) සෑදී ඇත්තේ 1 හා 0 යන ඉලක්කම් 32 ක් විවිධ ආකාරයන්ට එක් වීමෙනි. මෙලෙස, දෙකේ පාදයේ සංඛ්‍යා සාගරයක් ලෙසින් නිමව ඇති මෙම භාශාවෙන් පරිගණක වැඩසටහන් ලිවීම මෙනිසා සැබැවින්ම කටුකය. මේ ගැටලුවෙන් අත්මිදීමේ මාර්ගය එසේනම් කුමක්ද? එය නම්, C++ වැනි ක්‍රමලේඛණ භාශාවක් ඉගෙනීමය. C++ වැනි පරිගණක භාශාවන් නිපදවා ඇත්තේ ඔබව ඉහත සඳහන් කල කටුක ද්විමය සංඛ්‍යාංකිත කේත පිලිබඳව උගැනීමට සිදුවීමේ විපතින් ගලවා ගැනීමටය. C++ මගින් සිදුකරන්නේ ඔබට පරිගණක වැඩසටහන් ලිවීමට හැකිවන ලෙස පහසු භාශා ක්‍රමයක් සැපයීමය. C++ පරිගණක භාශාව සෑදී ඇත්තේ මිනිසාට තෙරුම් ගැනීමට හැකි ඉංග්‍රීසි භාශාවට ලංවූ වචන සමූහයකිනි. මේනිසා ඔබට C++ ඉගෙනීම මගින් වඩා පහසුවෙන් පරිගණක වැඩසටහන් ලිවීමට හැකිවේ.

C++ පරිගණක භාශාව සහ පරිගණකයෙහි ස්වභාවික භාශාව අතර සම්බන්ධය

මෙම සම්බන්ධය විමසා බැලීමට පෙර, “පරිගණකයෙහි ස්වභාවික භාශාව” පිලිබඳව තවදුරටත් විමසීම ප්‍රයෝජනවත්ය. ඔබ කලින් ද කියවූ පරිදි පරිගණකයකට කියවිය හැකි භාශාව සෑදී ඇත්තේ ද්විමය සංඛ්‍යා වලිනි. ද්විමය සංඛ්‍යා හෙවත් binary numbers යනු 1 සහ 0 යන සංඛ්‍යා වලින් සෑදුණු සංඛ්‍යාය. උදාහරණයක් වශයෙන් 100010110 යන සංඛ්‍යාව ද්විමය සංඛ්‍යාවක් ලෙස ගත හැකිය. ඕනෑම භාශාවක් නිර්මාණය වන්නේ වචන වලිනි. පරිගණකයේ භාශාවටද මෙය පොදුය. එහි වචනයක් (word) යනු ද්විමය අංක 32 ක් එක්වී සෑදුණු ඒකකයකි. උදාරහණයක් ලෙස අංක 32 කින් සැදි 10100110101011001010011110110111 යන සංඛ්‍යාව පරිගණකයක එක් වචනයක් ලෙස සැලකිය හැක. නමුත් මෙම 1 හා 0 යන සංඛ්‍යා පරිගණකයට පැමිණියේ කෙසේද? සරල ලෙසින් කියතහොත්, මෙම 1 හා 0 යන සංඛ්‍යා වලින් නිරූපණය වන්නේ පරිගණකය තුල ඇති මයික්‍රොප්‍රොසෙසර් (Microprossesor) පරිපථයෙහි ජනිත වන විද්‍යුත් සංඥාවන් වේ. පරිගණකයේ ක්‍රියාකාරීත්වයට හේතුවන්නේ මේවාය. ප්‍රොසෙසරය තුල අන්තර්ගත ට්‍රාන්සිස්ටර මගින් මෙම විද්‍යුත් සංඥාවන් ඉපදෙයි. 1 මගින් නිරූපණය වන්නේ ට්‍රාන්සිස්ටරයක් හරහා විද්‍යුතය ගමන් කරන අවස්ථාවකි. 0 මගින් නිරූපණය වන්නේ විද්‍යුතය ගමන් නොකරන අවස්ථාවය. මෙම 1 හෝ 0 යන එක් විද්‍යුත් සංඥාවක විශාලත්වය බිට් 1කි (1 Bit). (Bits යනු පරිගණකයක දත්ත ධාරිතාවන් ගණනය කිරීම පිණිස භාවිතා කෙරෙන කුඩාම ඒකකයයි) පරිගණක වැඩසටහක් මගින් සිදුකරන්නේ, මෙම ද්විමය කේතයන් ජනනය කිරීම මගින් පරිගණකයේ මයික්‍රොප්‍රොසෙසරය කිසියම් ක්‍රියාවක් සිදුවීම පිණිස පාලනය කිරීමය. Binary codes වලට ඇති මේ මයික්‍රොප්‍රොසෙසරය පාලනය කිරීමේ යාන්ත්‍රික මට්ටමේ හැකියාව නිසා මෙම කේතයන් සඳහා **Machine codes** හෙවත් යාන්ත්‍රික කේත යන නම භාවිතාවේ. මින් ඉදිරියට පරිගණකයේ භාශාව ගැන කථා කිරීමේදී අප මෙම Machine codes, නැතිනම් යාන්ත්‍රික කේත යන යෙදුම භාවිතා කරමු.

ඉහත ඔබ කියවූ පරිදි පරිගණකයේ භාශාවෙන් ක්‍රමලේඛණයේ යෙදීම යනු මෙම ද්විමය සංඛ්‍යාංකිත කේත හෙවත් යාන්ත්‍රික කේත භාවිතා කිරීමෙන් පරිගණක වැඩසටහන් ලිවීමය. මුල් කාලයේදී ක්‍රමලේඛකයන් විසින් අනුගමනය කරන ලද්දේ මෙම අසීරු ක්‍රමයයි. මෙම අසීරුතාවය හඳුනාගත් මුල් ක්‍රමලේඛකයින් විසින් Assembly නමින් පරිගණක භාශාවක් හඳුන්වා දෙනු ලැබීය. Assembly මගින් සිදුවූයේ යාන්ත්‍රික කේත මගින් සැපයිය යුතු පරිගණක විධානයන් වෙනුවට, ඉංග්‍රීසි වචන වලට සමාන වෙනත් විධානයන් හඳුන්වාදීමය.

එනම්, උදාහරණයක් ලෙස සංකීර්ණ යාන්ත්‍රික කේත වලින් ලියූ පරිගණක ක්‍රමලේඛණයක් (ක්‍රමලේඛණයක් යනු කවර හෝ පරිගණක භාශාවකින් ලියනු ලැබූ පරිගණක විධාන රැසක් ලෙස හැඳින්විය හැකිය) Assembly භාශාවට පෙරලුවහොත්, එය වඩා පහසුවෙන් වටහාගත හැකි, Assembly විධාන වලින් යුතු ක්‍රමලේඛනයක් බවට පත් වෙනු ඇත. මෙපරිදි පරිගණක ක්‍රමලේඛකයන්ට Assembly මගින් වඩා පහසුවෙන් වැඩසටහන් සැකසීමේ අවස්ථාව ලැබිණි. ක්‍රමලේඛකයා තම වැඩසටහන Assembly භාවිතයෙන් ලියා එය යාන්ත්‍රික කේතයන්ට පරිවර්ථනය කිරීමට Assembler නැමැති මෘදුකාංගය පාවිච්චි කරනු ලැබීය. Assembly වැනි, යාන්ත්‍රික කේත පාදක කොටගෙන නිර්මාණය කර ඇති පරිගණක භාශාවන් **Low Level Languages** හෙවත් පහත මට්ටමේ පරිගණක භාශා ලෙස හැඳින්වේ.

කෙසේවෙතත්, Assembly පරිගණක භාශාවද ඉගෙනීමට එතරම් පහසු නොවීය. එසේ වූයේ එය ක්‍රමලේඛණය වඩා පහසුවෙන් තෙරුම් ගත හැකි තත්වයට පත් කලද, එය මගින් කිසියම් පරිගණක ක්‍රියාවක් සිදුකිරීම සඳහා ලිවිය යුතු විධානයන් ප්‍රමාණය අඩු නොවූ හෙයිනි. සරල පරිගණක වැඩසටහනක් වුවද Assembly මගින් ලිවීමට උත්සාහ කළහොත්, එය ඉතා දිගු ක්‍රමලේඛණයක් වනු ඇත. මෙනිසා Assembly මගින් ක්‍රමලේඛණයේ යෙදීමද කාලය බොහෝ ලෙස වැයවන කටයුත්තක් වූයේය. මේ නිසා වඩා පහසු ලෙස උගත හැකි, වඩා සරල, නව පරිගණක භාශාවන් නිර්මාණයට පරිගණක ක්‍රමලේඛකයින් පෙලඹිණි. මේ අයුරින් නිපදවුණු භාශාවන්ය **C** සහ **C++**. Assembly මගින් සිදුවන්නේ යාන්ත්‍රික කේත සඳහා ආදේශක වන විධාන සැපයීම වූ අයුරින්, **C** / C++ මගින් සිදුවන්නේ එම Assembly විධාන රැස බැගින් එක් කොට ඒවා සඳහා ආදේශයන් වන වඩා සරල පරිගණක විධානයන් හඳුන්වාදීමයි. මේ නිසා යාන්ත්‍රික කේත භාවිතයෙන් ලියූ වැඩසටහනක් Assembly මගින් ලියූ විට එය වඩා සරල වන අතර, එයම **C** / C++ වලින් ලියූ විට එය වඩාත් කෙටි සහ සරල වේ. මේ අනුව **C** / C++ වැනි පරිගණක භාශාවන් තනා ඇත්තේ Assembly භාශාව පාදක කොට ගෙන, එය මතය. මේනිසා ක්‍රමලේඛකයෙකු විසින් C++ මගින් පරිගණක වැඩසටහනක් ලියූ විට ඔහු එය පලමුව Compiler නැමැති මෘදුකාංගය භාවිතා කොට Assembly භාශාවට පරිවර්ථනය කල යුතුය. ඉන්පසුව ඔහු එම පරිවර්ථනයද නැවත Assembler මෘදුකාංගය මගින් යාන්ත්‍රික කේත බවට පරිවර්ථනය කල යුතුය. මෙය සිදුකිරීම ඉතා පහසු දෙයකි. එය සිදුකරන ආකාරය ඉදිරියේදී ඔබ ඉගෙන ගනු ඇත. C++ වැනි භාශා Assembly භාශාව පාදක කොට සකසා ඇති ඉහල මට්ටමේ භාශා නිසා ඒවා පොදුවේ **High Level Languages** ලෙස හැඳින්වේ.

පරිගණක වැඩසටහනක් සැකසීමට, මූලික වශයෙන් ඔබට පහත දැක්වෙන දෑ අවශ්‍ය වෙනු ඇත:

1. පරිගණක වැඩසටහන් සකසන මෘදුකාංගයක් - Program editer.

අප පරිගණක වැඩසටහනක් ලියනු ලබන්නේ මෙම වර්ගයේ මෘදුකාංගයක් තුලය. ඔබ Windows පරිගණක පාවිච්චි කරන්නේ නම්, ඔබට තොරාගත හැකි හොඳම වැඩසටහන් සකසන මෘදුකාංගය වන්නේ Microsoft Visual C++ 6.0 ය. මෙය ඔබට පහසුවෙන් වෙලඳපොලෙහි සොයාගත හැකිය.

2. Compiler

Compiler මෘදුකාංගය මගින් සිදුකරන්නේ C++ වැනි High level language සඟයට වැටෙන පරිගණක භාශා මගින් ලියන ලද පරිගණක වැඩසටහන්, පරිගණකයට වැටහෙන ද්විමය සංඛ්‍යාවන් බවට පරිවර්ථනය කිරීම බව ඔබ ඉහත දී කියවූවා. ඔබ Microsoft Visual C++ 6.0 මිලදී ගත්විට, එහි Compiler මෘදුකාංගයක් ද අඩංගු වනු ඇත.

3. Debugger

මෙය, ඔබ ලියූ පරිගණක වැඩසටහනක දෝෂ ඇත්නම් ඒවා හඳුනාගැනීමට පහසුකම් සලසන මෘදුකාංගයකි. Microsoft Visual C++ 6.0 තුල Debugger මෘදුකාංගයක් ද අන්තර්ගත වේ. Visual C++ debugger පහසුකම භාවිතයට ගන්නා ආකාරය පොතෙහි **Appendix** කොටසෙහි කෙටියෙන් දැක්වේ.

මෙම පොතෙහි මතුවට අඩංගු වන සියලු වැඩසටහන් ලියා ඇත්තේ Microsoft Visual C++ 6.0 උපයෝගී කරගැනීමෙනි. මේනිසා මෙතැන් සිට මෙම පොතෙහි අඩංගු ක්‍රියාකාරකම් සිදුකිරීම සඳහා ඔබ එම මෘදුකාංගය

ඔබේ පරිගණකය තුල install කරගත යුතුය. Visual C++ 6.0 install හෙවත් ස්ථාපනය කරන ආකාරය **Appendix** කොටසේදී පැහැදිලි කර ඇත.

ඔබේ පලමු C++ පරිගණක වැඩසටහන

පහත දැක්වෙන්නේ C++ භාවිතයෙන් ඔබට ලිවිය හැකි ඉතාම සරල, ක්‍රියාකාරී වැඩසටහනකි. පලමුවෙන් ම කිවයුතු වන්නේ මේ වැඩසටහනෙහි අඩංගු සියලුම දේ වටහා ගැනීමට ඔබ උත්සාහ නොකල යුතු බවයි. එය ඉදිරිපත් කර ඇත්තේ හුදෙක් C++ මගින් ලියන ලද පරිගණක වැඩසටහනක් මෙවැන්නකියි හඳුන්වාදීම පිණිසය. එම නිසා එහි අඩංගු සියලුම දෑ සම්පූර්ණයෙන් ම වටහාගැනීමට උත්සාහ නොකරන්න. ඔබට දැනට නොවැටහෙන දෑ, මෙම පොත ඉදිරියට කියවීමත් සමග අවබෝධ වීමට පටන්ගනු ඇත:

```
=====================================

// hello.cpp

#include<iostream.h>

void main()
{
      cout << “Hello beautiful world!!!” << endl;
}

=====================================
```

අප දැන් සිදු කිරීමට යන්නේ මෙයයි: පලමුව Microsoft Visual C++ විවෘත කරමු. ඉන්පසු ඉහත වැඩසටහන ඒ අයුරින්ම එහි යතුරු ලියනය කරමු. එනම්, අපගේ වැඩසටහන විසින් කල යුතු දෑ, C++ විධානයන් රැසකින් ලියමු. මීලඟට අප කරන්නේ මෙම C++ විධානයන් යාන්ත්‍රික කේතයන් බවට පරිවර්ථනය කිරීමය (එසේ නොකලහොත් පරිගණකය එම විධානයන් හඳුනොගනු ඇත). එනම්, අප C++ පරිගණක වැඩසටහන compile කල යුතුය. අවසානයේදී අප එම පරිවර්ථනය කරන ලද පරිගණක වැඩසටහන ක්‍රියාත්මක කරමු. මෙය සිදුකරන ආකාරය පියවරින් පියවර පහතින් දැක්වේ.

මුලින්ම කල යුතු වන්නේ ඔබේ වැඩසටහන අන්තර්ගත කිරීම සඳහා Visual C++ තුල නව **Project** එකක් නිමවීමය. Project එකක් යනු කිසියම් පරිගණක වැඩසටහනක් සඳහා ප්‍රයෝජනයට ගැනෙන සියලු **C++** files අන්තර්ගත කිරීම සඳහා තනනු ලබන ඒකකයකි. ඉහතින් අප දුටු වැඩසටහනෙහි නම් තිබෙන්නේ එක් file එකක් පමණක් වූවද පසුව ඔබට දැකීමට ලැබෙන පරිදි පරිගණක වැඩසටහනක් යනු C++ files සමූහයක එකතුවකි. C++ files යනු “.cpp” යන extension එක සහිත computer files ය.

1. පලමුවෙන්ම, මෙම පොතෙහි අඩංගු වන සියලු වැඩසටහන් තැන්පත් කිරීම පිණිස, ඔබේ පරිගණකයෙහි දෘඪ තැටියෙහි ඔබ කැමති ස්ථානයක “VC++ book” නමින් නව folder එකක් තනන්න.

2. දැන් Microsoft Visual C++ 6.0 මෘදුකාංගය විවෘත කරන්න.

3. එහි **File** නැමැති menu එකෙහි ඇති New නැමැති item එක **(File > New)** select කරන්න.

4. මතුවෙන Dialog Box එකෙහි **Projects** window එක තුල ඇති **Win32 console application** නැමැති icon එක select කරන්න. (රූපය 1.1) මෙසේ සිදුකලයුත්තේ අප මේ අවස්ථාවේදී සෑදීමට බලාපොරොත්තු වන්නේ console වර්ගයේ පරිගණක වැඩසටහනක් නිසාය. මෙය මඳක් පහැදිලි කරගැනීමට අවශ්‍ය දෙයකි.

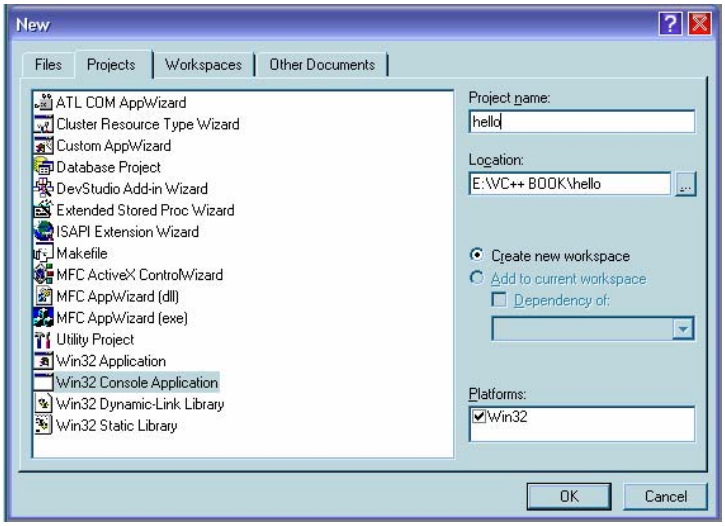
ප්‍රධාන වශයෙන් සලකා බලන විට පරිගණක වැඩසටහන් වර්ග දෙකකට බෙදේ.

එනම්,

(1) Console applications හෙවත් Terminal (DOS) window එක හරහා ක්‍රියාත්මක කලයුතු පරිගණක වැඩසටහන් සහ,

(2) GUI (Graphical User Interface) එකක් හෙවත් රූපමය මතුපිටක් සහිත පරිගණක වැඩසටහන් යනුවෙනි.

ඉහත අප දුටුවේ මෙම Console ඝණයට වැටෙන සරල වැඩසටහනකි. මේවාට පාවිච්චි කරන්නා සමග අන්තර් ක්‍රියා කිරීම පිණිස GUI එකක් හෙවත් රූපමය මතුපිටක් නොමැත. එය පාවිච්චි කරන්නා සමග කටයුතු කරන්නේ Terminal හෙවත් DOS window එක හරහාය. මේ සඳහා ඔහුට පරිගණකයේ යතුරු පුවරුව උපකාරී වේ. අනෙක් අතට GUI එකකින් යුතු වැඩසටහනක් යනු රූපමය මතුපිටක් මගින් වැඩසටහන පාවිච්චි කරන්නා සමග අන්තර්ක්‍රියා සිදුකරන මෘදුකාංගයකි. Microsoft Windows තුල ඔබ භාවිතයට ගන්නා සියලු ජනප්‍රිය පරිගණක වැඩසටහන් මේ ඝණයට අයිති වේ. Visual C++ ද අනිවාර්යයෙන්ම GUI application එකකි.



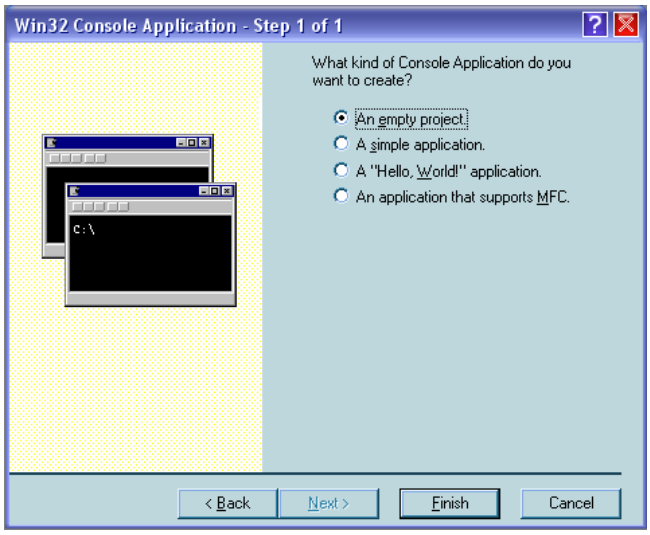
(රූපය 1.1)

5. **Location** නැමැති Text box එක මගින් සිදුවන්නේ ඔබ සාදන project එක දෘඩ තැටියේ ගබඩා කලයුතු ස්ථානය දැක්වීමය. දැන් Location text box එකට ඉදිරියෙන් ඇති Button එක click කර, දෘඩ තැටියෙහි ඉහත ඔබ තැනූ "VC++ book" folder එක select කරන්න.

6. **Project name** text box එක තුල අප සෑදීමට නියමිත වැඩසටහනේ නම ලෙස "hello" යනුවෙන් type කරන්න. (රූපය 1.1)

7. **OK** button එක ඔබන්න. මෙවිට "hello" යන නමින් යුත් folder එකක් "VC++ book" folder එක තුල Visual C++ මගින් නිපදවනු ඇත.

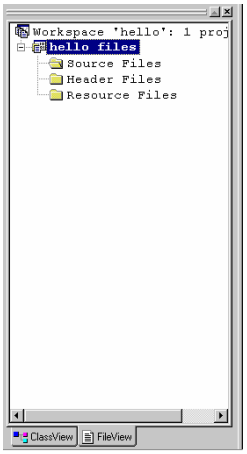
8. දැන් 1.2 රූපයේ දැක්වෙන dialog box එක ඔබ ඉදිරියේ දිස්වනු ඇත.



(රූපය 1.2)

9. ඉහත රූපයේ ආකාරයට එම dialog box එකෙහි ඇති **"An empty project"** යන option එක select කරන්න. එහිදී අප VC++ වෙත පවසන්නේ, දැනට කිසිදු file එකක් අඩංගු නොවන, හිස් project එකක් සාදන ලෙසය. ඉන්පසු **Finish** button එක click කරන්න. මෙවිට තවත් dialog box එකක් open වේවි. එයද OK කරන්න. දැන් **"hello"** නැමැති හිස් project එකක් ඔබ සාදා ඇත.

Visual C++ window එකෙහි දකුණු පස ඇති පටු window එක දෙස දැන් බලන්න (රූපය 1.3). **මෙම කොටස හැදිවෙන්නේ workspace යනුවෙනි.** එම window එකෙහි පහල අන්තයෙහි file view නැමැති tab එකක් ඇති ආකාරය බලන්න. එම tab එක මත click කරන්න. එවිට ඔබ මොහොතකට පෙර සෑදූ project එක මෙම file view නැමැති window එකතුල ඔබට දැකගත හැකිය (රූපය 1.3). එහි "hello files" යන්නට යටින් ඇත්තේ ඔබේ project එකට අදාල සියලු files ය. ඔබ සාදන C++ files අන්තර්ගත වන්නේ එහි "source files" යන නමින් දැක්වෙන folder එක තුලය. "Header files" යන folder එකතුල අන්තර්ගත වන්නේ ඔබ තවම හඳුනා නොගත් Headers නමින් හැඳින්වෙන files විශේෂයකි.



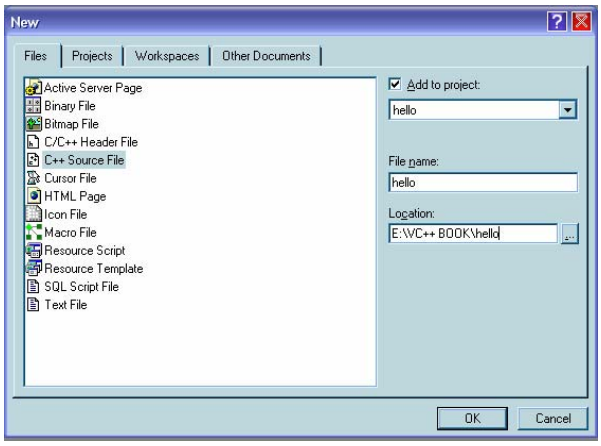
(රූපය 1.3)

ඔබ මෙම folders මත double click කර බැලුවහොත් ඒවායේ කිසිවක් අඩංගු නොවන බව ඔබට දැකිය හැකි වනු ඇත.

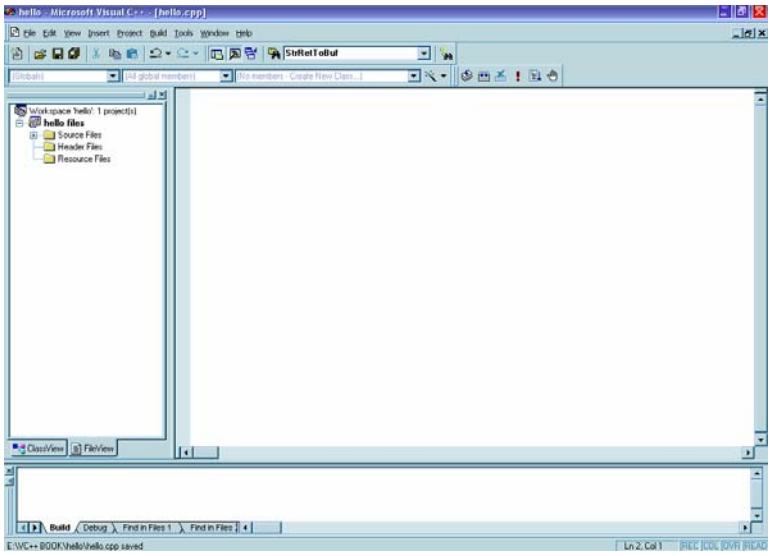
දැන් ඔබ සැරසෙන්නේ අපගේ හිස් project එකට C++ file එකක් ඇතුල්කර ඉහත අප දුටු පරිගණක වැඩසටහන එම file එකතුල type කිරීමටය. පහත දැක්වෙන ලෙස මෙය සිදුකරන්න:

1. **File > New** menu එක මගින් **New** dialog box එක open කරන්න (1.4 රූපය).

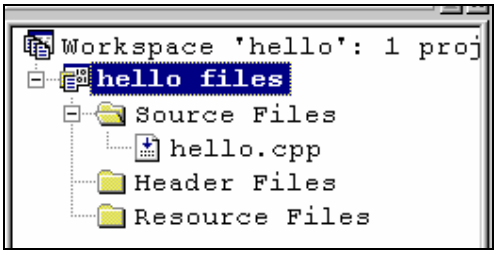
2. 1.4 රූපයේ දැක්වෙන පරිදි එහි **File** window එකෙහි ඇති **"C++ source file"** නැමැති item එක select කරන්න. ඉන්පසු Dialog box එකෙහි **"Add to project"** නැමැති check box එක check කරන්න. **"File name"** text box එක තුල දැන් අප නිමවන C++ file එකෙහි නම ලෙස "hello" යන්න type කර dialog box එක OK කරන්න. එවිට 1.5 රූපයේ පෙනෙන ලෙස නව හිස් C++ file එකක් Visual C++ තුල වම් පස ඇති විශාල window එකතුල open වේවි. **මෙම window එක හඳුන්වන්නේ "Editor window" යන නමිනි**. මේ අවස්ථාවේදී ඔබ workspace window එක දෙස බැලුවහොත් source files යන folder එකතුල ඔබ මේ දැන් නිමවූ **"hello.cpp" යන** file එක තිබෙනු දැකිය හැකිවනු ඇත (රූපය 1.6).



(රූපය 1.4)



(රූපය 1.5)

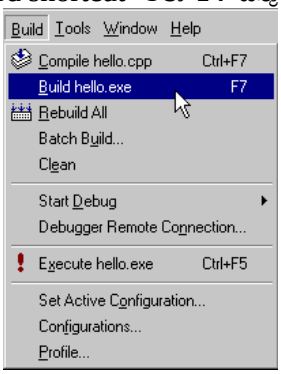


(රූපය 1.6)

3. දැන් ඔබ කලයුත්තේ ඉහත ඔබ උගත් පරිගණක වැඩසටහන, editor window එක තුල open වී ඇති මෙම නව document එක හෙවත් “hello.cpp” තුල type කිරීමය. දැන් එය සිදු කරන්න.

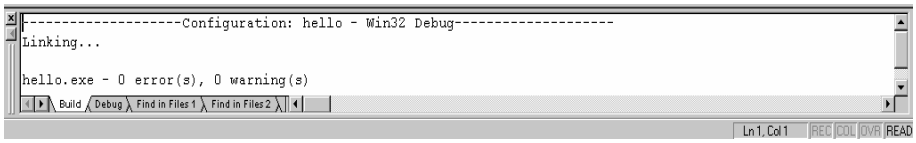
දැන් ඔබ C++ භාශාව මගින් සරල වැඩසටහනක් ලියනු ලැබුවා. වෙනත් වචන වලින් කියතහොත් ඔබ කිසියම් දෑ රැසක් සිදුකරණ ලෙස C++ විධාන රැසක් මගින් පරිගණකයට අණකලා. නමුත් C++ මගින් ඔබ කල ඒ අණ කිරීම් පරිගණකයට නොවැටහෙන නිසා ඔබ පෙරදී කියවූ ලෙස වැඩසටහන compile කිරීමෙන් එය ද්විමය කේත බවට පරිවර්ථනය කල යුතුයි. මෙය පහත දැක්වෙන ආකාරයට සිදුකරන්න.

1. Visual C++ හි **Build** Menu එකෙහි **“Build hello.cpp”** නැමැති item select කරන්න (රූපය 1.7). එසේ නැතිනම් එම Menu item එකෙහි keyboard shortcut වන **F7** යතුර ඔබන්න.



(රූපය 1.7)

2. දැන් ඔබේ පරිගණකයේ වේගයට අනුරූප කාලයකදී දැන් hello.cpp file එක compile වෙනු හෙවත් ද්විමය කේත බවට පරිවර්ථනය වනු ඇත. මෙවිට ඔබ දකින තවත් දෙයක් වන්නේ Visual C++ හි පහත කොටසේ ඇති දිගු window එකෙහි යම්කිසිවක් සටහන් වන බව හෙවත් print වන බවය. **මෙම pane එක හඳුන්වන්නේ “Output window” යන නමිනි** (1.8 රූපය).



(රූපය 1.8)

Output window එක මගින් කාර්යයන් කීපයන් සිදුකරයි. එකක් නම් කිසියම් file එකක් compile කිරීමේදී එයට අදාල විස්තර print කිරීම හෙවත් දිස්වීමට සැලැස්වීමයි. ඕඑයට අදාල විස්තරය යනුවෙන් ප්‍රධාන කොටම හැඳින්වෙන්නේ, compile ක්‍රියාවලියට ලක්වන පරිගණක වැඩසටහනෙහි ව්‍යාකරණ දෝශ හෙවත් **syntax errors** ආදිය ඇතිනම් එම දෝශයන්ද, එම දෝශයන් වැඩසටහනෙහි පවතින ස්ථානයන් පිලිබඳ විස්තර ආදියයි.
(මේ පිලබඳ වැඩි විස්තර දැනගැනීමට පොතෙහි **Appendix** කොටසේ ඇති **“පරිගණක වැඩසටහන් වල දෝශ - Programming errors”** නැමැති කොටස කියවන්න)

ඉහත වැඩසටහන ඔබ නිවැරදිව යතුරුලියනය කරනු ලැබුවානම්, වැඩසටහන දෝශ වලින් තොර බව 1.8 රූපයේ දැක්වෙන ආකාරයේ output එකක් මගින් output window එක ඔබට දක්වනු ඇත.

අප දැන් අපගේ **hello නැමැති** පරිගණක වැඩසටහන **compile** කර **අවසානයි. Microsoft Visual C++ විසින්** ඔබ ලියන ලද වැඩසටහන ද්විමය ඛේත වලින් ලියනු ලැබුවා.

දැන්, මඳකට ඉහතදී ඔබ සාදන ලද “VC++ book” folder එක Windows explorer මගින් විවෘත කරන්න. එහි අඩංගු “hello” නැමැති folder එක ඔබ දකිනු ඇත. එයද විවෘත කරන්න. ඉන්පසු එය තුල ඇති “Debug” නැමැති folder විවෘත කරන්න. එහි ඔබට “hello.exe” නැමැති file එකක් දැකගත හැකිවනු ඇත. ඔබ ලියූ වැඩසටහන ද්විමය ඛේතයන් ගෙන් ලියවී ඇත්තේ මෙම file එක තුලය. වෙනත් වචන වලින් කිවහොත්, මේ නම් ඔබ ලියූ පරිගණක වැඩසටහනෙහි ක්‍රියාකාරී අවස්ථාවයි. මේ නිසා ඔබට මෙම file එක open කිරීමෙන්

ඔබේ වැඩසටහන ක්‍රියාත්මක කල හැක. නමුත් මෙහිදී අප “hello.exe” මත click කරමින් එය ක්‍රියාත්මක නොකර, Visual C++ මගින් ම එය ක්‍රියාත්මක කරවමු. එය පහත ලෙස එය සිදුකරන්න:

12. visual C++ හි, **Build > Execute hello.exe** item එක select කරන්න. නැතිනම් Ctrl + F5 ඔබන්න. එවිට DOS හෙවත් ඔබේ පරිගණකයේ Terminal window එක තුල “hello.exe” file එක open වනු ඇත. මෙසේ වන්නේ ඔබ නිමවූයේ DOS තුල ක්‍රියාත්මක වන console වැඩසටහනක් වන නිසාය. මේ පොතෙහි ඉදිරියට හමුවෙන සියලුම වැඩසටහන් මේ වර්ගයේ console වැඩසටහන් වනු ඇත. දැන් Terminal window එකෙහි `“Hello beautiful world!!!”` යන්න print ව තිබෙනු ඔබ දකිනු ඇත.

මේ අයුරින් ඔබේ පලමු පරිගණක වැඩසටහන ඔබ සාර්ථකව නිමකර ඇත. දැන් අප මෙම කුඩා වැඩසටහන නිමවී ඇත්තේ කෙසේදැයි සලකා බැලීම අප අරඹමු. එහි කොටස් එකින් එක ගෙන පහතින් විස්තර කර ඇත:

```
==================

// hello.cpp

==================
```

ඉහත දැක්වෙන්නේ ඔබ ලියූ වැඩසටහනෙහි ඉහලින්ම ඇති පේලියයි. පලමුවෙන්ම ඇති ’’**//**’’ සලකුණ හැඳින්වෙන්නේ comment එකක් යනුවෙනි. අප එය C++ ක්‍රමලේඛණයක් තුල යොදන්නේ එයින් ඇරඹෙන පේලිය එම වැඩසටහනෙහි ක්‍රියාකාරී කොටසක් නොවිය යුතු විටදීය. මෙනිසා ඉහත වැඩසටහනෙහි **// hello.cpp** යන පේලිය, වැඩසටහනෙහි ක්‍රියාකාරී කොටසක් නොවේ. **hello.cpp** යනු හුදෙක් අප මෙම වැඩසටහන ලියනු ලබන file එකෙහි නමයි. වැඩසටහන compile කරන අවස්ථාවේදී ’’**//**’’ සලකුන හමුවූ විට compiler මෘදුකාංගය එම පේලිය compile නොකර නොසලකා හරියි. ඔබට අවශ්‍ය නම් මෙම පේලිය ඉවත් කර වැඩසටහන ක්‍රියාත්මක කල හැකිය. එයින් කිසිම වෙනසක් සිදු නොවේ.

මෙවැනි comments ක්‍රමලේඛණයක් තුල යොදන්නේ හේතු කීපයක් මතය. එක් ප්‍රධාන හේතුවක් නම් කිසියම් ක්‍රමලේඛණ කොටසක් මගින් කෙරෙන කාර්යය වාර්ථා කර තැබීමය. මෙලෙස වාර්ථා කර තබන්නේ, ඔබ ලියන වැඩසටහන දිගු සහ සංකීර්ණ එකක් වන්නේ නම්, එහි අඩංගු කිසියම් කේත රැසකින් සිදුකරන කාර්යය කාලය ගත වීමත් සමඟ ඔබට අමතක වී යෑමට පිලිවන් හෙයිනි. ක්‍රමලේඛණයක ඇති එවැනි සංකීර්ණ ස්ථාන වලදී, එම ස්ථානයේ අඩංගු කේතයන්ගෙන් සිදුවන්නේ කුමක්දැයි comment එකක් මගින් ලේබල් කර තැබීම පසුවට ඉතා ප්‍රයෝජනවත් වනු ඇත. තවත් හේතුවක් නම්, ඔබ ලියූ ක්‍රමලේඛනයක් වෙනත් ක්‍රමලේඛකයෙකු විසින් කියවනු ලබන විටදී, ඔබ ලියූ පරිගණක කේතයන්ගෙන් සිදුවන්නේ කුමක්දැයි ඔබ එහි තබා ඇති comments ආධාරයෙන් ඔහුට වඩා පහසුවෙන් වටහා ගත හැකි වීමය.

```
======================

#include<iostream.h>

======================
```

ඉහත වැඩසටහන වැනි ඉතාම සරල පරිගණක වැඩසටහනකදී පවා විශාල කාර්යයන් ප්‍රමාණයක් ඔබට නොපෙනෙන ලෙස compiler මෘදුකාංගය විසින් ඉටුකරයි. මේ කාර්යයන් සිදුවන්නේ ණිතිරයට යටින්ඩය. මෙම තිරයට යටින් කෙරෙන කාර්යයන් ගෙන් සමහරක් බොහෝවිට සෑම වැඩසටහනකටම පොදු දේය. උදාහරණයක් ලෙස, පරිගණකය මත ධාවනය වන සෑම වැඩසටහනකට ම දත්ත ඇතුල් කරගැනීමේ හා පිටකිරීමේ (data input-output) හැකියාව අත්‍යවශ්‍යය. එනිසා එය සෑම වැඩසටහනකටම පොදු දෙයකි. මේනිසා පරිගණක ක්‍රමලේඛකයෙකුට, මේ පොදු ක්‍රියාකාරීත්වයන් ඔහු ලියන සෑම වැඩසටහනක් තුලදීම නැවත නැවතත් ක්‍රමලේඛණය කිරීමට සිදුවේ. මෙය ක්‍රමලේඛකයෙකුගේ කාලය අපතේ යවයි. මෙයට පිලියමක් වශයෙන්, මෙම පොදු වූ කාර්යයන් සියල්ල ක්‍රමලේඛණය කර පුස්ථකාලයන් හෙවත් C++ Libraries තුල ගබඩා කර තිබේ. මෙනිසා Visual C++ භාවිතයට ගන්නා ක්‍රමලේඛකයන් හට, සෑම පරිගණක වැඩසටහනකටම පොදුවූ එම ක්‍රියාකාරීත්වයන් තමන් විසින් නැවත නැවත ක්‍රමලේඛණය නොකර Visual C++ Libraries තුලින් පහසුවෙන් ලබාගත හැක. වෙනත් ආකාරයකට කියතහොත් එම libraries තුල අඩංගු වන පරිගණක කේතයන් අපගේ වැඩසටහන් තුල පාවිච්චියට ගත හැකිය.

මේ ආකාරයට, ඉහත **#include <iostream.h>** යනුවෙන් දැක්වෙන්නේ **iostream** යන Library එක අප ලියන වැඩසටහනකට ඇතුල් කරන ආකාරයයි. **iostream** යනු input-output stream යන්නයි. වැඩසටහනෙහි ඉහත කේතයට පසුව අඩංගු වන `cout` හා `endl` යන විධාන යුගල ගබඩා කර ඇත්තේ මෙම **iostream** library එක තුලයි. එම library එක වැඩසටහනට ඇතුල් නොකලහොත්, අපට එම විධානයන් පාවිච්චියට ගැනීමට නොලැබෙනු ඇත.

#include යන්න directive එකක් ලෙස හැඳින්වේ. **#include directive** එකෙහි කාර්යය නම් අප ලියනා පරිගණක වැඩසටහනකට, පිටතින් යම් ක්‍රියාකාරීත්වයක් ඇතුල් කිරීමය. මෙහිදී එය **iostream** library එක වැඩසටහන තුලට ඇතුල් කරයි.

```
==================
void main()

==================
```

ඉහත දැක්වෙන්නේ ඔබට හමුවන පලමුවන C++ function එකයි. Function එකක් යනු සරලව කියතහොත්, C++ පරිගණක කේත රැසක් අඩංගු කර නිමවන ඒකකයකි. C++ වැඩසටහනක් ක්‍රියාත්මක වීම ඇරඹෙන්නේ මෙම `main()` නැමැති functions එකිනුයි. main() යටින් ඔබ සඟල වරහන් යුගලක් දකිනු ඇත. කිසියම් function එකක් තුල අඩංගු විය යුතු C++ කේතයන් අන්තර්ගත විය යුත්තේ එම function එකෙහි නමට පසුව එන ඉහත වැනි සඟල වරහන් තුලය. Functions පිලිබඳ වැඩි විස්තර ‘Functions’ පරිච්ඡේදයේදී සලකා බලමු.

```
=============================================

cout << “Hello beautiful world!!!” << endl;

=============================================
```

මෙම code එක මගින් සිදුවන්නේ `“Hello beautiful world!!!”` යන්න Terminal window එක මතට output කිරීමයි.

`cout <<` යනු මෙම කාර්යය ඉටුකරනු ලබන විධානයයි.

`endl <<` යනු DOS window එක තුල print විය යුතු වචන හෝ සංඛ්‍යා පෙලක් අවසානය විය යුතු ස්ථානය තීරණය කරන කේතයයි. මෙම කේතය අන්තර්ගත වන්නේ main() හි සඟල වරහන් තුල වීම නිසා, මෙය main function එකට අයත් කේතයක් බව ඔබට වැටහෙනු ඇත. තවද ඔබ දකින දෙයක් වන්නේ ඉහත කේතය අවසානයේ ඇති “ **;** ” ලකුණ හෙවත් Semicolon character එකය. Semicolon එකින් දක්වන්නේ කිසියම් Expression එකක අවසානයයි. Expression එකක් යනු Operators නමින් හැඳින්වෙන වස්තූන් ගෙන් සැදි පරිගණක කේතයකි. Expressions සහ Operators යනු මොනවාදැයි ඔබට ඉදිරියේදී ඉගෙනීමට ලැබෙනු ඇත. දැනට, ඉහත පරිගණක කේතය expression එකක් බව පමණක් දැනගන්න. Semicolon එකක් මගින් සිදුවන්නේ ඉහත වැනි expression එකක් අවසන්වන ස්ථානය දැක්වීම බවද මතක තබාගන්න.

Whitespaces

Whitespaces යනුවෙන් හැඳින්වෙන්නේ පරිගණක ක්‍රමලේඛණයක කිසිවක් සඳහන් නොවෙන හිස් ඉඩ ප්‍රමාණයන්ය. ඉහත වැඩසටහනෙහි #include<iostream.h> යන්නටත් void main() යන්නටත් අතරින් ඇත්තේ whitespace එකකි. එමෙන්ම { ටත්, cout << "Hello beautiful world!!!" << endl; යන කේතයටත් අතරින් ඇත්තේ whitespace එකකි. නමුත් ඇත්ත වශයෙන්ම C++ පරිගණක භාශාව, පරිගණක ක්‍රමලේඛණයක ඇති මෙම whitespaces ගැන වැඩි තැකීමක් නොකරයි. එනම්, ඔබට අවශ්‍ය නම් කිසියම් පරිගණක වැඩසටහනක වැඩි කොටසක් කිසිම whitspace එකකින් තොරව වූවද ලිවිය හැක. උදාහරණයක් ලෙස, ඉහත වැඩසටහනෙහි main function එක whitespaces වලින් තොරව ලිවූ හොත් එය පහත ලෙස දිස්වනු ඇත:

```
============================================================

void main() { cout << "Hello beautiful world!!!" << endl;}

============================================================
```

ඉහත කේතය දෙස බැලීමෙන් ඔබට පැහැදිලිවන දෙයනම්, whitespaces භාවිතයෙන් පරිගණක වැඩසටහනක පැහැදිලිතාවය වැඩිවන බවය. සියල්ල එක පේළියකට ලිවීමෙන් වැඩසටහනින් සිදුවෙන දෙය අපහැදිලිවී යයි. එම නිසා පරිගණක වැඩසටහන් ලිවීමේදී ඔබ සෑමවිටම සුදුසුලෙස whitespaces යොදාගන්න.

හොඳයි, අපි ඉහතදී ඉතාම සරල C++ වැඩසටහනක් ලියන්නේ කෙසේදැයි දුටුවා. එම වැඩසටහනෙහි ඇති ප්‍රධාන කොටස් පිලිබඳව යම් අවබෝධයක් ඔබ ලබන්නට ඇති.

RAM (Random Access Memory) මතකය

පරිගණක ක්‍රමලේඛකයෙකු ලෙස ඔබට RAM මතකය පිලිබඳ යම් හෝ අවබෝධයක් තිබිය යුතුය.

RAM මතකය පවතින්නේ ඔබේ පරිගණකයේ RAM චිපය තුලයි. ඔබේ පරිගණකයෙහි යම් මොහෙතක ක්‍රියාත්මක කර ඇති වැඩසටහන් සියල්ලම රැඳි ඇත්තේ මෙම RAM මතකය තුලයි. එම සෑම වැඩසටහනක්ම RAM තුල පවතින මතකයෙන්, එම වැඩසටහනට විශේෂ වූ බයිට ධාරිතාවක් පරිභෝජනය කරයි. (බයිටය (byte) යනු පරිගණක මතක ධාරිතා මනින කුඩාම ඒකකය වේ.) උදාහරණයක් ලෙස කිසියම් මොහොතක ඔබ Windows Media Player වැඩසටහන ක්‍රියාත්මක කරගෙන සංගීතයට සවන් දෙන අතර Internet Explorer වැනි වැඩසටහනක් උපයෝගී කරගෙන අන්තර්ජාලයේ සැරිසරණවා යැයි ද, තෙවැනි වැඩසටහනක් ලෙස Disk Defragmenter වැඩසටහන පසුබිමෙහි තම ක්‍රියාවෙහි නිරත වනවායැයි ද සිතන්න. මෙහිදී මෙම වැඩසටහන් තුනම RAM මතකයෙන් තමාට අවශ්‍ය බයිට කොටස පරිභෝජනය කරමින් ඒ තුල ජීවත් වෙයි. මේ ආකාරයට, ඔබ විසින් ලියනු ලබන වැඩසටහනක්ද ක්‍රියාත්මක අවස්ථාවේදී පවතින්නේ මෙම RAM මතකය තුලයි. RAM මතකය “තාවකාලික මතකය” ලෙසද හැඳින්වේ. එයට හේතුව නම් RAM මතකය තුල ඔබට කිසිවක් ස්ථිර ලෙස ගබඩා කල නොහැකි වීමය. එහි ඇති දත්ත සුරක්ෂිතව පවතින්නේ පරිගණකය ධාවනය වන කාලය අතරතුරදී පමණය. මෙයට හේතුව නම්, RAM චිපයෙහි දත්ත රැඳී පවතින්නේ එයට විද්‍යුතය සැපයෙන තාක් කල් පමණක් වීම හේතුවෙනි. RAM චිපයෙන් විදුලිය විසන්ධි කල විගසම එහි ගබඩා වී ඇති සියලු දත්ත එයට අහිමි වී යයි. ඔබට ස්ථිර ලෙස කිසිවක් ගබඩා කරගැනීමට අවශ්‍ය නම් එය පරිගණකයේ දෘඪ තැටියෙහි (Hard Disk) ගබඩා කරන්න. දෘඪ තැටියෙහි ගබඩා කර ඇති දත්ත විදුලිය විසන්ධි කිරීමකින් මැකී නොයන්නේ එහි මතකය රැඳී පවතින්නේ ඇත්තේ විද්‍යුතයේ ආධාරයෙන් නොව, චුම්භක බලයෙන් වීම හේතුවෙනි.

නමුත් පරිගණකයක මතකය මෙලෙස RAM මතකය හා ස්ථිර මතකය හෙවත් ලෙස දෙආකාරයකින් පවතින්නේ මන්ද? පරිගණකය ක්‍රියාත්මක වීමේදී RAM චිපය පාවිච්චි නොකර දෘඪ තැටියෙහිම අවශ්‍ය දත්ත ගබඩා නොකරන්නේ මන්ද? හේතුව නම් RAM චිපයේ ක්‍රියාකාරීත්වය දෘඪ තැටියේ ක්‍රියාකාරීත්වයට වඩා අතිශයින් වේගවත් වීමය. කිසියම් පරිගණකයක්, යම් මොහෙතක විවෘත වී ඇති මෘදුකාංග වල දත්ත ලබා ගැනීමේ සහ දත්ත පිටකිරීමේ (I/O) අවශ්‍යතාව දෘඪ තැටියෙහි මතකයෙන් පමණක් සපුරාලීමට තැත්කලහොත් එම පරිගණකයේ කාර්යක්ෂමතාව සැලකිය යුතු ලෙස පහල බසිනු ඇත. මෙලෙස RAM මතකය පරිගණකයක පාවිච්චි වන්නේ මෘදුකාංග වලට අවශ්‍ය වේගවත්ව දත්ත සමඟ කටයුතු කිරීමේ අවශ්‍යතාව සපුරාලීම උදෙසාය.

RAM තුල ඇති සෑම bit එකකටම මතක ලිපිනයක් හෙවත් memory address එකක් ඇත. මෙම ලිපිනය මගින් RAM තුල සෑම bit එකක්ම පිහිටි ස්ථානය දැක්වේ. Memory address පිලිබඳව මෙහිදී කෙටි සඳහනක් පමණක් කරන්නේ ඔබට එය මෙම පොතෙහිදී පසුවට හමුවන නිසාය. එය වැඩි වශයෙන් භාවිතාවනු ඇත්තේ **“Pointers”** නැමැති පරිච්ඡේදයේදීය. Memory address එකක් මගින් අපට කිසියම් දත්තයක් RAM තුල සැබැවින්ම ඇති ස්ථානය සොයාගත හැක. එය හරියටම ඔබ කිසියම් නිවසක් පිහිටි ස්ථානය එහි ලිපිනයෙන් සොයාගැනීම වැනිය. නමුත් memory address එකක් යනු ඔබ ලියුම් කවර මත ලියනු ලබන විටද නම් සහිත ලිපිනයක් වැන්නක් නම් නොවනු ඇත. Memory address එකක් යනු තනිකරම සංඛ්‍යාත්මක අගයකි. එම සංඛ්‍යාවන් හැමවිටම දක්වනු ලබන්නේ 16 වැනි පාදයේ සංඛ්‍යා ලෙසිනි. 16 පාදයේ සංඛ්‍යා අප Hexadecimal numbers යනුවෙන් හඳුන්වමු.

නමුත් memory addresses මගින් bits ඇති ස්ථාන සෙවීමේ අරමුණ කුමක්ද? එහි අරමුණ නම් පරිගණක මතකයේ ඇති දත්ත වලට සෘජුව ඇතුල්වීමේ අවස්ථාව ලබාගැනීමය. මේ පිලිබඳ වැඩි විස්තර **“Pointers”** පරිච්ඡේදයේදී අපි සලකා බලමු.

Variables

පරිගණක වැඩසටහනක් නිර්මාණය කරණ විට, එය විසින් යම් යම් දත්ත මතකයේ තබා ගැනීමේ අවශ්‍යතාවන් මතුවේ. උදාහරණයක් ලෙස, භාෂා ශබ්දකෝෂයක් ලෙස සකසා ඇති මෘදුකාංගයක් ගැන සිතන්න. එයට ඔබ එක් භාෂාවකින් වචනයක් ඇතුල් කල විට එය එම වචනය අනෙක් භාෂාවට පරිවර්ථනය කර ඔබට ලබා දෙනු ඇත. මෙහිදී මෘදුකාංගය ඔබ ලබා දෙන වචනය තම මතකයේ තබාගෙන එයට සමාන වූ අනෙක් භාෂාවේ වචනය තීරණය කල යුතුය. "මතකයේ තබාගැනීම" ලෙස හැඳින්වෙන්නේ ඔබ ලබාදෙන වචනය, පරිගණක මතකයේ තාවකාලිකව ගබඩා කිරීමයි. මෙසේ ගබඩා කරන්නේ පරිගණකයේ RAM මතකයේය. නමුත් අප කිසියම් දත්තයක් RAM තුල ගබඩා කල විට, එම දත්තයත්, මෙතෙක් RAM හි ගබඩාවී ඇති අනෙක් බොහෝ දත්තයන් අතර තවත් එක් දත්තයක් පමණක් බවට පත්වේ. මෙසේ ගබඩා කෙරෙන දත්ත ඉන්පසුව අපට නැවත ප්‍රයෝජනය සඳහා ලබා ගැනීමට සිදුවේ. මෙම නැවත ලබාගැනීම සඳහා අප විසින් අප ගබඩා කල දත්තය නිවැරදිව නැවත හඳුනාගත යුතුය. නමුත් දත්ත සාගරය ඇති එක් කුඩා දත්තයක් පමණක් වෙන්කර හඳුනාගන්නේ කෙසේද? මෙහිදී අපගේ පිහිටට එන්නේ මෙම variable නැමැති ඒකකයි. Variable එකක් යනු අප විසින් RAM හි ගබඩා කරණ ලද කිසියම් දත්තයක් සඳහා ආරෝපණය කරණු ලබන අර්ථාන්විත නාමයකි. මේ මගින් අපට අවශ්‍ය දත්තය, RAM තුල අන්තර්ගත අනෙක් දත්ත අතරින් වෙන්කර හඳුනා ගත හැකිය. එය හරියටම පුද්ගලයන් රාශියක් අතරින්, ඔබට අවශ්‍ය පුද්ගලයෙකු ඔහුගේ නමින් හඳුනාගැනීම වැනිය.

පරිගණක වැඩසටහනක් තුල variable එකක් සෑදීමේදී අප විසින් කරුණු 2ක් සම්පූර්ණ කල යුතුය:

1. පලමුවෙන්ම අප සාදන variable එකෙහි වර්ගය සහ නාමය compiler මෘදුකාංගයට හඳුන්වා දිය යුතුය. මෙය variable declaration ලෙස හැඳින්වේ. මෙහි අවශ්‍යතාව නම්, සැබෑ ලෙසම variable එක නිමවීමට පෙර එයට අවශ්‍ය වන ඉඩ RAM තුලින් වෙන්කර ගැනීමට අවශ්‍ය වීමයි.

2. ඉන්පසු ඔබට variable එකට අගයක් ලබාදී එහි පැවැත්ම සැබෑ ලෙසම තහවුරු කල හැකිය. මෙය variable definition ලෙස හැඳින්වේ. අප එම ලබා දෙන අගය නම්, අපට RAM තුල ගබඩා කිරීමට අවශ්‍ය දත්තය වේ.

මේ පිලිබඳ වැඩි විස්තර මීලඟ එන, දත්ත වර්ගයන් පිලිබඳ පරිච්ඡේදයේදී සලකා බලමු.

Identifiers

Identifiers යනු, පරිගණනක වැඩසටහනක් තුලදී ඔබට අවශ්‍ය දත්ත හඳුනාගැනීම සඳහා ඒවා නම් කරනවිට, ඔබ එම දත්ත වෙත දෙනු ලබන නාමයන්ය. ඉහතදී ඔබ හඳුනාගත් variables, පරිගණක වැඩසටහනක් තුලදී සාදන විට, එයට සපයනු ලබන නාමය identifier එකක් සඳහා උදාහරණයකි. නමුත් identifiers ලෙස හඳුන්වන්නේ variables පමණක් නොවේ.

Expressions

පරිගණක වැඩසටහනකදී නොයෙක් ගණනයන් කිරීමට සිදුවීම සාමාන්‍ය දෙයකි. මෙවැනි අවස්ථාවන් වලදී expressions යොදාගැනීමට අවශ්‍යවේ. උදාහරණයක් ලෙස පහත දැක්වෙන්නේ identifiers 3 ක් භාවිතාකර ලියනු ලැබූ expression එකකි.

Salary = Basic + Overtime

ඉහත expression එකෙහි ප්‍රධාන දෑ 2 ක් දැකිය හැකිය:

1. Operands.
ඉහත expression එකෙහි දැකිය හැකි operands වන්නේ Salary, Basic සහ Overtime යන identifiers ය.

2. Operators.
ඉහත expression එකෙහි දැකිය හැකි operators වන්නේ " = " හා " + " යන ගණිතමය සලකුණු යුගලය. Operators වල කාර්යය නම්, expression එකකදී, operands භාවිතා කරමින් ගණිත කර්මයන් සිදු කිරීමය. ඉහතදී, + operator එක මගින් Basic සහ Overtime යන operands යුගලයෙහි අගයන් එකතු කරනු ලබන අතර, = operator එක මගින් එම එකතුවෙන් ලැබෙන පිලිතුර Salary නම් operand එකට ආදේශ කරනු ලබයි. නමුත් expressions යනු ගණනය කිරීම් පමණක් නොවේ. එය භාවිතාවන තවත් අවස්ථා ඇත. Expressions සහ operators පිලිබඳව ඔබට ''Operators'' පරිච්ඡේදයේදී වැඩිවිස්තර හමුවෙනු ඇත.

Keywords

Keywords යනු පරිගණක වැඩසටහනක් තුලදී විශේෂ අර්ථයන් සහිත නම් හෝ වචනයි. මේවා C++ භාශාව මගින් විශේෂ කාර්යයන් සඳහා වෙන්කර ඇති වචනයි. Keywords ඔබට identifiers ලෙස භාවිතා කිරීමට නොහැකියි. පහත දැක්වෙන්නේ C++ හි අඩංගුවන keywords ය. වගුවේ දැක්වෙන ඇතැම් keywords පිලිබඳව ඔබට ඉදිරියේදී කියවීමට ලැබෙනු ඇත.

asm	**auto**	**bad_cast**	**bad_typeid**
bool	**break**	**case**	**catch**
char	**class**	**const**	**const_cast**
continue	**default**	**delete**	**do**
double	**dynamic_cast**	**else**	**enum**
except	**explicit**	**extern**	**false**
finally	**float**	**for**	**friend**
goto	**if**	**inline**	**int**
long	**mutable**	**namespace**	**new**
operator	**private**	**protected**	**public**
register	**reinterpret_cast**	**return**	**short**
signed	**sizeof**	**static**	**static_cast**
struct	**switch**	**template**	**this**
throw	**true**	**try**	**type_info**
typedef	**typeid**	**typename**	**union**
unsigned	**using**	**virtual**	**void**
volatile	**while**		

2. දත්ත වර්ගයන් හෙවත් Data types

වැඩසටහනක දත්ත හෙවත් Data ගබඩා කරනු ලබන්නේ variables තුල බව ඔබ පසුගිය පරිච්ඡේදයේදී දුටුවා. නමුත් කිසියම් දත්තයක් ගබඩා කිරීමට ඕනෑම variable එකක් පාවිච්චි කල නොහැකිය. උදාහරණයක් ලෙස කිසියම් කෙනෙකුගේ නමක් වැනි දත්තයක් ගබඩා කිරීම සඳහා සහ ඔහුගේ උපන් දිනය ගබඩා කිරීම සඳහා වෙනස් වර්ගයන් දෙකක variables දෙකක් පාවිච්චි කල යුතුය. මෙයට හේතුව වන්නේ නම හා වයස යන දත්ත යුගල එකිනෙකට වෙනස් දත්ත වර්ග දෙකකට අයත් වීමය. මෙවැනි දත්ත වර්ග රුසක් C++ තුල අඩංගුය. මෙතැන් සිට දැනට ඔබේ අවශ්‍යතාවයට සරිලන පරිදි, මෙම දත්ත වර්ගයන්ගෙන් කීපයක් පමණක් පැහැදිලි කර ගැනීමට උත්සාහ ගනිමු.

1. int (integer)

int ලෙස හැඳින්වෙන්නේ සංඛ්‍යාත්මක දත්තයන්ය. කිසියම් සංඛ්‍යාත්මක අගයක් variable එකක් තුල ගබඩා කිරීමට නම් අප int වර්ගයේ variable එකක් සෑදිය යුතුයි. මෙලෙස variable එකක් සාදන ආකාරය පැහැදිලි කරගැනීමට අප variable එකක් නිමවා බලමු. year නැමැති නමින් int වර්ගයේ variable එකක් පහතින් සාදා ඇති ආකාරය බලන්න:

```
==========================================

    int year;

==========================================
```

ඉහත, `variable` එකෙහි නාමයට`(year)`ඉදිරියෙන් ඇති `int` නැමැති කොටසින් දැක්වෙන්නේ අප සාදන variable එක int වර්ගයට අයත් බවයි. මෙහිදී අප කරන්නේ year නැමැති variable සඳහා පරිගණකයේ RAM මතකයෙන් කොටසක් වෙන් කිරීම පමණයි. මෙලෙස variable එකක් සෑදීමේදී අප පලමුවෙන් එම variable එකට අදාල කොටස හෙවත් බයිට සංඛ්‍යාව RAM මතකයෙන් වෙන්කර ගත යුතුයි. මෙම ක්‍රියාව variable declaration ලෙස හැඳින්වේ. එය හරියටම කිසියම් ගොඩනැගිල්ලක් ඉදි කිරීමට පෙර, එය ඉදිකිරීම සඳහා සෑහෙන භඬම් ප්‍රමාණයක් වෙන් කරගැනීම හා සමානය. RAM මතකයෙන් වෙන්කරන මෙම බයිට හෙවත් මතක ප්‍රමාණය, සාදන variable එකෙහි දත්ත වර්ගය අනුව වෙනස් වේ - ඔබ තැනීමට නියමිත ගොඩනැගිල්ලේ විශාලත්වය අනුව, භඬමියෙහි විශාලත්වය තීරණය කල යුතුය. මෙලෙස int variable එකක් නිපදවීමේදී RAM මතකයෙන් වෙන් වන ඉඩ වන්නේ බයිට 4 කි.

දැන් අප මෙම year නැමැති variable එක සඳහා අවශ්‍ය මතකය වෙන්කර ගත් නිසා, අපට දැන් එය තුල සංඛ්‍යාත්මක අගයක් තැන්පත් කල හැක - ඔබට අවශ්‍ය භඬම් ප්‍රමාණය වෙන්කර ගත් පසු, ගොඩනැගිල්ල ඉදිකල හැකිය. මෙය variable definition ලෙස හැඳින්වේ. එනම් මෙහිදී සිදුවන්නේ variable එකට අගයක් ආදේශ කර එහි පැවැත්ම RAM තුල ස්ථිර කිරීමයි. එය සිදුකරන්නේ පහත අයුරිනි:

```
==========================================

    year = 2010;

==========================================
```

දැන් year නැමැති variable එකෙහි අගය 2010 වේ. වෙනත් වචන වලින් කියතහොත් year නැමැති variable එක RAM මතකය තුල වෙන්කරගෙන ඇති ඉඩ ප්‍රමාණයෙහි, එනම් බයිට 4 ක් තුල, 2010 නැමැති සංඛ්‍යාව

තැන්පත් වී ඇත. අප මෙහිදී variable declaration සහ variable definition යන ක්‍රියා දෙක අවස්ථා දෙකකදී සිදුකලත්, පහසුව පිණිස මේ දෙකම එකවිට පහත පරිදි සිදුකල හැක:

```
================================================

      int year = 2010;

================================================
```

අප දැන් int variable එකක් යොදාගෙන සරල පරිගණක වැඩසටහනක් ලියා බලමු:

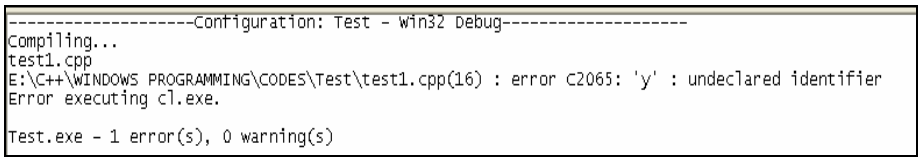
```
=======================================================

      // ints.cpp

      #include<iostream.h>

      void main()
      {
        int year = 2010;
        cout << “Days in a year = “ << year << endl;
      }

=======================================================
```

ඉහත වැඩසටහන Visual C++ මගින් පෙර පරිදිම Compile කර Run කරන්න. Terminal window එකෙහි “Days in a year = 2010” යන්න print වෙනු ඇත. මෙහිදී සිදුවූයේ කුමක්ද? පලමුව අප 2010 අගය year නැමැති variable තුල ගබඩා කලෙමු. ඉන්පසුව cout << කේතය, year තුල ඇති අගය, එනම් 2010, Terminal window එක මතට output කලේය.

Scope

```
=====================================================

#include<iostream.h>

void main()
{
      int x = 10;

      {
            int y = 20;
            cout<< "Inside x:" << x << endl;
            cout<< "Outside y:" << y << endl;
      }

      cout<< "Outside x:" << x << endl;
      cout<< "Outside y:" << y << endl; // error
}

=====================================================
```

ඉහත වැඩසටහන Visual C++ තුල යතුරු ලියනය කොට compile කිරීමට උත්සාහ කරන්න - ඔබට එය compile කිරීමට නොහැකිවනු ඇත. සිදුවන එකම දෙය වනු ඇත්තේ Visual C++ හි output window එක තුලට පහත දැක්වෙන error message එක print වීමය. (ඔබට ලැබෙන error massage එකෙහි file path එක වෙනස්වනු ඇත)



(රූපය 2.1)

දැන් ඔබ output window එකෙහි මෙම error message එක මත double click කල යුතුය. (ඉහල සිට ඇති 4 වැනි පේලිය මත double click කරන්න) එවිට editor window එකෙහි මෙම error එක නිපදවීමට පාදකවුණු කේතය highlight වනු ඇත. එම කේතය නම් 4 වැනි cout විධානය අඩංගු කේතය බව ඔබට highlight වී පෙනෙනු ඇත. Error එකින් කියවෙන්නේ y යනු declare නොකරන ලද ආගන්තුක variable එකක් බවයි. මෙසේවන්නේ මක්නිසාද? ඔබ y declare කර එහි අගය 20 බවට define ද කර ඇතුවා නොවේද? නමුත් ඔබ නැවත වරක් වැඩසටහන දෙස බැලුවහොත් පෙනෙන දෙයක් වන්නේ, y හි declaration එක ඇත්තේ සඟල වරහන් යුගලක් තුල බවයි. Error එක නිපදවූ cout අඩංගු කේතය ඇත්තේ මෙම වරහන් වලින් පිටතයි. මෙම කේතය මගින් y ව ඉල්ලා ඇත (හෙවත් y ව refer කර ඇත). නමුත් compiler මෘදුකාංගයට y නමින් identifier එකක් සොයා ගැනීමට නොහැකිවූ නිසා එය ඉහත දැක්වුණු error එක නිපදවූයේය. Compiler මෘදුකාංගයට y හඳුනාගැනීමට නොහැකිවූයේ y හි definition එක ඇත්තේ, ඇතුලත සඟල වරහන් යුගල තුල වීම නිසයි. මෙහිදී y හි පැවැත්ම මෙම වරහන් තුලට පමණක් සීමාවෙයි. වරහන් වලින් පිටත සිට එයට ඉහතදී මෙන් refer කරනු ලැබුවහොත් compiler මෘදුකාංගයට එය හඳුනාගැනීමට නොහැකිව යයි. වරහන් තුලදී නම් එය නිසැකවම y හඳුනාගනියි. සඟල වරහන් තුල ඇති දෙවැනි cout command එක error එකක් නිපදනොවීම හේතුවෙන් මේ බව තහවුරු වෙයි. දැන් error එක සහිත කේතය වැඩසටහනින් ඉවත් කොට හෝ එය comment කොට (එම පේලිය මුලට “ // “ සලකුණක් යොදා) නැවත වැඩසටහන compile කිරීමට උත්සාහ කරන්න. එවිට එය නියමාකාරව එය compile වන අයුරු ඔබ දකිනු ඇත.

මෙලෙස variable එකක් සඟල වරහන් යුගලක් තුල define කිරීමෙන් එහි පැවැත්ම එම සඟල වරහන් තුලට පමණක් සිමා කිරීම හැඳින්වෙන්නේ **“Scoping”** යනුවෙනි. වරහන් දෙක තුල ඇති ඉඩ එම variable එකෙහි **scope** එක ලෙස හැඳින්වෙයි. එම variable එක refer කලහැක්කේ මෙම scope එක තුලදී පමණි.

2. char (Character)

char – දත්ත වර්ගයෙන් යුතු variable එකක අනතර්ගත කල හැකි වන්නේ පරිගණක යතුරු පද්ධතිය තුල ඇති එක් අකුරක්, ඉලක්කමක් හෝ සලකුණක් හෙවත් **character** එකකි.

උදාහරණ - 'G’, ‘k’, ‘8’, ‘2’, ‘>’, ‘?’

ඉහත උදාහරණ වල ඇති “ ‘ “ සලකුණු හෙවත් single quotes දෙස බලන්න. අප char අගයක් හඳුනාගන්නේ මේ single quotes වලිනි. මෙහිදී ‘8’ සහ ‘2’ යනු ඉලක්කම් නොවේ. මෙසේ වන්නේ ඒවා single quotes වලින් ආවරණය කොට ඇති නිසාය. එවිට ඒවා ඉලක්කම් නොව. characters බවට පත් වේ. මෙලෙසම ‘>’, ‘?’ වැනි සලකුණුද මෙහිදී characters ලෙස සැලකේ.

දැන් අප char පාවිච්චි කර ලියූ වැඩසටහනක් දෙස අවධානය යොමු කරමු:

==========

```
// characters.cpp

#include <iostream.h>

void main()
{
  char x = ‘b’, y = ‘y’, s = ‘e’;

  cout<< x << y << s << endl;
```

```
}
```

==========

ඉහත වැඩසටහන compile කර run කරන්න. “bye” යන වචනය console window එක තුල print වෙනු ඇත. “bye” යන වචනය සැදුනේ ‘b’, ‘y’, ‘e’ යන characters ත්‍රිත්වය එක පෙලට වෙනවෙනම print වීමෙන් බව ඔබට පැහැදිලි වනු ඇත.

ඉහත variables, define කර ඇති ආකාරය දෙස බලන්න. එම variables තුනම define කර ඇත්තේ එක්වරම. එක් පෙලකට බව ඔබට පෙනේ. variables වෙන් කර ඇත්තේ කොමා වලිනි. මෙය ඔබ පෙරදී දුටු variables define කිරීමට සමානයි. එනම්, ඉහත define කිරීම ම පහත ආකාරයෙන් ද සිදුකල හැක:

```
==============================
      char x = ‘b’;
      char y = ‘y’;
      char s = ‘e’;
==============================
```

3. bool (Boolean)

කිසිවෙකු දැන් ඔබෙන් පහත ප්‍රශ්ණය අසයි:

“දැන් ඔබේ කාමරයේ වඳුලි බල්බය ඇත්තේ දැල්වඳ?”

ඔබේ පිලිතුර විය යුත්තේ “ඔවු” හෝ “නැහැ” යන දෙකින් එකකි. එනම්, “දැන් ඔබේ කාමරයේ ඇති වඳුලි බල්බය දැල්වී ඇත” යන ප්‍රකාශනය සත්‍ය හෝ අසත්‍ය විය හැකිය. වෙනත් ආකාරයකින් කියතහොත් එම ප්‍රකාශනය සත්‍ය සහ අසත්‍ය යන අවස්ථා දෙකින් යුක්තය. පරිගණක වැඩසටහනක මෙසේ අවස්ථා දෙකකින් පමණක් සමන්විත දේ දැක්වීමට මෙම **bool** නැමැති data type එක භාවිතාවේ. bool – data type එකින් යුතු variable එකකට තිබිය හැක්කේ අගයන් දෙකකි. එනම්, සත්‍ය අවස්ථාව සහ අසත්‍ය අවස්ථාව. පරිගණක වැඩසටහන් ලිවීමේදී මෙම සත්‍ය අවස්ථාව **true** යන keyword එකින් හෝ 1 හෝ 0 ට වැඩි ඕනෑම සංඛ්‍යාවකින් දක්වන අතර අසත්‍ය අවස්ථාව දක්වන්නේ **false** යන keyword එකින් හෝ 0 අගය මගිනි. පරිගණක වැඩසටහනකදී **bool** data type එක පාවිච්චි කෙරෙන අවස්ථාවක් වන්නේ, වැඩසටහන තුල කිසියම් තත්වයක් සැපිරී ඇති හෝ නැති බව තීරණය කිරීමටය. මෙය වටහා ගැනීම සඳහා අප ඉහත වඳුලි බල්බ උදාහරණය පරිගණක වැඩසටහනකට යොදා බලමු:

==========

```
// bool.cpp

#include <iostream.h>

void main()
{
  bool on = true;
  if(on){
      cout<< “The bulb is on” << endl;
  }
  else{
      cout<< “the bulb is off” << endl;
  }
}
```

==========

ඉහත වැඩසටහනෙහි ඔබට අලුත් දෙයක් වන **if-else** statement එක හමුවේ. **if-else** statement එක ගැන අප මඳක් පසුව කථා කරමු. දැනට ඔබ මේ වැඩසටහනෙහි අවධානය කල යුත්තේ bool – variable එකක් වන on වෙත සහ එයට ආදේශ කර ඇති අගය වන true අගය වෙතය. වැඩසටහනෙහි පලමුව සිදුවන්නේ on නැමැති bool දත්ත වර්ගයට අයත් variable එකක් define වී එයට true අගය ආදේශ වීමය. ඉන්පසු, ඔබ තවම හඳුනාගෙන නැති if-else statement එක ක්‍රියාත්මක වීමෙන් on හි අගය කුමක්දැයි පරීක්ෂාවට ලක්වේ. එහි

අගය **true** වුවහොත්, “The bulb is on” යන වචන පෙල print වන අතර අගය **false** වුවහොත්, “The bulb is off” යන්න print වේ. වැඩසටහන compile කර run කර බලන්න. on හි අගය true නිසා “The bulb is on” යන්න print වෙනු ඇත. දැන් on වෙත false අගය ආදේශ කර (true වෙනුවට on ඉදිරියේ false යන්න type කරන්න) නැවත වැඩසටහන run කර බලන්න. එවිට print වනු ඇත්තේ “The bulb is off” යන්නයි. ඒ අනුව මෙම වැඩසටහනින් දැක්වෙන්නේ bool – variable එකක් පරිගණක වැඩසටහනකදී ප්‍රයෝජනයට ගත හැකි අවස්ථාවක්ය. on හි අගය true හෝ false වීම මත වැඩසටහනෙහි ක්‍රියාකාරීත්වය දෙආකාරයක් ගත්තේය. වෙනත් වචන වලින් කිවහොත් අපට වැඩසටහනක් ක්‍රියාත්මක විය යුතු මාර්ගය කුමක්දැයි එය ධාවනය වන මොහොතේදී මෙවැනි ආකාරයන්ගෙන් තීරණය කල හැකිය.

4. float

අප int – data type ගැන කථා කිරීමේදී සලකා බැලුවේ පූර්ණ සංඛ්‍යා පිළිබඳව පමණක් බව ඔබ දුටුවාද? int – variables තුල ගබඩා කල හැක්කේ පූර්ණ සංඛ්‍යාවන් පමණි. උදාහරණයක් ලෙස 10.5 වැනි දශම සංඛ්‍යා එහිදී භාවිතා කල නොහැක. float දත්ත වර්ගය ඇත්තේ මෙම අවශ්‍යතාව සපුරාලීම සඳහාය. පහත දැක්වෙන්නේ float භාවිතා කල වැඩසටහනකි:

```
=====================================================

      // floats.cpp

      #include<iostream.h>

      void main()
      {
        float time = 10.35;
        cout << “The time is: “ << time << endl;
      }

=====================================================
```

5. string

මෙම දත්ත වර්ගය භාවිතා කරන්නේ වචන හෝ වාක්‍ය ගණයේ දත්ත සමග කටයුතු කිරීමටයි. අප ඉහත සාකච්ඡා කරන ලද සියලුම දත්ත වර්ගයන් C++ තුල ස්වභාවිකම අඩංගුවන දත්ත වර්ගයන්ය. නමුත් මෙම string දත්ත වර්ගය C++ තුලට කෘත්‍රිමව එක්කල දෙයකැයි කිවහැකිය. එමෙන්ම strings පිලිබඳව සලකා බැලීමට අනෙත් data types වලට වඩා වැඩි කරුණු ගණනාවක් ඇත. මේනිසා මෙම විස්තර ගැන මෙතැනදී කථා නොකර ඒ සඳහා වෙනම කොටසක් මෙම පොතෙහි “C++ Libraries” නැමැති පරිච්ඡේදයේදී වෙන් කර ඇත. නමුත් දැනට string – data type එක යනු කුමක්දැයි මූලික ලෙස පහත වැඩසටහනින් පැහැදිලි කරගන්න:

```
=====================================================

// strings.cpp

#include<iostream.h>
#include<string>
using namespace std;

void main()
{
  string message = "Hello there! I'm learning C++.";
}

=====================================================
```

ඉහත වැඩසටහනෙහි message නැමැති string – variable එකක් define කර ඇති අතර එයට කෙටි වාක්‍යයක් ආදේශ කර ඇත.

3. Operators

මෙතෙක් ඔබ සලකා බැලූ පරිගණක වැඩසටහන් වල ඔබ පාවිච්චි කල variables දෙස බලන්න. ඔබ ඒවා define කර ඒවා සඳහා අගයන් ආදේශ කිරීමෙන් අනතුරුව ඒවායේ අගයන් නොවෙනස්ව පැවතුනා නේද? නමුත් සැබෑ පරිගණක වැඩසටහනකදී ඔබ පාවිච්චි කරන දත්ත වෙනස්කම් වලට ලක් කිරීමට සිදුවීම නිතර සිදුවන කරුණකි. උදහරණයක් ලෙස ඔබ ඔබට අවශ්‍ය ඊ-මේල් ලිපිනයන් ගබඩා කර තබා ගැනීම සඳහා මෘදුකාංගයක් නිමවීමේ යෙදෙන බව සිතන්න. එහි ගබඩා කර ඇති ලිපිනයන් සංඛ්‍යාව වාර්ථාකර තැබීමට int – variable එකක් ඔබ පාවිච්චි කරතැයි ද සිතන්න. මෘදුකාංගයට අඩංගු කරණ ලද ලිපිනයන් සංඛ්‍යාව වැඩිවත්ම ඔබ මෙම variable එකෙහි අගයද එම ලිපිනයන් සංඛ්‍යාව අනුව ඉහල නැංවිය යුතුයි. මේ ආකාරයට වැඩසටහනක දත්තයන් වෙනස් කිරීමට සිදුවන විට අප operators ප්‍රයෝජනයට ගනිමු.

Operators භාවිතයෙන් දත්ත එකතු කිරීම, අඩු කිරීම, ගුණ කිරීම සහ බෙදීම නැතහොත් කිසියම් දත්තයන් යුගලක් හෝ කීපයක් සමාන අගයක් ගනඳැයි පරීක්ෂා කිරීම වැනි නොයෙක් ක්‍රියාවන් සිදුකිරීමට හැකිය. Operators මගින් මෙම ක්‍රියා සිදුකිරීමට නම් අප operators අන්තර්ගත කර expressions ලිවිය යුතුය. Expression එකක් යනු කුමක්දැයි ඉහතදී ඉතා කෙටියෙන් ඔබ මීට ඉහතදී හඳුනාගත්තා. Operators අඩංගු කරණ ලද expression එකක් කවරාකාරදැයි බැලීමට ප්‍රථමයෙන් අප operators යනු මොනවාදැයි උගත යුතුය.

C++ තුල operators රැසක් හමුවේ. පහත දැක්වෙන සටහනින් එම operators වැඩි හරියක් දැක්වේ.

වර්ගය	නම	සංඛේතය
Arithmatic operators	Multiplication	*
	Division	/
	Modulus	%
	Addition	+
	Subtraction	-
Logical operators	Logical AND	&&
	Logical OR	||
	Unray NOT	!
Relational operators	Greater than	>
	Less than	<
	Greater than or Equal to	>=
	Less than or Equal to	<=
	Equal to	==
	Not equal to	!=
Increment & Decrement Operators	Increment	++
	Decrement	--
Assignment Operators	Assignment	=
	Addition	+=

	Assignment	
	Subtraction Assignment	-=
	Multiplication Assignment	*=
	Division Assignment	/=

ඔබට පෙනෙන පරිදි operators සැලකිය යුතු තරම් විශාල සංඛ්‍යාවක් ඇතත්, ඇත්තෙන්ම මේවා පිලිබඳව ඉගෙනීම එතරම් අපහසු නැත. මෙතැන් සිට මෙම operators ගැන විමසා බලමු.

Arithmatic operators

Arithmatic ගණයේ Operators මගින් සිදුවන්නේ එකතු කිරීම, ගුණ කිරීම වැනි සාමාන්‍ය ගණිත කර්මයන්ය. පහත දැක්වෙන්නේ Arithmatic operators පාවිච්චි කර ලියනු ලැබූ සරල වැඩසටහනකි. Expression එකක් යනු කුමක්දැයි යන කරුණද ඔබ මෙම වැඩසටහනින් දකිනු ඇත:

```
============================================================
```

```
// ArithmaticOps.cpp

#include<iostream.h>

void main()
{
  int toShop = 500, backHome = 500, distance;
  distance = toShop + backHome;
  cout << "The distance I walked: " << distance << "
  meters"    << endl;

  int time = 50, shoppingTime = 30, walkingTime,
  speed;
  walkingTime = time - shoppingTime;
  speed = (toShop + backHome) / walkingTime;
  cout << "My average speed was: " << speed << "
  meters per minute" << endl;

  int weekShoppingTimes = 3, allTime;
  allTime = time * weekShoppingTimes;
  cout << "Time I spent on shopping for a week: " <<
  allTime    << " minutes" << endl;

}
```

```
============================================================
```

(ඉහත වැඩසටහනෙහිදී හා ඉදිරියට හමුවීමට නියමිත වැඩසටහන් වලදී ඇතැම් පරිගණක කේතයන් එක පෙලක් ලෙස දැක්වීම සඳහා ඉඩ මදිවන නිසා එම කේතයන් පේලි දෙකක් වශයෙන් දක්වා ඇත. [මෙම බිඳුනු කේතයන් වක් ඊ සටහන් වලින් දැක්වේ] නමුත් ඔබ Visual C++ හි මෙම වැඩසටහන් යතුරු ලියනය කරන විට එම පේලි යුගලයන් එකම පේලි ලෙස type කරන්න. නොඒසේනම් බොහෝවිට ඔබේ වැඩසටහන් වල errors මතුවෙනු ඇත)

ඉහත වැඩසටහනෙන් පෙන්නුම් කරන්නේ කිසියම් පුද්ගලයෙකු කඩපොලට ගොස් බඩු මිලදී ගැනීමේදී ගත කල කාලය හා ඔහු ගමන් කල දුර ආදියයි. වැඩසටහනෙහි පලමුවෙන් සිදුකර ඇත්තේ පුද්ගලයාගේ නිවසේ සිට කඩපොලටත්, එහි සිට නිවසටත් ඇති දුර ද, මුලු දුර ද වාර්ථා කර තැබීම සඳහා පිලිවෙලින් `toShop`, `backHome` සහ `distance` යන int - variables ත්‍රිත්වය තැනීමයි. `toShop` සහ `backHome` වෙත 500 ආදේශ කර ඇත. 500 අගය මීටර් ලෙස සලකන්න. ඊලග පේලියෙන් ඔබ දකින්නේ addition operator (+) එක ද

toShop හා backHome variables (හෙවත් operands) දෙක ද භාවිතා කොට ලියනු ලැබූ expression එකකි. Expression යනු කුමක්දැයි මූලික වශයෙන් ඔබ 1වැනි පරිච්ඡේදයේදී හඳුනාගත්තා. මෙම expression එකෙහි කාර්යය වන්නේ toShop හා backHome යන variables දෙක එකතු කර මුලු දුර, distance variable තුල ගබඩා කිරීමයි.

ඉන්පසුව ඇති කේතයෙන් distance තුල ගබඩා වූ එම අගය console window එක මතට output වේ.

දෙවැනි කොටසේදී, කඩපොලට ගොස් පැමිණිමේදී ගතවූ මුලු කාලය වාර්ථා කර තැබීමට time ද බඩු මිලදී ගැනීමට ගතවූ කාලය දැක්වීමට shoppingTime ද කඩපොලට හා ආපසු නිවසට ඇවිදීමට ගතවූ කාලය දැක්වීමට walkingTime ද පුද්ගලයාගේ ඇවිදීමේ වේගය ගබඩා කිරීමට speed ද යන variables රැස සාදා ඇත. එහිදී time වෙත 50 ද, shoppingTime වෙත 30 ද ආදේශ කර ඇත. 50 හා 30 අගයන් විනාඩි ලෙස සලකන්න. දැන් සිදුවන්නේ කඩපොලට යාමට හා ඒමට ගතවූ මුලු කාලය ලබාගැනීමට, time හි අගයෙන් shoppingTime හි අගය subtraction operator (-) භාවිතා කර අඩු කිරීමය. එය walkingTime තුල ගබඩා කර ඇත. ඉන්පසු පුද්ගලයාගේ සාමාන්‍යය ඇවිදීමේ වේගය සොයා එම අගය speed තුල ගබඩා කර ඇත. එය සිදුකර ඇත්තේ division operator (/) භාවිතා කර, කඩපොලට හා ආපසු නිවසට ඇති දුර, පුද්ගලයාගේ ඇවිදීමේ කාලයෙන් බෙදීම සඳහා expression එකක් ලිවීමෙනි.

තෙවැනි කොටසින් weekShoppingTimes සහ allTime යන variables යුගල define කර ඇත්තේ පිලිවෙලින් සතියකට කඩපොලට ගිය වාර ගණන සහ ඒ සඳහා සතියකදී ගතකල මුලු කාලය වාර්ථා කිරීමටය. අවසානයේදී multiplication operator (*) භාවිතා කර එක් වරක් කඩපොලට යෑම සඳහා ගත වූ කාලය සතියකට ගිය වාර ගණින් ගුණකර මුලු කාලය ලබා ගෙන ඇත.

මෙම වැඩසටහනේදී +, -, /, * යන operators හතරම භාවිතා වූ අතර එහි කිවයුතු දෙයකට ඇත්තේ multiplication operator (*) ගැන පමණයි. එනම්, ඔබ මෙතෙක් ගුණ කිරීම සඳහා පාවිච්චි කල x ලකුණ මෙහිදී Asterix - * ලකුණ බවට පත්වී ඇත. තවද, වැඩසටහනෙහි 2 වැනි කොටසේ සුලු කිරීමේදී වරහන් පාවිච්චි කර ඇති ආකාරය බලන්න. ඔබ දැනටමත් බොහෝ විට දැන සිටිය හැකි පරිදි අප සුලු කිරීමකදී වරහන් යොදන්නේ එම වරහන් තුල ඇති කොටස වෙනම ඒකකයක් ලෙස ප්‍රථමයෙන් සුලු කරගැනීමට අවශ්‍ය වූ විටය. එනම් මෙහිදී ප්‍රථමයෙන් toShop හා backHome හි අගයන් එකතු වී, ඉන්පසුවයි එම අගය walkingTime අගයෙන් බෙදෙන්නේ. මෙවැනි අවස්ථා වලදී වරහන් භාවිතා කිරීම යහපත් පුරුද්දකි. එමගින් ඔබට අවශ්‍ය සුලුවීම නිවැරදි ලෙස සිදුවන බවට ඔබට තහවුරු කර ගත හැක.

දැන් ඉතිරිව ඇත්තේ Modulus operator (%) ගැන සාකච්ඡා කිරීමටයි. පහත වැඩසටහන පරීක්ෂා කර බලන්න:

```
=======================================================
```

```
// Modular.cpp

#include<iostream.h>

void main()
{
  int aNumber = 27;
  int mod = aNumber % 2;
  cout << mod << endl;

  int anotherNumber = 65;
  mod = anotherNumber % 5;
  cout << mod << endl;
}
```

```
=======================================================
```

ඉහත වැඩසටහන ක්‍රියාත්මක කර බලන්න. පලවෙනි හා දෙවැනි අවස්ථා වලදී mod හි අගය ලෙස පිලිවෙලින් 1 හා 0 console window එක තුල print වේ. පලමු අවස්ථාවේදී mod හි අගය 1 වූයේ මෙසේයි: modular operator (%) මගින් සිදුවන්නේ කිසියම් සංඛ්‍යාවක් බේදීමෙන් පසු ලැබෙන ඉතිරිය ගණනය කිරීමයි. මේ අනුව aNumber යන variable එකෙහි අගය වන 27, 2 න් බෙදා ලැබෙන ඉතිරිය mod තුල තැන්පත් කර ඇත. 27, 2 න් බෙදූ විට 13 ද ඉතිරිය 1 ද වේ. මේනිසා පලමු අවස්ථාවේදී mod = 1 වේ. 2 වන අවස්ථාවේදීත් සිදුන්නේ මෙයමයි. එයද පරීක්ෂා කර බලන්න.

Logical operators

අප Logical Operators භාවිතා කරන්නේ, එහි නමින් ම කියවෙන පරිදි, තර්කානුකූල ක්‍රියාවන් සිදුකිරීම සඳහාය. මෙම Logical operators මගින් සිදුවන්නේ අවස්ථාවන් කිහිපයක් ගෙන එම අවස්ථාවන් අතර තර්කානුකූල සම්බන්ධයක් ගොඩනැගීමය. මේ සම්බන්ධය ඇති වන්නේ එම අවස්ථාවන්ගේ සත්‍ය-අසත්‍ය භාවයන් අතරයි. මෙය වටහා ගැනීමට අප උදාහරණයක් ගෙන බලමු.

දැන් ඔබ, ඔබ ඉදිරියේ සිටිනා A, B හා C නැමැති ලමුන් තිදෙනෙකු සිතින් මවා ගන්න. A ගේ හිසකේ දුඹුරු පැහැ බව ඔබට පෙනෙන අතර ඇස් නිල් පැහැයෙන් යුතුය. B ගේ හිසකේ කලු පැහැ ගන්නා අතර ඇගේ ඇස් දුඹුරු පැහැතිය. C රතු පැහැ හිසකෙසින් යුතු වන අතර දැස් කාල වර්ණය. මේ මොහොතේ කිසිවෙකු පැමිණ ඔබට මෙසේ කියයි: **“A ගේ හිසකෙස් දුඹුරු පැහැයෙන් යුතුයි. ඇස් නිල් පාටයි. B ට ඇත්තේ දුඹුරු පැහැති හිසකෙස්ය. නමුත් ඇස් කලු පාටයි. C ගේ හිසකෙස් රතු පාටයි. ඈට ඇත්තේ කලු පැහැ දෙනෙත්ය.”** මෙම පුද්ගලයා ගේ කියමන සලකා බලන්න. එය සම්පූර්ණ ලෙස සත්‍ය නොවන බව ඔබට පෙනේවි. ඔහු පලමුවෙනි හා අවසන් අවස්ථා වලදී සත්‍ය ප්‍රකාශ කරන නමුත් දෙවැනි අවස්ථාවේදී අසත්‍ය ප්‍රකාශ කරයි. කෙසේ වුවද ඔහු වැඩි කොටසක් කියා ඇත්තේ සත්‍යයයි. එසේනම්, ඔබ තීරණය කරන්නේ ඔහු සත්‍ය ප්‍රකාශ කල බවද අසත්‍ය ප්‍රකාශ කල බවද? ඔහු තිදෙනා පිලිබඳවම සත්‍ය කියා ඇත්නම් ඔබට ඔහු සත්‍ය පවසා ඇතැයි නිගමනය කල හැක. ඔහු තිදෙනා පිලිබඳවම අසත්‍ය දේම කියා ඇතනම් ඔබට ඔහු පවසා ඇත්තේ අසත්‍ය බව තීරණය කල හැක. නමුත් මේ අවස්ථාවේදී මේ දෙකින් එකක් වත් ඔබට නිගමනය කල නොහැකිය. පරිගණක වැඩසටහන් වලදීද මෙවැනි අවස්ථා මතුවේ. එනම් ඔබට අවස්ථා කීපයක් සලකා බලා ඒවායේ සත්‍ය-අසත්‍ය තාව අනුව යම් යම් තීරණ වලට එළඹීමට සිදුවේ. ඇතැම් අවස්ථා වලදී සියල්ල සත්‍ය වේ. තවත් අවස්ථාවකදී සියල්ල අසත්‍ය වේ. නමුත් ඇතැම් අවස්ථා වලදී සමහරක් දේ පමණක් සත්‍ය වේ. මෙවැනි අවස්ථාවන් හඳුනා ගැනීමේදීයි අපට Logical Operators වැදගත් වන්නේ.

මින් ඉහතදී අප සලකා බැලූ Arithmatic Operators සැමවිටම පිලිතුර ලෙස ඉතිරි කලේ ධ්වඳ සංඛ්‍යාත්මක අගයන්ය. නමුත් Logical Operators පිලිතුර ලෙස ලබා දෙන්නේ 1 හෝ 0 යන අගයන් යුගල පමණය. මෙහිදී 1 අගය true (සත්‍ය) නැමැති අගයට අනුරූප වන අතර 0 අගය false (අසත්‍ය) නැමැති අගයට අනුරූප වේ. මෙම true සහ false අගයන් Boolean data type එකට ආවේණික වූ අගයන් බව ඔබ data types පරිච්ඡේදයේදී හඳුනාගත්තා මතක ඇති.

අප දැන් Logical Operators භාවිතා කර ඉහත A, B, සහ C තිදෙනා පිලිබඳ උදාහරණය ධ්වඳ අයුරින් පරිගණක වැඩසටහනකට ආදේශ කර ඇති ආකාරය බලමු.

Logical AND (&&)

Logical AND හි කාර්යය නම් අවස්ථාවන් කීපයක් ගෙන එම අවස්ථා **සියල්ලම සත්‍ය නම් පමණක්** true අගය නිපදවීමය. එම එක් අවස්ථාවක් වුවද අසත්‍ය වේ නම් Logical AND නිපදවන්නේ false අගයයි. මෙය සිදුවන්නේ පහත ආකාරයෙනි:

```
=====================================================

// LogicAnd.cpp

#include<iostream.h>
 #include<string>
using namespace std;

void main()
{

  string aHair = “Brown”, aEyes = “Blue”, bHair = “Black”,
    bEyes = “Brown”, cHair = “Red”, cEyes = “Black”;
```

```
  // (1)
  bool hairCheck = (aHair == “Brown”);
  cout << hairCheck << endl;

  // (2)
  hairCheck = (aHair == “Brown”) && (bHair == “Red”);
  cout << hairCheck << endl;

  // (3)
  bool eyesCheck = (aEyes == “Blue”) && (bEyes == “Brown”)
      && (cEyes == “Black”);
  cout << eyesCheck << endl;

}

=====================================================
```

ඉහත වැඩසටහනෙහි දැනට ඔබ නොදන්නා **Equal to (==) operator** එක, expressions සඳහා පාවිච්චි කර තිබේ. Equal to (==) operator ගැන දැනට මේ ආකාරයෙන් සිතන්න: Equal to (==) operator එක ඔබ දැනටමත් දන්නා Assignment operator (=) එකට වඩා වෙනස් කාර්යයක් ඉටු කරයි. = මගින් සිදුවන්නේ කිසියම් operands යුගලක් **සමාන කිරීම** වූ අතර, == මගින් සිදුවන්නේ කිසියම් operands යුගලක් **සමාන දැයි පරීක්ෂා කිරීම**යි. එම operands යුගලෙහි අගයන් සමාන නම් 1 හෙවත් true අගයද, අසමාන නම් 0 හෙවත් false අගයද == මගින් නිපදවයි. එසේනම් ඉහත වැඩසටහනෙහි 1 වැනි කොටසෙහි ඇති `aHair == “Brown”` යන expression එකින් සිදුවන්නේ aHair නැමැති operand එකෙහි අගය වන්නේ “Brown” දැයි පරීක්ෂා කිරීමයි. aHair වෙත “Brown” යන අගය ඉහතින් assign කර ඇති නිසා මෙහිදී == මගින් true හෙවත් 1 නිපදවෙයි. නිපදවෙන මෙම අගය hairCheck නැමැති bool type variable එක තුල ගබඩා කර ඇති අතර, වැඩසටහන ක්‍රියාත්මක කිරීමේදී hairCheck හි අගය ලෙස 1 print වෙනු ඇත.

දැන් 2 වන කොටස දෙස බලන්න. එහිදී Logical AND - && පාවිච්චි කර ඇත. ඔබ මොහොතකට පෙර දුටු පරිදි `aHair == “Brown”` යන්න true අගය නිපදවයි. `(bHair == “Red”)` යන expression නිපදවන්නේ false අගයයි. මක්නිසාදයත් `bHair` හි අගය “Black” වන ලෙස එය define කර ඇති නිසාය. දැන් මෙම expressions යුගල && මගින් සම්බන්ධ කර ඇත. මෙවිට සිදුවන්නේ කුමක්ද? සිදුවන්නේ එම expressions මගින් නිපදවුනු true සහ false අගයන් && මගින් පරීක්ෂාවට ලක් වීමය. ඉහත සඳහන්වූ පරිදි && මගින්, එය පරීක්ෂාවට ලක් කෙරෙන අවස්ථා සියල්ල true හෙවත් සත්‍ය අගයක් ගනී නම් true අගය නිපදවයි. මෙහිදී පරීක්ෂාවට ලක්වෙන පලවෙනි expression එක වන `(aHair == “Brown”)` හි අගය true වන අතර දෙවන expression එක වන `(bHair == “Red”)` හි අගය false වේ. එසේනම්, `(aHair == “Brown”) && (bHair == “Red”)` යන expression එක මගින් නිපදවෙන්නේ false අගයයි. false අගය `hairCheck` තුල ගබඩා කෙරෙන අතර false අගයට අනුරූප 0 අගය print වේ. මෙය හරියටම ඔබට ඉහත දී හමුවූ පුද්ගලයා මෙසේ කීම වැනිය: “A ට ඇත්තේ දුඹරු හිසකේය. B ගේ හිසකේ රතුය.“ ඔහු දෙවැනි අවස්ථාවේදී අසත්‍ය ප්‍රකාශ කරයි. මේ නිසා && මගින් false නිපදවේ.

දැන් තෙවැනි කොටසට. `(aEyes == “Blue”)` යන `expression` එක නිපදවන්නේ true අගයය. `(bEyes == “Brown”)` නිපදවන්නේ ද true අගයය. `(cEyes == “Black”)` ද true අගය නිපදවයි. සියල්ලෙහිම අගය true වෙ. මෙනිසා && මගින් නිපදවෙන්නේ ද true අගයයි. මෙනිසා true අගයට අනුරූප 1 අගය print වේ.

මීලඟට අප ඉගෙනීමට යන්නේ Logical OR (||) operator එක ගැනයි.

Logical OR (||)

Logical OR මගින් ඉටුවන්නේ අවස්ථා දෙකක් පරීක්ෂා කර, එම අවස්ථා දෙකින් **එකක් හෝ සත්‍ය නම්** true අගය නිපදවීමයි. පහත උදාහරණයෙන් මෙය විස්තර වේ:

```
============================================================

// LogicOr.cpp

#include<iostream.h>
#include<string>
using namespace std;

void main()
{

  string aHair = “Brown”, aEyes = “Blue”, bHair = “Black”,
     bEyes = “Brown”, cHair = “Red”, cEyes = “Black”;

  // (1)
  bool hairCheck = (aHair == “Brown”) || (cHair == “Red”);
  cout << hairCheck << endl;

  // (2)
  bool eyesCheck = (aEyes == “Black”) || (bEyes == “Brown”)
      || (cEyes == “Blue”);
  cout << eyesCheck << endl;

  // (3)
  hairCheck = (aHair == “Black”) || (bHair == “Red”)
  || (cHair == “Brown”);
  cout << hairCheck << endl;

}

============================================================
```

ඉහත (1) අවස්ථාවේදී `hairCheck` හි අගය true වේ. මක්නිසාද යත් `(aHair == “Brown”)` හා `(cHair == “Red”)` යන අවස්ථා දෙකම true අගයන් ගන්නා හෙයිනි.

(2) අවස්ථාවේදී `(bEyes == “Brown”)` හැරෙන්නට අනෙක් expressions අසත්‍ය ය. නමුත් `eyesCheck` හි අගය true බවට පත්වේ. මෙය සිදුවන්නට හේතුව නම්, || මගින් true නිපදවීමට, එය පරීක්ෂා කරන අවස්ථා එකක් හෝ සත්‍ය වීම ප්‍රමාණවත් වීමය.

(3) අවස්ථාවේදී සියලුම expressions, false අගය ගැනීම නිසා || මගින් false අගය නිපදවයි. || මගින් true නිපදවීමට නම් එය පරීක්ෂා කරන අවස්ථා එකක් හෝ සත්‍ය වීම අත්‍යවශ්‍ය වීම නිසා මෙය සිදුවේ.

Logical NOT(!)

ඔබට ඉහතදී හමුවූ පුද්ගලයා දැන් C වෙත පැමිණ මෙසේ කියතැයි සිතන්න: “ඔබේ ඇස් නිල් පාටයි!” මෙහිදී C කුමක් සිතුවද මෙය ඇසූ විටම ඔබට වැටහෙනු ඇත්තේ මෙම පුද්ගලයා C ට අසත්‍ය දේ ප්‍රකාශ කරන බවයි. මක්නිසාදයත් C ගේ ඇස් නිල් පාට නොව කාලවර්ණ බව ඔබට පෙනෙන හෙයිනි. පුද්ගලයාගේ අසත්‍ය ප්‍රකාශය expression එකක් මගින් මෙසේ ලිවිය හැක:

```
cEyes == “Blue”
```

දැන් කිසිවෙකු ඔබෙන් මෙම ප්‍රශ්ණය අසයි: “ඔහු C ගැන කියූ දෙය **අසත්‍යයක්ද?**”. ඔබේ පිලිතුර වන්නේ “ඔව්” යන්නයි. එහිදී එම ප්‍රශ්ණය ආසා ඇති ආකාරයට පිලිතුරු සැපයීමට ඔබ ලෙස තීරණය කල යුතු වන්නේ පුද්ගලයාගේ ප්‍රකාශණය **අසත්‍යද** යන වගයි. ඔබට ඉහත හමුවූ Logical operators මගින් සිදුකලේ කිසියම් දෙයක් සත්‍ය දැයි සොයා බැලීමයි. නමුත් ඉහත පරිදි කිසිවක් **අසත්‍ය** දැයි පරීක්ෂා කිරීමට නම් **Logical NOT** හි සහය අප ලබාගත යුතුය. Logical NOT මගින් සිදුවන්නේ කිසියම් අවස්ථාවක් අසත්‍ය නම් true අගය නිපදවීමයි. මෙය පහත වැඩසටහනෙහි දැක්වේ:

```
============================================================

// LogicNot.cpp

#include<iostream.h>
 #include<string>
using namespace std;

void main()
{

  string aHair = “Brown”, aEyes = “Blue”, bHair = “Black”,
       bEyes = “Brown”, cHair = “Red”, cEyes = “Black”;

  bool cEyesCheck =!(cEyes == “Blue”);
  cout << cEyesCheck << endl;

}

============================================================
```

ඉහත වැඩසටහනෙහි `cEyes == “Blue”` යන expression එක මගින් false නිපදවුනහොත් **!** – operator එක මගින් true නිපදවෙනු ඇත. Expression එක true වුවහොත් false අගය නිපදවේ. මේනිසා `cEyesCheck` හි අගය ඉහතදී true අගයක් ගනු ඇත.

Relational operators

C අනෙක් දෙදෙනාටම වඩා උසින් වැඩි, ඇය *High heels* පැලඳ ගෙන සිටින නිසා බව ඔබ නිරීක්ෂණය කරයි. A අනෙක් දෙදෙනාටම වඩා උසින් අඩුය. ඔබ දැන් මෙසේ කිවහොත් එය සත්‍ය වනු ඇත: “A, B ට වඩා උසින් අඩුයි”. මෙසේ කිවහොත් එය අසත්‍ය වනු ඇත: “C, B ට වඩා උසින් අඩුව පෙනෙයි”. ඔබ මෙහිදී සිදු කරන්නේ මේ තිදෙනාගේ උස වල අනුපාතිකයන් අතර සම්බන්ධයක් (Relation) ගොඩනැගීමයි. Relational operators වල කාර්යය වන්නේද මෙවැනි අවස්ථා අතර අනුපාතික සම්බන්ධයක් ඇති කිරීමය.

Greater than operator (>)

Greater than operator මගින් සිදුවන්නේ එය පරීක්ෂා කරන අවස්ථාවන් යුගල අතරින් දකුණු පස ඇති අවස්ථාව, වම් පස අවස්ථාවට වඩා අගයෙන් විශාල දැයි නිරීක්ෂණය කිරීමයි. එය එසේ නම් > - operator එක විසින් true අගයද, එය එසේ නොවන්නේ නම් false අගයද නිපදවේ. පහත උදාහරණය දෙස බලන්න. එහි සලකා බැලෙන්නේ ඉහත උදාහරණයේ ආ A,B,C තිදෙනාගේ උස අගයන් පිලිබඳවය:

```
============================================================

float aHeight = 4.5, bHeight = 4.7, cHeight = 5.0;

// (1)
bool heightCheck = aHeight > cHeight;
// (2)
heightCheck = cHeight > bHeight > aHeight;

============================================================
```

ඉහත (1) කොටසින් සිදුවන්නේ `aHeight` යන variable එකෙහි අගය `cHeight` යන variable එකෙහි අගයට වඩා අගයෙන් විශාල දැයි පරීක්ෂා කිරීමය. ඔබට පෙනෙන පරිදි මේ අවස්ථාවේදී එය අසත්‍ය හෙයින් > මගින් false අගය නිපදවේ.

(2) අවස්ථාවේදී සිදුවන්නේ කුමක්ද? එම අවස්ථාවේදී සිදුවන්නේ “C අනෙක් දෙදෙනාටම වඩා උසින් වැඩියි” යන කියමන පරීක්ෂාවට ලක් කිරීමයි. එනම්, `cHeight > bHeight` යන expression එක මගින් `cHeight`, `bHeight` ට වඩා විශාල දැයි පරීක්ෂා වේ. ඉන්පසු `bHeight > aHeight` expression එක මගින් `bHeight`, `aHeight` ට වඩා විශාල දැයි පරීක්ෂා වෙ. මේ අනුව `cHeight > bHeight > aHeight` හි පොදු හරය ලෙස ලැබෙන true හෝ false අගයෙන් ගම්‍ය වන්නේ ඉහත කියමනෙහි සත්‍ය-අසත්‍යතාවයි. තවත් පැහැදිලි ලෙස කියතහොත්, `cHeight > bHeight` හා `bHeight > aHeight` යන අවස්ථා දෙකෙන්ම true අගය නිපදවුන හොත් එහි අදහස නම් `cHeight` හෙවත් C ගේ උස අනෙක් දෙදනාට වඩා වැඩි බව සත්‍යයක් බවයි.

නමුත් දැන් මෙහිදී එම expression එකෙන් කියවෙන අදහසම විරුද්ධ අතටද සිතිය හැක. එනම් මෙසේද කිව හැකි බව ඔබට පෙනෙනු ඇත**:** ඉහත expression එක මගින් `aHeight` හෙවත් A ගේ උස, `bHeight` හෙවත් B ගේ උසටද, `cHeight` හෙවත් C ගේ උසටද වඩා අඩු දැයි පරීක්ෂා කෙරේ.

Less than operator (<)

Less than operator යනු Greater than operator හි විරුද්ධ අවස්ථාවයි. එනම්, එය මගින් සිදුවන්නේ එය පරීක්ෂා කරන අවස්ථාවන් යුගල අතරින් දකුණු පස ඇති අවස්ථාව, වම් පස අවස්ථාවට වඩා අගයෙන් විශාල දැයි පරීක්ෂා කිරීමයි. එය සත්‍ය නම් < විසින් true අගයද, එය එසේ නොවන්නේ නම් false අගයද නිපදවය. පහත උදාහරණයෙන් මෙය දැක්වේ:

```
============================================================

float aHeight = 4.5, bHeight = 4.7, cHeight = 5.0;

// (1)
bool heightCheck = cHeight < aHeight;

// (2)
heightCheck = aHeight < bHeight < cHeight;

============================================================
```

ඉහත (1) කොටසින් නිපදවෙන්නේ false අගය බවද (2) කොටසින් නිපදවෙන්නේ true අගය බවද ඔබට වැටහෙනු ඇත. තව දුරටත් පැහැදිලි කරතහොත් (2) වැනි අවස්ථාවෙන් කියවෙන්නේ පහත කියමනයි: “`aHeight, bHeight, cHeight` යන variables ත්‍රිත්වය අතරින් කුඩාම අගයක් ගන්නේ `aHeight` ය.“ ඒ අනුව ඉහත එහි අගය true වේ.

Greater than or equal to operator (>=)

මෙම operator එක යනු ඔබ ඉහතින් දුටු Greater than operator සහ Equal to operator එකෙහි එකතුවයි. එනම් එය එහි දකුණු පස ඇති expression එක, වම් පස ඇති expression එකට වඩා විශාල දැයි පරීක්ෂා කරනවා මෙන්ම, එම expressions එක යුගල සමානදැයඳ පරීක්ෂා කරයි. එම අවස්ථාවන් දෙකින් එකක් හෝ සත්‍ය නම් එය true නිපදවයි:

උදා.
```
============================================================

int a = 10, b = 20, c = 10;

// (1)
bool check = (b >= a);
// (2)
check = (c >= a);

============================================================
```

(1) අවස්ථාවෙහිදී, `d, a` ට වඩා විශාල නිසා `check` හි අගය true වේ.
(2) අවස්ථාවෙහි `a` හා `c` සමාන නිසා `check` හි අගය true වේ.

Less than or equal to operator (>=)

මෙය Greater than or equal to operator එකෙහි විරුද්ධ අවස්ථාවයි. එයින් පරීක්ෂා කෙරෙන්නේ එහි දකුණු පස ඇති expression වම් පස ඇති expression එකට වඩා කුඩා හෝ සමාන දැයි යන්නයි.

Equal to operator (==)

Equal to operator මගින් පරීක්ෂා වන්නේ එහි පරීක්ෂාවට ලක් වෙන අගයන් යුගල හරියටම සමානද යන්නයි:

උදා:

```
============================================================

int a = 10, b = 20, c = 10;

// (1)
bool check = (b == a);
// (2)
check = (c == a);

============================================================
```

(1) අවස්ථාවේදී false නිපදවෙන අතර (2) අවස්ථාවේදී true නිපදවේ.

Not equal to operator (!=)

මෙය Equal to operator හි විරුද්ධ අවස්ථාවයි. පරීක්ෂාවට ලක්වෙන අගයන් යුගල එකිනෙකට වෙනස් නම් එය true අගය නිපදවයි, සමාන නම් false නිපදවයි:

උදා:

```
============================================================

int a = 10, b = 20, c = 10;

// (1)
bool check = (b != a);
// (2)
check = (c != a);

============================================================
```

(1) අවස්ථාවේදී true නිපදවෙන අතර (2) අවස්ථාවේදී false නිපදවේ.

<u>Increment & Decrement operators</u>

Increment operator (++)

මෙය Addition operator (+) හි විකල්පයක් ලෙසද හැදින්විය හැකිය. නමුත් increment operator මගින් සිදුවන්නේ කිසියම් සංඛ්‍යාත්මක අගයක් එකකින් පමණක් ඉහල දැමීමය. මේ නිසා, සංඛ්‍යාත්මක අගයකට 1 ට වඩා ඉහල සංඛ්‍යාවක් එක්වරම එකතු කිරීමට අවශ්‍ය වූ විට increment operator එක පාවිච්චි කල නොහැක. Increment operator හි ව්‍යාකරණ විධිය පහත දැක්වේ:

```
============================================================
```

```
int number = 50;
number++;
cout<< number << endl;

============================================================
```

ඉහත, number හි අගය එකකින් ඉහල නංවා ඇත. එනිසා එහි නව අගය 51 ලෙස print වනු ඇත.

Decrement operator (--)
මෙය increment operator හි විරුද්ධ අවස්ථාවයි. එනම් decrement operator මගින් සිදුවන්නේ කිසියම් සංඛ්‍යාත්මක අගයක් 1 කින් පහල දැමීමයි:

උදා:

```
============================================================

int number = 50;
number--;
cout<< number << endl;

============================================================
```

49 print වේ.

Assignment Operators

Assignment operators භාවිතා කරන්නේ variables වලට අගයන් ආදේශ කිරීමේදීය.

Assignment operator (=)

මෙය ඔබ දැනටමත් හඳුනන operator එකකි. සරල ආදේශයන් සඳහා මෙය භාවිතා වේ:

උදා:

```
============================================================

int number;
number = 15;

============================================================
```

Addition assignment (+=)

Addition assignment operator එකද addition operator හි ආදේශකයක් ලෙස භාවිත කල හැක. මෙය භාවිතා කරන ආකාරය හඳුනාගැනීමට addition operator මගින් සිදුකල හැකි assignment එකක් addition assignment මගින් සිදුකරන ආකාරය සලකා බලමු:

```
============================================================

int number;
number = 15;
// (1)
number = number + 20;
// (2)
number += 20;

============================================================
```

ඉහත (1) අවස්ථාවේදී number වෙත 20 ක් එකතු කර ඇත්තේ addition operator එක භාවිතා කිරීමෙනි. (2) අවස්ථාවේදී දැක්වෙන්නේ එම ආදේශ කිරීමම addition assignment භාවිතා කිරීමෙන් සිදුකරණ ආකාරයය. එහිදී, addition operator භාවිතයේදී expression එකෙහි වම් පස යේදිය යුතු දෙවැනි number යෙදුම ඉවත් වී ඇත. += මගින් සෘජුවම, අවශ්‍ය අගය number variable එකට එකතුවේ. මේනිසා කිසියම් variable එකකට කිසියම් සංඛ්‍යාවක් කෙලින්ම එකතු කරගැනීමට අවශ්‍ය වූ විට addition assignment operator භාවිතය වඩා පහසුවේ.

Subtraction assignment (-=)

මෙය addition assignment හි ප්‍රතිවිරුද්ධ අවස්ථාවයි:

උදා:

```
============================================================

number -= 20;

============================================================
```

ඉහතදී, number වෙතින් 20 ක් අඩුකර ඇත.

Multiplication assignment (*=)

මෙය සරල ගුණ කිරීම් සඳහා භාවිතා වේ. මෙය ක්‍රියාත්මක වන්නේ ඉහත operators යුගල ක්‍රියාත්මක වූ ආකාරයටමයි. පහත උදාහරණයෙන් මෙය පැහැදිලි වනු ඇත:

```
============================================================

number *= 20;

============================================================
```

ඉහත සිදුවන්නේ number හි අගය 20 න් සරල ලෙස ගුණවීමය. මෙය පහත multiplication operator මගින් සිදුකර ඇති ගුණ කිරීමට සමාන වේ:

```
============================================================

number = number * 20;

============================================================
```

Division assignment (/=) ක්‍රියාකරන්නේ ද ඉහත පරිදිමය:

උදා:
```
============================================================

number /= 20;

============================================================
```

ඉහත, number හි අගය 20 න් බෙදා, පිලිතුර number තුල ගබඩා වේ.

මෙතැනින් operators ගැන විස්තරය නිමාවට පත්වේ. නමුත් ඉහතදී සඳහන් නොවුනු operators වර්ගයක් ඇත. මේවා Bitwise operators නමින් හැඳින්වේ. Bitwise operators බොහෝවිට භාවිත වන්නේ ද්විමය සංඛ්‍යා සමග කටයුතු කිරීමේදීය. මෙය ඔබ දැන්ම අතනොගසන කටයුත්තක් නිසා Bitwise operators ගැන කථා කිරීම වෙනත් අවස්ථාවකට ඉතිරි කර ඇත.

4. පරිගණක වැඩසටහනක ක්‍රියාකාරීත්වය

පාලනය කිරීම.
(Program Flow Control Statements)

(මෙම පරිච්ඡේදය කියවීමට පෙර, මෙම පොතෙහි **Appendix** කොටසේ ඇති **"Debug කිරීම සහ Errors හඳුනාගැනීම"** යන කොටස කියවීම සුදුසුය)

පරිගණක වැඩසටහනක් ගලා යන ජල පහරකට සමානය. ජල පහරක ඇති ශක්තියෙන් නිසි ප්‍රයෝජන ගැනීමට නම්, අප එම ජල පහරෙහි ගමනට බාධා කල යුතුය. ජල පහරක ගමන පාලනය නොකල විට, එහි ගැබ්ව ඇති විශාල ශක්තියෙන් ප්‍රයෝජනයක් ගැනීමට නොහැකි වෙයි. පාලනය නොකල පරිගණක වැඩසටහනක්ද පාලනය නොකල ජල ධාරාවක් මෙන් නිකරුනේ ගලයි. පරිගණක වැඩසටහනකටද පාලනයකින් තොරව ගැලීමට ඉඩ හලවිට, එයින් කිසියම් ප්‍රයෝජනවත් දෙයක් කරගැනීමට අපට හැකි නොවේ. මෙනිසා අප කලයුතු වන්නේ, අප විසින් නිර්මාණය කරනු ලබන පරිගණක වැඩසටහන්වල ක්‍රියාකාරීත්වය නිසි පරිදි පාලනය කිරීමෙන්, ඒවා යහපත් මෘදුකාංගයන් බවට පත් කිරීමය.

පරිගණක වැඩසටහනක මෙම "යහපත් බව" යන්නෙන් හැඳින්වෙන්නේ කුමක්ද? සරල ලෙස ගත්විට එය කරුණු කිහිපයක් මත රඳා පවතියි. ඒ අතරින් එකක් නම්, කිසියම් **පරිගණක වැඩසටහනක්, එය පාවිච්චි කරන්නා වෙත දක්වන සැලකිල්ලය**. වැඩසටහනක් පාවිච්චි කිරීම එය භාවිතා කරන්නා හට අපහසු හා සංකීර්ණ බව හැගේ නම්, එය එම වැඩසටහනෙහි ආකර්ශණීය භාවය පහත හෙලයි. තවත් කරුණක් නම් වැඩසටහනක් කිසියම් දෙයක් සිදුකිරීමට ගන්නා කාලයයි. පරිගණක වැඩසටහනක් ඔබ ලිවිය යුත්තේ හැකි තරම් අඩු කාලයක් ගතකොට එය භාවිතාකරන්නා හට අවශ්‍ය කාර්යය ඉටුකරදීමේ අරමුණින් යුතුවය.

මේ කාර්යයන් සිදු කිරීමට නම් අප පරිගණක වැඩසටහන් වල ක්‍රියාකාරීත්වය අපට අවශ්‍ය පරිදි පාලනය කරගත යුතුය. මෙය සිදුකිරීම සඳහා C++ හි විවිධ ක්‍රමෝපායයන් ඇත. දැන් අප පරිගණක වැඩසටහනක ක්‍රියාකාරීත්වය පාලනය කිරීමේ එම ක්‍රමෝපායයන් හඳුනාගැනීම අරඹමු.

ඉහත කියූ ක්‍රමෝපායයන් සියල්ලම ගත්විට ඒවා හැඳින්වීමට "Program control statements" යන්න භාවිතා වේ. Program control statements වර්ග කීපයකට බෙදා දැක්විය හැක. පහත දැක්වෙන්නේ ඒවා වර්ගීකරණය කර ඇති ආකාරයයි.

Conditional Statements	Looping Statements	Jumping Statements
if – statement	while - statement	goto – statement
if…else - statement	do…while - statement	
else…if - statement	for - statement	
switch…case - statement	break & continue	

දැන් අප මෙම Program control statements වැඩසටහන් වලදී භාවිතා කරණ ආකාරය වටහා ගැනීමට උත්සාහ කරමු.

Conditional Statements

Conditional statements මගින් පරිගණක වැඩසටහනක් තුල උද්ගත වන තත්වයන්ට අනුව එම වැඩසටහනෙහි ක්‍රියාකාරීත්වය හෙවත් එහි ගලායාම පාලනය කල හැකියි. පහත දැක්වෙන්නේ එම conditional statements සඳහා පැහැදිලි කිරීම්ය.

if - statement

if - statement එක මගින් සිදුවන්නේ පරිගණක වැඩසටහනක් තුලදී කිසියම් තත්වයක් සැපිරුණ විට පමණක් යම් ක්‍රියාවක් සිදුකිරීමය.

නඳසුනක් ලෙස කිසිවෙකු මෙසේ කීවා යැයි සිතන්න: "මා '*2001: A space odyssey*' නැමැති පොත මිලදී ගන්නේ එම පොතෙහි මිල රුපියල් 200 ට අඩුනම් පමණයි". එනම්, ඔහුට පොත මිලදී ගැනීමට නම් එහි මිල රුපියල් 200 ට අඩු වීමේ තත්වය සැපිරිය යුතුය. මෙම කියමන පදනම් කරගෙන අප දැන් if භාවිතා කරමින් පරිගණක වැඩසටහනක් ලියමු:

```
============================================================

// if.cpp

#include<iostream.h>

void main()
{

  int bookPrice = 180;


  if(bookPrice < 200)
   {
       cout << "I buy the book(1)." << endl;
  }

   bookPrice = 230;

  if(bookPrice < 200)
   {
       cout << "I buy the book(2)." << endl;
  }

}

============================================================
```

ඉහත වැඩසටහනෙහි (1) අවස්ථාවේදී, if මගින් bookPrice හි අගය 200 ට වඩා අඩුදැයි පරීක්ෂා කෙරේ. මෙය සිදුවන්නේ if හි වරහන් තුල ඇති (bookPrice < 200) යන expression එක මගිනි. if මගින් පරීක්ෂා කෙරෙන්නේ මෙම expression එක සත්‍ය වේදැයි යන්නයි. වෙනත් අයුරකින් කිවහොත්, if මගින්, bookPrice හි අගය 200 ට වඩා අඩු වීම යන තත්වය සැපිරෙන්නේ දැයි පරීක්ෂා කරයි. එම තත්වය සම්පූර්ණ වුවහොත්, හෙවත් (bookPrice < 200) යන්න true වුවහොත්, if යටතේ ඇති පරිගණක කේතයන් ක්‍රියාත්මක වේ. දැන් මෙහිදී bookPrice හි අගය 180 නිසා (bookPrice < 200) යන expression එක true හෙවත් සත්‍ය වේ. එනම් bookPrice හි අගය 200ට වඩා අඩු වීම යන තත්වය සම්පූර්ණ වී ඇත. මෙනිසා if තුල ඇති කේතය ක්‍රියාත්මක වී "I buy the book(1)" යන්න print වේ. (2) අවස්ථාවෙහි if මගින් නැවතත් එම පරීක්ෂාව සිදුවේ. මෙවර bookPrice හි අගය 200 ට වැඩි නිසා (bookPrice < 200) යන expression එක false හෙවත් අසත්‍ය වේ. එහෙයින් "I buy the book(2)." යන්න print නොවේ. එසේනම්, if මගින් වැඩසටහන

ක්‍රියාත්මක වන ආකාරය පාලනයට ලක් කර ඇත . එනම් දෙවන අවස්ථාවේ ඇති cout විධානය ක්‍රියාත්මක නොවන අයුරින් if මගින් වැඩසටහන පාලනය කෙරේ.

if…else - statement

දැන් ඉහත පුද්ගලයා මෙසේ කියයි: "පොතෙහි මිල 200 ට අඩුවුනොත් මා එය මිලදී ගන්නා අතර, මිල 200 ට වඩා වැඩිවුනොත් මම වෙනත් පොතක් මිලදී ගනිමි." මෙම කියමන තුලදී ඔහු අවස්ථා 2 ක් පිලිබඳ කථා කර ඇත. පෙර අවස්ථාවේදී ඔහුට ඇත්තේ තෝරාගැනීම් 1ක් පමණක් නමුත් මේ අවස්ථාවේදී ඔහුට තෝරා ගැනීමේ අවස්ථා 2 ක් ඇත. පරිගණක වැඩසටහනකදී මෙවැනි තෝරාගැනීමේ අවස්ථා යුගලක් පමණක් ඇති අවස්ථා වලදී අපට පාවිච්චි කිරීමට සිදුවන්නේ if…else ය. if…else යනු if හි ක්‍රියාකාරීත්වයෙහිම වැඩි දියුණුවකි. දැන් ඉහත ප්‍රකාශය if…else යොදා ගනිමින් පරිගණක වැඩසටහනකට අන්තර්ගත කරන්නේ කෙසේදැයි බලමු:

```
============================================================

// ifElse.cpp

#include<iostream.h>

void main()
{

    int bookPrice = 250;

    if((bookPrice < 200))
    {
         cout << "I buy the book." << endl;

    }else {

          cout << "I buy some other book." << endl;
  }
}
```

ඉහතදී if මගින් bookPrice හි අගය පෙර පරීදිම 200 ට අඩුදැයි පරීක්ෂා වේ. bookPrice දැන් 250 ක් වන නිසා පලමුවන cout විධානය ක්‍රියාත්මක නොවේ. ක්‍රියාත්මක වන්නේ else තුල ඇති cout විධානයයි. bookPrice හි අගය 200 ට අඩු වූවානම් else තුල ඇති cout විධානය ක්‍රියාත්මක නොවන අතර, ක්‍රියාත්මක වන්නේ if තුල ඇති cout විධානයයි.

else…if - statement

පෙර ප්‍රකාශය පුද්ගලයා එය දැන් මෙලෙස වෙනස් කරයි: "පොතෙහි මිල 200 ට අඩුවුනොත් මා එය මිලදී ගන්නා අතර, මිල 150 ට වඩා අඩුවුනොත් මම පොත් දෙකක් ගෙන එකක් මගේ යහලුවකුට තෑගි කරමි. පොතෙහි මිල 100 ටද වඩා අඩු වුවහොත් මම පොත් 3 ක් ගෙන 2 ක් මගේ යහලුවන් දෙදෙනෙකුට තෑගි කරමි." දැන් සිදුවන්නේ කුමක්ද? ඔහු හට තෝරා ගැනීමට දැන් **අවස්ථා 3 ක් ඇත**. පරිගණක වැඩසටහනකදී, මෙලෙස තෝරාගැනීමේ අවස්ථාවන් 2 කට වැඩියෙන් ඇති අවස්ථා වලදී else…if control statement එක පාවිච්චි කිරීමට හැකිය. පහත වැඩසටහනින් මෙය පැහැදිලි වනු ඇත:

```
=====================================================

// elseIf.cpp

#include<iostream.h>

void main()
{

    int bookPrice = 85;

    if((bookPrice < 200) && (bookPrice > 150))
    {

         cout << "I buy the book." << endl;
```

```
   }else if ((bookPrice < 150) && (bookPrice > 100))
   {

         cout << “I buy 2 copies of the book.” << endl;

    }else if (bookPrice < 100)
    {

         cout << “I buy 3 copies of the book.” << endl;

    }

}
```

==========

if මගින් මුලින්ම && operator එක භාවිතා කිරීමෙන්, bookPrice හි අගය 200 ට අඩුදැයඳ යන්න සහ 150 වැඩඳැයඳ යන්න පරීක්ෂා කෙරේ. එනම් bookPrice හි අගය ඇත්තේ 200 හා 150 අතර දැයි සොයා බැලේ. bookPrice අගය වන 85, 200 ට අඩු නමුත් 150 ට වඩා වැඩි නොවන නිසා පලමු වැනි cout විධානය ක්‍රියාත්මක නොවන බව පැහැදිලියි. යොදා ඇති operator එක && නිසා පරීක්ෂා කරන අවස්ථා දෙකම සත්‍ය විය යුතුයි.

මීලගට යොදා ඇති else if මගින් bookPrice අගය ඇත්තේ 150 ත් 100 ත් අතර දැයි පරීක්ෂාවේ. එයත් සත්‍ය නොවන හෙයින් මීලඟට තෙවන අවස්ථාවේ else…if මගින් bookPrice 100 ට අඩුදැයි බැලේ. මෙය සත්‍ය වන නිසා එය තුල ඇති cout විධානය ක්‍රියාත්මක වේ. මේනිසා ඉහත වැඩසටහන ඔබ ක්‍රියාත්මක කලහොත් print වන්නේ තෙවැනි අවස්ථාවේ else if තුල ඇති cout විධානය පමණි. මෙලෙස, තෝරා ගැනීමට ඇති අවස්ථා ගණන වැඩිවන විටදී එම අවස්ථා ගණනට අනුරූප else if statements ගණනකින් එම අවස්ථා සමග කටයුතු කල යුතුය.

Looping Statements

දෙවන සණයේ Programming control statements වර්ගය වන්නේ Looping statements ය. Looping statements ඉගෙනීම සඳහා අප දැන් සරල පරිගණක වැඩසටහනක් සලකා බලමු.

මෙම වැඩසටහන ක්‍රියාකරන්නේ මේ ආකාරයටයි: අප විසින් වැඩසටහනට කිසියම් සංඛ්‍යාත්මක අගයක් සැපයිය යුතුය. එවිට වැඩසටහන මෙම සංඛ්‍යාව පරීක්ෂාකර, එය 100 හා 200 අතර සංඛ්‍යාවක් නොවෙයි නම් එහි අගය වෙනසකට ලක් කරයි. එනම්, සංඛ්‍යාව 200 ට වැඩි නම් එහි අගය 200 දක්වා අඩු කරන අතර, එය 100 ට අඩුනම් එහි අගය 100 දක්වා ඉහල දමයි. වෙනත් වචන වලින් කිවහොත්, වැඩසටහන විසින් අප සපයන සංඛ්‍යාව නියමිත පරාසයකට ඇතුල් කල හැකි වනසේ වෙනස් කරයි. කෙසේද අප මෙවැනි වැඩසටහනක් සාදන්නේ? එය ඉතාමත් සරලයි.

සිදුකලයුතු දේ වන්නේ මේවායි: (1) වැඩසටහන විසින් අප දෙනු ලබන සංඛ්‍යාව පරීක්ෂා කර, එය 100 ට අඩුද, එසේ නැතිනම් 200 ට වැඩි දැයි යන්න තීරණය කල යුතුයි. (2) සංඛ්‍යාවේ අගය 100 ට අඩු නම්, එය 100 වනතුරු එහි අගය නොකඩවා ඉහල දැමිය යුතුයි. (3) සංඛ්‍යාවේ අගය 200 ට වැඩි නම්, එය 200 වනතුරු එහි අගය නොකඩවා පහල දැමිය යුතුයි. අප Looping statements භාවිතයට ගන්නේ මෙවැනි කටයුතු සිදුකලයුතු අවස්ථාවලදීයි. පහත, while නැමැති looping statement එක ආධාරයෙන් මෙය සිදුකරන ආකාරය දැක්වේ.

while - statement

while මගින් සිදුකරන්නේ, කිසියම් තත්වයක් සැපිරෙන තාක්, එතුල අන්තර්ගත වන පරිගණක කේත නැවත නැවත ක්‍රියාත්මක කරවීමයි. මෙහිදී නැවත if statement එක ක්‍රියාකල ආකාරය සිහියට නගා ගන්න. if මගින් සිදුවූයේ කිසියම් තත්වයක් සැපිරේ නම් එක්වරක් පමණක් තමා තුල ඇති කේතයන් ක්‍රියාත්මක කරවීමයි. නමුත් while වැනි looping statements මගින් සිදුවන්නේ කිසියම් තත්වයක් සත්‍ය වන තෙක් කේතයන් නොනවත්වා නැවත නැවතත් ක්‍රියාත්මක කිරීමයි.

දැන් අප ඉහත සඳහන් කල වැඩසටහන while මගින් සකසන්නේ කෙසේදැයි බලමු:

```
=========================================

// while.cpp

#include<iostream.h>

void main()
{
  int number=0;
  cout<<"Enter a number: ";
  cin>> number;

  while(number<100)
  {
   number++;
   cout<<number<<“, “;
  }

  while(number>200)
  {
   number--;
   cout<<number<<“, “;
  }

   cout<<number<<endl;
}

=====================================
```

ඉහත වැඩසටහනේදී පලමුවෙන්, cout හා cin මගින් අප number නැමැති variable එක තුලට අගයක් ඇතුල් කරමු. (cout මගින් සිදුවන්නේ කිසියම් දත්තයක් console එක මතට output කිරීම වූ සේ, cin මගින් කෙරෙන්නේ අප console එක මතට type කරන කිසියම් දත්තයක් වැඩසටහන තුලට ආදානය කිරීමයි. ඉහත එය ඇතුල් වන්නේ number තුලටය. cout භාවිතා කිරීමේදී ඒ සමඟ අප “ << ” ලකුණ යොදමු. cin භාවිතා කිරීමේදී අප එය ඉදිරියෙන් යෙදිය යුතු වන්නේ “ >> “ ලකුණය.) දැන් ඔබට while statements 2 ක් දැකිය හැකිය පලමු වැන්නෙන් පරීක්ෂා කෙරෙන්නේ number හි අගය 100 අඩුද යන්නයි. එය සත්‍යයක් නම්, එම while තුල ඇති කේතය අනවරතව ක්‍රියාත්මක වීම ඇරඹේ. එනම්, number හි අගය නැවත නැවතත් 1 බැගින් ඉහල යයි. මෙය නවතින්නේ number = 100 වූ විටය.

දෙවැනි while loop එකින් පරීක්ෂා කරන්නේ number හි අගය 200 ට වඩා වැඩඳ යන්නය. එය සත්‍ය නම්, while තුල කේතය අනවරතව ක්‍රියාත්මක වීමෙන් number හි අගය 200 දක්වා ඉහල යනු ඇත. number හි අගය 100 ත් 200 ත් අතර නම්, while loops දෙකින් එකක් හෝ ක්‍රියා නොකරන බව පැහැදිලියි. මෙම වැඩසටහන ක්‍රියාත්මක කරන්න. Terminal window එක තුල කිසියම් සංඛ්‍යාවක් type කර enter key එක ඔබන්න. එවිට cin විධානය ඔබ type කල සංඛ්‍යාව number variable එකට ආදේශ කරනු ඇත. ඔබ එයට 100 ට අඩු සංඛ්‍යාවක් සැපයුවහොත් 100 දක්වා එහි අගය වැඩි වන ආකාරයද, 200 වැඩි සංඛ්‍යාවක් සැපයුවහොත් 200 දක්වා එහි අගය අඩුවන ආකාරයද, වැඩසටහන මගින් output කරනු ඇත.

(මෙම වැඩසටහනෙහි ක්‍රියාත්මක වීම හොඳින් දැක ගැනීමට නම් එය debug කර ක්‍රියාත්මක වීමේ ලක්ෂය ගමන් කරන ආකාරය නැරඹිය යුතුය. Debug කිරීම පිලිබඳව **Appendix** කොටසෙහි සඳහන්වේ)

do…while - statement

දෙවැනි loops වර්ගය වන්නේ මෙයයි. මෙය එක් කරුණකින් හැරෙන්නට while statement එකට හරියටම සමානයි. ඉහත උදාහරණයම Do… while මගින් ලියූ විට පහත පරිදි දිස්වනු ඇත:

```
================================

// Dowhile.cpp
```

```
#include<iostream.h>

void main()
{
  int number=0;
  cout<<"Enter a number: ";
  cin>> number;

  do
  {
    number++;
    cout<<number<<", ";
  }while(number<100);

  do
  {
    number--;
    cout<<number<<", ";
  }while(number>200);

  cout<<number<<endl;
}
```

==========

do… while හි ව්‍යාකරණය තරමක් වෙනස් බව ඔබට දැකිය හැක. do… while හි සිදුවන්නේ සියල්ලට පලමුවෙන් do යටතේ ඇති කේතයන් ක්‍රියාත්මක වීමයි. එනම් පලමුවන වාරයේදී while මගින් පරීක්ෂා කරණ තත්වය සත්‍ය වුවත් අසත්‍ය වුවත් do යටතේ ඇති කේතය අනිවාර්යයෙන් ක්‍රියාත්මක වේ. ඉන්පසු while හි පරීක්ෂාව සිදුවන අතර, එය අසත්‍ය වුවහොත් loop එක බිඳී යයි. සත්‍ය වුවහොත් do තුල ඇති කේත දිගටම ක්‍රියාත්මක වේ. මේ අනුව, ඉහත while loops මගින් පරීක්ෂා කෙරෙන තත්වයන් යුගලම අසත්‍ය වුවත්, පලමු do statement එක තුලදී සංඛ්‍යාවට 1ක් එකතු වී එය print වනු ඇති අතර, දෙවැනි do මගින් එයින් 1ක් අඩු වී print වනු ඇත. මෙය ඉහත අපගේ වැඩසටහන සඳහා එතරම් අර්ථයක් සහිත ක්‍රියාවක් නොවුනත්, එයින් ඔබට do… while ක්‍රියාකරන ආකාරය උගත හැකියි.

for - statement

for statement එක එක් අතකින් while හිම දියුණු අවස්ථාවක් ලෙස සැලකිය හැක. for මගින් සිදුවන්නේ ද කිසියම් තත්වයක් සැපිරෙන තුරු ඒ යටතේ ඇති පරිගණක කේත රැස අනවරත ලෙස ක්‍රියාත්මක කරවීමය. නමුත් for හි while ට වඩා මඳ සංකීර්ණ බවක් ඇත. දැන් අප මෙම for statement එක ක්‍රියා කරන්නේ කෙසේදැයි විමසා බලමු.

==========

```
for(int i = 0; i < 10; i++)
{
  // code…
}
```

==========

ඉහතින් දැක්වෙන්නේ ඔබට බොහෝවිට දැකීමට හැකි ආකාරයේ for – statement එකකි. එහි වරහන් තුල කොටස් 3 ක් ඔබට දැකිය හැකියි. පලමු කොටසින් int වර්ගයේ variable එකක් define වී ඇත. දෙවන කොටසින් එම variable එකෙහි අගය කිසියම් සංඛ්‍යාවකට වඩා අඩුදැයි පරීක්ෂා වේ. තෙවැනි කොටසින් variable එකෙහි අගය 1 කින් ඉහල නැංවේ. for ක්‍රියාත්මක වන්නේ මේ කොටස් සියල්ල මගිනි.

පහත දැක්වෙන්නේ මෙම ක්‍රියාත්මක වීම සිදුවන ආකාරයයි:

for තුල පලමුව සිදුවන්නේ i නැමැති int variable එක define වීමෙන් එය 0 ට සමාන වීමයි. ඉන්පසුව < operator එක උපයෝගී කරගෙන i හි අගය 10 ට වඩා අඩුදැයි පරීක්ෂා කෙරේ. එය 10 ට වඩා අඩුනම් ඊලගට

සිදුවන්නේ i++ මගින් i හි අගය 1 කින් ඉහල දැමීමයි. එනිසා දැන් i හි අගය 1 වේ. දැන් for යටතේ ඇති කේත ක්‍රියාත්මක වීම ඇරඹේ. කේත සියල්ල එක්වරක් ක්‍රියාත්මක වීමෙන් පසු නැවත ක්‍රියාත්මක වීමේ ලක්ෂය ඉහලට පැමිණේ. මේ අවස්ථාවේදී for විසින් නැවතත් `i < 10` මගින් i හි අගය 10 ට අඩුදැයි බලයි. තවමත් එය 10 ට අඩුනිසා `i++` මගින් නැවතත් i හි අගය 1කින් ඉහල නංවයි. දැන් i = 2 බව ඔබට පෙනේ. ඉන් අනතුරුව නැවතත් for හි පරිගණක කේතයන් ක්‍රියාත්මක වේ. මෙලෙස නැවත නැවතත් i හි අගය 10 ට අඩුනම් එය 1කින් ඉහල යාමත්, for යටතේ ඇති කේත ක්‍රියාත්මක වීමත් සිදුවේ. මෙය ඇණහිටින්නේ i = 10 වන අවස්ථාවේදීය. එනම්, i = 10 වී ක්‍රියාත්මක වීමේ ලක්ෂය ඉහලට ආ විට, i < 10 යන්න අසත්‍ය වන නිසා i++ යන කේතය ක්‍රියා විරහිතව යයි. මේනිසා ඉන්පසු i එකකින් වැඩිවන්නේත් නැත, for යටතේ ඇති කේතයන් ක්‍රියාත්මක වන්නේත් නැත.

දැන් මෙය තවත් පැහැදිලි කර ගැනීම පිණිසද for සහ while අතර ඇති සම්බන්ධය වටහා ගැනීම පිණිසද, while මගින් ලියූ වැඩසටහනක් නැවත for මගින් ලියනු ලබන්නේ කෙසේදැයි පරීක්ෂා කර බලමු. පහත දැක්වෙන්නේ while මගින් ලියන ලද වැඩසටහනකි:

```
=============================================

      // whileProg.cpp

      #include<iostream.h>

      void main()
      {

        int count = 0;
        while(count < 20)
        {
          cout << “Go…” << endl;
          count++;
        }
        cout << “Halt!” << endl;
      }

=======================================
```

ඉහත වැඩසටහන ක්‍රියාත්මක කලහොත්ද “Go…” යන්න 20 වාරයක් print වන වනු ඇත. ඉන්පසු “Halt!” යන්න print වේ. දැන් මෙම සරල වැඩසටහන for statement එක භාවිතා කර ලියන්නේ කෙසේදැයි දැන් බලමු. පහත දැක්වෙන්නේ මෙම වැඩසටහනයි:

```
=================================================

     // forProg.cpp

      #include<iostream.h>

      void main()
      {
         for(int count = 0; count < 20; count++)
         {
            cout << “Go…” << endl;
         }
         cout << “Halt…” << endl;
      }

=================================================
```

ඉහත for මගින් ලියන ලද වැඩසටහනින් සිදුවන්නේද “Go…” යන්න 20 වාරයක් print වී, ඉන්පසු “Halt!” යන්න print වීමයි. දැන් මෙම වැඩසටහන් දෙකෙහි දක්නට ලැබෙන වෙනස් කම් මොනවාද? පලමුවෙන්, while යොදාගන්නා අවස්ථාවේදී count - variable එක define කර ඇත්තේ වෙනම ඒකකයක් ලෙසය. නමුත් for යොදාගන්නා අවස්ථාවේදී එය for statement එක තුලම, එහි කොටසක් ලෙස define කර ඇත. එමෙන්ම

count හි අගය ඉහල නැංවීමෙඳිද එම කාර්යය සිදුව ඇත්තේ for හිම කොටසක් ලෙසයි. එසේනම් for මගින් සිදුකර ඇත්තේ while මගින් සිදුකල කාර්යයන් ඒකරාශී කර එක් ස්ථානයකට ගැනීම බව කිව හැකිය.

break & continue

break හා continue යන යුගල ඉහත ඔබ දුටු සියලුම Looping statements තුලට අන්තර්ගත කිරීම සඳහා යොදාගැනේ. මේවායේ කාර්යය නම් Looping statements පාලනය කිරීමයි.

break හි කාර්යය නම් Looping statement එකක ක්‍රියාකාරීත්වය අතර මග නවත්වා දැමීමය. මෙයට උදාහරණයක් ලෙස, break statement එක while තුලට යොදා ඇති අන්දම පහත වැඩසටහනින් දැක්වේ:

```
=====================================================

      // break.cpp

      #include<iostream.h>

      void main()
      {

        int count = 0;
        while(count < 20)
        {
          cout << “Go…” << endl;
          count++;
          if(count==5)
          {
            break;
          }
        }
        cout << “Halt!” << endl;
      }

=======================================================
```

ඉහත වැඩසටහනේ count=5 වූ විට if තුල ඇති break statement එක ක්‍රියාත්මක වී while loop එක බිඳී යයි. මේ නිසා Go… යන්න print වෙනු ඇත්තේ 5 වරක් පමණි. break statement එක නොතිබුණා නම් Go… යන්න 20 වරක් print වන බව ඔබට පැහැදිලිය.

continue statement එක break statement එකින් මඳක් වෙනස් වේ. break මගින් කෙරෙන්නේ loop එකෙහි ක්‍රියාව සම්පූර්ණයෙන් නැවත්වීම වූ අතර continue මගින් කෙරෙන්නේ loop එකෙහි දැනට සිදුවෙමින් පවතින ක්‍රියාත්මක වීමේ වාරය පමණක් අතරමග නවත්වා ඊලඟ ක්‍රියාත්මක වීමේ වාරයට වැඩසටහන ක්‍රියාත්මක වීමේ ලක්ෂය පැන්වීමය. මෙය පහත වැඩසටහනින් පැහැදිලිවේ:

```
=======================================================

      // continue.cpp

      #include<iostream.h>

      void main()
      {
        int count = 0;
        while(count < 20)
        {
         count++;
         if(count==5)
         {
            continue;
         }
         cout << "Go..." << endl;
        }
```

```
        cout << "Halt!" << endl;
    }
```

```
=======================================================
```

ඉහතදී, count=5 වූ විට if තුල ඇති continue statement එක ක්‍රියාත්මක වන අතර, එමනිසා එම වාරයේ while හි ක්‍රියාත්මක වීම අතරමග ඇණ හිටීමෙන් `cout << "Go..." << endl;` කේතය ක්‍රියාත්මක වීම එක්වරක් මගහැරී යයි. නමුත් continue මගින් loop එක බිඳ දමන්නේ නැති නිසා, while දිගටම ක්‍රියාත්මක වේ. මේනිසා අවසානයේදී Go... යන්න print වී තිබෙනු ඇත්තේ 19 වරකි. continue statement එක නොතිබුණා නම් Go... යන්න 20 වරක් print වේ.

5. Functions

Functions යනු මොනවාදැයි ඔබ අසයි. නමුත් ඔබ එය දැනටමත් දැක ඇත. ඔබ ඉහත පරිච්ඡේද වලදී දුටු main(), C++ හි ඔබට හමුවෙන පලමු හා ප්‍රධාන function එකයි. නමුත් function එකක් අප අර්ථ දක්වන්නේ කෙසේද?

Function එකක් යනු C++ පමණක් නොව වෙනත් බොහෝ පරිගණක වැඩසටහන් වලද මූලික ක්‍රියාකාරී ඒකකයයි. වෙනත් වචන වලින් කියතහොත් functions නොමැතිව C++ මගින් පරිගණක වැඩසටහන් ලිවීමක් පිලිබඳව සිතිය නොහැකිය. Function එකක් තුල අඩංගු වන්නේ පරිගණකය ලවා කිසියම් ක්‍රියාවක් කරගැනීම සඳහා ලියන ලද පරිගණක කේත රැසකි. Function එකක් මගින් කිසියම් පරිගණක කේත රැසක් සඳහා ආවරණයක් සාදයි. එම කේතයන් මගින් සිදුවෙන කාර්යය අනුව Functions එකෙහි නම තීරණය කරනු ලැබෙයි. උදාහරණයක් ලෙස පරිගණක තිරය මත කිසියම් දෙයක් print කරන්නාවූ පරිගණක කේතයන් අඩංගු කොට නිමවන ලද Function එකක් print නමින් නම් කල හැක. ඔබ දුටු main නැමැති function එක වෙත එම නම ලැබී ඇත්තේ එය පරිගණක වැඩසටහනක පලමුවෙන්ම ක්‍රියාත්මක වන function එක වන හෙයිනි.

Functions මගින් පරිගණක වැඩසටහනකට සිදුවෙන සේවයන් කීපයක් පැහැදිලිව දැකිය හැකියි :

1. එක් වරක් ලියන ලද පරිගණක කේතයන් නැවත නැවත ප්‍රයෝජනයට ගැනීමේ පහසුකම සැලසීම.

2. පරිගණක වැඩසටහනක් කොටස් වලට බෙදා වෙන්කිරීම. මේ මගින් එම වැඩසටහන ලිවීම සහ වටහා ගැනීම පහසු වේ.

අප දැන් function එකක් යනු කුමක්දැයි හඳුනාගැනීමේ පලමු පියවර ලෙස එය පාවිච්චි කර ලියන ලද සරල පරිගණක වැඩසටහනක් කුමන ආකාරයක් ගනඳැයි විමසා බලමු. පහත දැක්වෙන වැඩසටහන පරීක්ෂා කර බලන්න. එය ඉදිරිපත් කර ඇත්තේ ඔබට function එකක් ක්‍රියාකරණ ආකාරය ගැන සරල ලෙස පෙන්වා දීමට පමණි. එනිසා එහි ඇති සියලු දේ වටහා ගැනීමට උත්සාහ නොකරන්න. Functions ගැන විස්තර, අප පහත වැඩසටහන ගැන සාකච්චා කිරීමෙන් අනතුරුව වැඩඳුරටත් සලකා බලමු:

```
=========================================================

// introToFunctions.cpp

#include<iostream.h>

void print();

void main()
{
  print();
}

void print()
 {
  cout << “Hi! This is my 1st function.” << endl;
}

=========================================================
```

අප variables ගැන කථා කිරීමේදී, variable එකක් define කිරීමට පෙර එය declare කල යුතු බව ඔබ දුටුවා. (මේ ක්‍රියාව පිලිබඳව ඔබගේ මතකය ගිලිහී ඇත්නම් කරුණාකර පලමු පරිච්ඡේදයේදී variables යටතේ එන කොටස සහ 2 වන පරිච්ඡේදයේදී int – data type යටතේ එන කොටස නැවත කියවන්න) මෙය functions සඳහාද පොදුවන කරුණකි. මේනිසා variable එකක් මෙන්ම function එකක් ද declare කර ඉන්පසු define කල යුතුයි. Function එකක් පලමුව declare කිරීමෙන් සිදුවන්නේ අප වැඩසටහන තුල එම function එක පාවිච්චි කිරීමට යන බව පවසමින් compiler මෘදුකාංගයට එය හඳුන්වාදීමය. එවිට එය වැඩසටහන compile කරගෙන යාමේදී, function declaration එක හමුවූ විට function එක සමග කටයුතු කිරීමට දැන සිටිනු ඇත. තවද, variable එකක් සම්බන්ධයෙන් ඔබ දුටු පරිදිම function එකකදීද declare සහ define කිරීම එක්වර සිදුකල හැකිය.

ඉහත වැඩසටහනේදී `void print()` යන්නෙන් සිදුවන්නේ මෙම function එක declare කිරීමේ ක්‍රියාවයි. එහි definition එක සිදුවන්නේ main function එකට පහලින් ඇති,

```
void print()
{
   cout << “Hi! This is my 1st function.” << endl;
}
```

යන කොටසිනි. එනම්, print function එක මගින් සිදුකරණ කාර්යය තීරණය කර එයට නියම අගයක් දෙන්නේ මෙහිදීය. එය සිදුකරණ කාර්යය නම්, `cout << “Hi! This is my 1st function.” << endl` යන කේතය ක්‍රියාත්මක කරවීම බව ඔබට පෙනෙනු ඇත.

main() තුලද `print()` ලෙස function එකෙහි නම සඳහන් වී ඇති බව ඔබට දැකිය හැක. මෙය function call ලෙස හැඳින්වේ. එයින් සිදුවන්නේ print function එක ක්‍රියාත්මක කරවීමයි. මෙවිට print තුල ඇති පරිගණක කේතයන් ක්‍රියාත්මක වීම සිදුවේ. මෙය සිදුවන ආකාරය නැරඹීම සඳහා ඔබ, ඉහත වැඩසටහන debug කිරීම ආරම්භකර, Visual C++ හි **step in** සහ **step over** යන ක්‍රියාකාරීත්වයන් මගින් වැඩසටහන

ක්‍රියාත්මක වන අනුපිලිවෙල පරීක්ෂා කල යුතුය. (පොතෙහි **"Debug කිරීම සහ Errors හඳුනාගැනීම"** යන කොටස කියවීමෙන් මෙය සිදු කරණ ආකාරය ඔබට උගත හැකිය). එවිට main තුලින් ආරම්භවන වැඩසටහන ක්‍රියාත්මක වීමේ ලක්ෂය main තුල ඇති print හි function call එක වෙත පැමිණ එතැනින් print තුලට ඇතුල්වන බව ඔබ දකිනු ඇත. වැඩසටහන ක්‍රියාත්මක වීමේ ලක්ෂය print තුල ඇති කේතය මතින් ගමන් කිරීමෙන් පසු එය නැවත main තුලට ආපසු ඇතුල් වෙනු ඇත. මෙවිට Terminal window එක දෙස ඔබ බැලුවහොත්, එය තුල `"Hi! This is my 1st function."` යන්න print වී තිබෙනු ඔබ දකිනු ඇත. මේ අනුව ඉහත පෙනෙන පරිදි, ඔබ විසින් සාදන ලද functions වලට ඇතුල් වීමේ ප්‍රධාන දොරටුව වන්නේ main function එකයි.

ඉහතදී, function එකක් පරිගණක වැඩසටහනක් තුල මූලික ලෙස ක්‍රියාත්මක වන්නේ කෙසේදැයි දැක්වුණා. දැන්, functions යනු මොනවාදැයි වඩාත් ලංව බලමු.

ඉහත ඔබ දුටු පරිදි function එකක භාවිතයේ අවස්ථා 3 ක් ඇත.

1. **Function declaration**
2. **Function definition**
3. **Function call** – මෙමගින් සිදුවන්නේ වැඩසටහනක කිසියම් ස්ථානයක ඇති function එකක්, එම වැඩසටහනෙහි තවත් ස්ථානයක සිට (ඉහත වැඩසටහනෙහි දී නම් main හි සිට) ක්‍රියාත්මක කරවීමයි.

Function එකක් කොටස් 3 කින් සමන්විතය. පහත දැක්වෙන්නේ එම කොටස් 3 යි.

1. Return type.
2. Function name.
3. Parameter list.

Function එකක් කැල්කියුලේටර් යන්ත්‍රයකට සමාන කල හැක. ඔබ කැල්කියුලේටරයකට දත්ත ඇතුල් කල විට (input) එය ඔබ ඇතුල් කල දත්තයන් වෙනස්කම් වලට භාජනය කර, ආපසු ඔබට දත්තයක් ලබා දෙයි (output). Function එකක් මගින් සිදුවන්නේද මෙයයි. එය පිටතින් දත්ත ලබාගෙන, එම දත්ත වෙනස්කම් වලට භාජනය කර, එමගින් ප්‍රතිඵලයක් නිපදවා, ආපසු දත්තයන් පිට කරයි. ඉහත කොටස් තුනෙන් පලමු වැන්න වන Return type යන්නෙන්, Function එකෙන් පිටකරණ මෙම ප්‍රතිඵල දත්තයේ දත්ත වර්ගය දක්වයි. දෙවැනි කොටසින් දක්වන්නේ function එකෙහි නම බව ඔබට පැහැදිලියි. තෙවැනි කොටස වන Parameter list යන්නෙන් දැක්වෙන්නේ function එක, input එක ලෙස එය තුලට ලබා ගන්නා දත්තයන්ය. නමුත් Parameter list එකක් හෝ Return type එකක් නොමැති හෙවත්, input එකක් හෝ output එකක් නොමැති functions ද සෑදිය හැකියි. මේ වර්ගයේ functions මගින් කිසිම දත්තයක් තමා තුලට ලබා නොගන්නා අතර කිසිම ප්‍රතිඵලයක් පිටතට ලබා නොදේ. අප ඉහතින් සලකා බැලූ අවසන් වැඩසටහනෙහි print function එක මෙවැන්නකි. එය සිදුකල එකම දෙය වූයේ වචන පේලියක් console window එක මත print කිරීම පමණි.

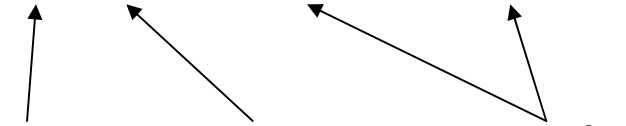
1. Function declaration

Function declaration එකක් මගින් සිදුවන්නේ නව function එකක් compiler මෘදුකාංගයට හඳුන්වාදීම බව ඔබ ඉහත ස්ථානයකදී දුටුවා. එසේනම් දැන් අප ඉහත Return type, Function name, Parameter list යන කොටස් තුනෙන් ම සමන්විත වූ function declaration එකක් ලියන්නේ කෙසේ දැයි බලමු. පහත දැක්වෙන්නේ එවැනි function declaration එකකට උදාහරණයකි:

```
=====================================================

            int add(int num1, int num2);

=====================================================
```

පහත දැක්වෙන්නේ ඉහත add function එකෙහි declaration එක, ඉහතින් ඔබ දුටු කොටස් වලට බෙදා ඇති ආකාරයයි.

```
int add(int num1, int num2)
```



මෙම function එක output එක ලෙස ලබා දෙන දත්තයෙහි වර්ගය int ය. මෙය Return type යන්නෙන් දැක්වේ. එමෙන්ම එය num1 හා num2 යන int variables යුගල input එක ලෙස ලබා ගනියි. මෙය parameter list යන්නෙන් දැක්වේ. Parameter list ලෙස අප මෙතැන දී සඳහන් කර ඇත්තේ variables 2 ක් පමණක් නමුත් ඔබට කැමති ඕනෑම variables ගණනකින් යුත් parameter lists සහිත functions සෑදිය හැකියි.

2. Function definition

ඉහත function declaration එකට අදාල function definition එක පහත දැක්වේ:

```
============================================================

      int add(int num1, int num2)          // function head
      {                                    //
            int addition = num1 + num2;    // function body
            return addition;               //
      }

============================================================
```

add හි function declaration එකින් add හි definition එක වෙනස් වන්නේ කෙසේද? පලමුව එයට සඟල වරහන් යුගලක් එකතු වී ඇත. සඟල වරහන් වි්වෘත වන ස්ථානයෙන් දැක්වෙන්නේ add function එකෙහි ආරම්භක ස්ථානයයි. නැතහොත් function එක බලපැවැත්වීමට පටන්ගන්නා ස්ථානයයි. සඟල වරහන් වැසෙන ස්ථානයෙන් function එකෙහි බල ප්‍රදේශය අවසන්ව යයි. Function එක තුල අඩංගු විය යුතු සියල්ල මෙම වරහන් අතර අන්තර්ගත විය යුතුය. මෙම වරහන් අතර කොටස function body යනුවෙන් හැඳින්වේ. මෙහි function declaration එකට සමාන ලෙස ලියා ඇති ඉහලම කොටස function head ලෙස හැඳින්වේ. Function definition එකෙහි function head එක අනිවාර්යයෙන්ම return type එකින්ද parameter list එකින්ද function declaration එකට සමාන විය යුතුය.

Function definition එක තුල සිදුවන්නේ කුමක්ද? මුලින් addition නමින් variable එකක් සාදා ඇති අතර එයට parameter list එකෙහි ඇති variable යුගලෙහි අගයන් ගේ එකතුව ආදේශ කර ඇත. ඉන්පසු return නැමැති keyword එක ඔබට දක්නට ලැබේ. return හි කාර්යය නම්, එය ඉදිරියේ ඇති variable එකෙහි අගය function එකින් පිටතට ලබා දීමය. එනම් function එකෙන් පිටතට දත්ත ප්‍රථදානය කරන්නේ return keyword එක මිිගිනි. return ක්‍රියාත්මක වීමත් සමඟම function එකෙහි ක්‍රියාවද නිමා වී යයි.

නමුත් දැන් function එකෙන් return වන අගයට සිදුවන්නේ කුමක්ද? මෙයට පිලිතුර තව මොහොතකින් ඔබට ලැබෙනු ඇත.

3. Function call

පෙරදීද සඳහන් වූ පරිදි function එකක් ක්‍රියාත්මක වීම ආරම්භ වන්නේ function call මගිනි. දැන් අප ඉහත add function එකට main තුල සිට call කරණ ආකාරය බලමු:

```
=========================================================

            void main()
            {
              int result = add(10,2);
              cout << result << endl;
            }

=========================================================
```

add හි function call එක, ඔබට පෙරදී හමුවූ print function එකෙහි function call එකින් වෙනස් මගක් ගෙන ඇති ආකාරය පෙනේද? මෙම call කිරීම සිදුවන්නේ expression එකක් තුලදීය. මෙයින් සිදුවන්නේ add මගින් return කරණු ලබන අගය result නැමැති variable එකට ආදේශ වීමයි. function call එක සිදුවෙන ආකාරය බලන්න. add ඉදිරියේ ඇති වරහන් තුල සංඛ්‍යා යුගලක් අන්තර්ගතය. මෙය නම් add function එක තුලට දත්ත ඇතුල් කිරීමේ මාර්ගයයි. මෙය parameter passing යනුවෙන් හැඳින්වේ. මෙම සංඛ්‍යා යුගල function definition එකෙහි num1 හා num2 යන parameters වෙත ආදේශ වෙයි. එවිට add ක්‍රියාත්මක වීමෙන්, එම අගයන් යුගලෙහි ඓක්‍යය වන 12 අගය addition variable එක තුල ගබඩා වෙයි. දැන් return මගින් 12 අගය function එකින් පිට කෙරේ. එම අගය main තුල ඇති result variable එක ලබා ගනියි. අවසානයේදී එම අගය print වේ.

ඔබ ඉහත දුටු function declaration, definition සහ call යන අවස්ථාවන්, එක වැඩසටහනක් ලෙස පහතින් පෙලගස්වා ඇත:

```
=========================================================

#include<iostream.h>

int add(int num1, int num2);

void main()
{
      int result = add(10,2);
      cout << result << endl;
}

int add(int num1, int num2)
{
      int addition = num1 + num2;
      return addition;
}

=========================================================
```

6. Arrays

පරිගණක වැඩසටහන් වලදී, කිසියම් දත්ත පෙලක් සමග කටයුතු කිරීමට සිදුවන අවස්ථාවන් මතුවේ. උදාහරණයක් ලෙස සතියේ දින 7 හිදී වාතයේ ආර්ද්‍රතාව සම්බන්ධ පරිගණක වැඩසටහනක් ලිවීමට අවශ්‍ය නම්, එම ආර්ද්‍රතා අගයන් 7 ගබඩා කරගැනීමට සිදුවෙනු ඇත. මෙය කිරීම සඳහා පහත පරිදි variables 7 ක් සෑදීමට සිදුවෙනු ඇත.

```
=========================================================

float humidMon = 3.3, humidTue = 3.12, humidWed = 2.99,
      humidThu = 3.01, humidFri = 3.00, humidSat = 2.98,
      humidSun = 2.91;

=========================================================
```

දැන් මෙම අගයන් ප්‍රාවිච්චියට ගැනීමට අවශ්‍ය වූ විට මෙම variables එකින් එක වෙන වෙනම 7 වරකදී භාවිත කලයුතුය. ඔබට පෙනෙන පරිදි මේ ආකාරයේ එකම ගණයේ දත්ත පෙලක් සමග කටයුතු කිරීමේදී variables සංඛ්‍යාවේ ඇතිවන ඉහලයාම නිසා ඔබට ඒවා භාවිතා කිරීමේ අපහසුතාවයන්ට මුහුණදීමට සිදුවේ. මෙය වඩාත් පැහැදිලිවන්නේ ඔබට කටයුතු කිරීමට සිදුවෙන දත්ත ප්‍රමාණය ඉහල යනවිටයි. උදාහරණයක් ලෙස ඉහත සඳහන් වැඩසටහනෙහි සතියේ දින 7 හි පමණක් නොව වසරේ දින 365 හිම ආර්ද්‍රතාවන් ගබඩා කරගැනීමට සිදුවූයේයැයි සිතන්න. එවිට කටයුතු ඔබේ පාලනයෙන් තොරව යාම ඇරඹේ. කිසිවෙකුට variables 365 ක් වැනි විශාල සංඛ්‍යාවක් සෑදීමට හෝ එවැනි variables සංඛ්‍යාවක් සමග වෙනවෙනම කටයුතු කිරීමට හැකියාවක් නැති බව ඔබට වැටහෙනු ඇත. එසේනම් අප මෙවැනි විශාල දත්ත සංඛ්‍යාවන් පාලනය කරන්නේ කෙසේද? අප **Arrays** ප්‍රාවිච්චි කරන්නේ මේ සඳහාය.

Array එකක් යනු එක් පොදු කරුණක් හා සම්බන්ධ දත්ත පෙලක් එක්කොට නිපදවනු ලැබූ ඒකකයකි. වෙලඳ සැලක සතියේ දින හත තුලදී, එක් එක් දිනයේ ලැබුණු ආදායම, සංයුක්ත තැටියක ඇති ගීත වල නම් ලැයිස්තුවක්, ක්‍රිඩකයෙක් කිසියම් ක්‍රීඩාවක් කල වාර ගණනේදී ලබාගත් ලකුණු වල අගයයන්, වැනි දත්ත Arrays තුල ගබඩා කලහැකි දේ සඳහා උදාහරණයන්ය. Array එකක් තුල ගබඩා කර ඇති සෑම දත්තයකටම ස්ථානයක් හෙවත් **index number** එකක් ඇත. ඔබට Array එකක් තුලට අලුත් දත්ත ඇතුල් කල හැකියි (උදාහරණයක් ලෙස ආහාර වට්ටෝරුවකට තවත් එක් ආහාරයක නමක් එක් කිරීම සලකන්න). තිබෙන දත්ත වෙනස් කලහැකියි (නම් ලැයිස්තුවක කිසියම් ස්ථානයක ඇති නමක් වෙනස් කිරීම සලකන්න). කිසියම් දත්තයක් ඉවත් කර දැමිය හැකියි (ඔබේ මිතුරන්ගේ e-mail ලිපින ලැයිස්තුවෙන් එක් අයෙකුගේ ලිපිනයක් ඉවත් කර දැමීම සලකන්න). මේ සෑමදෙයක්ම සිදුකිරීමට ඔබ ඉහත සඳහන් කල, අදාල දත්තයේ index number එක දැනගත යුතුය. Index number එක මගින් සිදුකරන්නේ අදාල දත්තය සමග Array එක මාර්ගයෙන් සම්බන්ධයක් ඇති කිරීමයි. Arrays සමග කටයුතු කිරීම බොහෝවිට පහසුයි. එසේනම්, Arrays පිලිබඳ තාක්ෂණික කරුණු මෙතැන් සිට සලකා බැලීම අරඹමු.

Arrays පිලිබඳව කථා කිරීමේදී ප්‍රධාන අවස්ථා 3ක් සලකා බැලිය යුතුයි:

1. Arrays නිපදවීම

පහත දැක්වෙන්නේ array එකක් සෑදීමේදී හෙවත් define කිරීමේදී භාවිත කරණ ව්‍යාකරණයයි.

```
=========================================================
```

data type array name [length] = { data 0, data 1, data 2, ... }

==========

ඉහත ඔබට පෙනෙන පරිදි array එකක් define කිරීමේදීද පලමුව යෙදෙන්නේ එහි අඩංගු වන දත්ත වලට ගැලපෙන data type එකය. දෙවනුව array එකෙහි නම යෙදේ. තෙවනුවට හමුවන කොටු වරහන් වැදගත්ය. මෙම කොටු වරහන් තුල යෙදිය යුත්තේ array එකෙහි length එක හෙවත් එහි ධාරිතාවයි. Array එකක ධාරිතාව යනු එම array එකෙහි ඇති දත්ත සංඛ්‍යාවයි. ඉහතදී දැක්වූ සතියේ දින 7 හි ආර්ද්‍රතාව පිලිබඳ උදාහරණයේදී එම array එකෙහි ධාරිතාව හෙවත් length එකෙහි අගය වන්නේ 7 යි. මක්නිසාදයත් array එක තුල ආර්ද්‍රතා අගයන් 7 ක් අන්තර්ගත වන බැවිනි. මෙනිසා එහිදී එම array එක නිර්මාණය කිරීමේදී වරහන් තුල යෙදිය යුත්තේ 7 අගය ය. මෙසේ අප array length එක සඳහන් කලයුත්තේ, compiler මෘදුකාංගය විසින්, සෑදිය යුතු array එකෙහි ධාරිතාව දැනගතයුතු නිසාය. ධාරිතාව දැනගත් විට, compiler මෘදුකාංගයට array එකට අවශ්‍ය වන මතක පරිමාව RAM තුලින් නිවැරදිව වෙන්කර ගත හැකිය (අවශ්‍යවන මුලු RAM මතක ධාරිතාව සොයාගන්නේ array එකෙහි අඩංගු වන දත්ත වර්ගයේ විශාලත්වය හෙවත් දත්ත වර්ගයේ bit පරිමාව, array එකෙහි අඩංගු දත්ත සංඛ්‍යාවෙන් වැඩිකිරීමෙනි. ආර්ද්‍රතා උදාහරණයේදී ඇත්තේ int වර්ගයේ දත්ත 7 ක් නිසා int හි විශාලත්වය වන 4 bit අගය 7 න් වැඩිකල යුතුය. එවිට එම array එකෙහි විශාලත්වය හෙවත් size එක ලෙස 28 bit අගය ලැබේ. මෙම array size යන්න array එකෙහි length එක සමග පටලවා නොගන්න. array length යනු array එකෙහි ඇති දත්ත සංඛ්‍යාව වන අතර size යනු එම array එක RAM තුලින් වෙන්කරගන්නා මතක ප්‍රමාණයයි. මීලගට " = " ලකුණ යෙදීම මගින්, අවශ්‍ය වන දත්තයන් array එක තුලට ආදේශ කර, array එක define කිරීම සිදුවේ. මෙහිදී සගල වරහන් ({}) භාවිතා කර ඇති බවට ඔබට දැකිය හැකිය. සඟල වරහන් පාවිච්චි කිරීමට සිදුවන්නේ මෙහිදී අප කටයුතු කරන්නේ දත්ත එකකට වඩා වැඩි සංඛ්‍යාවක් සමග වීම හේතුවෙනි. මෙම සගල වරහන් තුල අප array එකෙහි ගබඩා කල යුතු දත්ත අන්තර්ගත කරමු. ඉහත සටහන දෙස බලන විට ඔබට දැකගත හැකි තවත් දෙයක් වන්නේ සඟල වරහන් තුල අඩංගු දත්ත ආරම්භ වන්නේ 0 න් බවයි (data 0). ඉන්පසු ඒවා data 1, data2 ලෙස ඉහලට වැඩෙයි. වෙනත් වචන වලින් කියතහොත් **array එකක index එක ආරම්භ වන්නේ 0 වෙනි - 1 න් නොවේ**. මේ අනුව array එකක පලමු දත්තයෙහි index number එක 0 වන අතර දෙවැන්නෙහි index number එක 1 වේ. තෙවැන්න 2 වේ. Array එකක length එක හෙවත් array එකෙහි අඩංගු කල හැකි දත්ත සංඛ්‍යාව සොයාගත හැකි ක්‍රමයක් නම් එහි ඇති අවසන් දත්තයට හිමි index number එකට 1ක් එකතු කිරීමයි.

Array length = Last index number + 1

ඒ අනුව, අවසන් දත්තයේ index number එක 10 වන array එකක length එක 11 ක් වේ, එනම් එහි දත්තයන් 11 ක් ගබඩා කල හැකිවෙන අතර එම array එක define කිරීමේදී ඔබ එහි කොටු වරහන් තුලට යෙදිය යුතු අගය 11 වනු ඇත.

ඉහත කී කරුණ තවත් අයුරකින් කිවහොත් : **Length එක 10 වන Array එකක උපරිම index number එක 9 වන අතර එහි ගබඩා කල හැකි උපරිම දත්ත සංඛ්‍යාව 10 කි**.

ඉහත array එකෙහි සගල වරහන් තුල ඇති දත්ත එකින් එක වෙන් කර ඇත්තේ කොමා වලින් බව නිරීක්ෂණය කරන්න. Array එකක අගයන් 2 ක් අතර කොමා යෙදීමට සෑමවිටම මතක තබා ගන්න.

Array එකක අඩංගු දත්ත සහ ඒවායේ memory addresses

Array එකක දත්ත අඩංගු වන්නේ එක් පෙලකට බවද ඒවා එකක් පිටුපස තවෙකක් වශයෙන් සංවිධානයව ඇති බවද ඔබ ඉහතදී දුටුවා. මෙම දත්ත, එකකට පසුව තවෙකක් වශයෙන් define කරන ලද variables පෙලකට සමානවේ. නමුත් array එක මගින් මෙම variable පෙල එක් ඒකකයකට ගොනු කර ඇති නිසා එම ඕනෑම variable එකකට array එකෙහි index number එක මගින් ලගා වීමට හැකියාව සැලසී ඇත. Array එකක් තුල ඇති variables පෙලක් RAM මතකය තුල ඇත්තේද එක් පෙලක් වශයෙන් එකිනෙකට සමීපයෙනුයි. මේනිසාම එම දත්තයන්ගේ memory addresses දැක්වෙන hexadecimal සංඛ්‍යාවන්ද එකිනෙකට බොහෝ සේ සමානවේ. (මෙම දත්ත පෙල සමාන වන්නේ සැබෑ ලෝකයේ ඇති නිවාස පෙලකටය. එකිනෙකට ලංව ඉදිකර ඇති නිවාස වල ලිපිනයන් වෙනස් වන්නේ සුලු වශයෙනි). එක් පෙලකට ඇති දත්ත හෙවත් variables දෙකක memory addresses එකිනෙකින් වෙනස් වන්නේ එම variables අයත් data type එකෙහි විශාලත්වයට සමාන අගයකිනි. උදාහරණයක් ලෙස array එකක් තුල එක ලීගින් ඇති num1 සහ num2 නැමැති int variables දෙකක් සලකන්න. num1 හි memory address එක දක්වන hexadecimal සංඛ්‍යාව

0012FF70 යන අගය යැයි සිතන්න. int දත්ත වර්ගයේ variable එකක විශාලත්වය 4 bit නිසා num2 variable එකෙහි address එක num1 හි address එකින් වෙනස් වන්නේ 4 කිනි. එනම්, එහි address එක විය යුත්තේ, 0012FF70 අගයට 4ක් එක් කිරීමෙන් ලැබෙන 0012FF74 යන අගයයි. මෙලෙසින්ම, array එකෙහි ඊලඟට num3 නමින් තවත් variable එකක් ඇතිනම්, එහි address එක වියයුත්තේ 0012FF74 අගයට 4 ක් එක් කිරීමෙන් ලැබෙන 0012FF78 යන සංඛ්‍යාවයි.

2. Array එකක අන්තර්ගත අගයන් ලබාගැනීම

Array එකක ඇති කිසියම් අගයක් ලබාගැනීමට ප්‍රථමයෙන්, එම අගයෙහි index number එක දැනගත යුතුය. Index number එක දන්නා විට int array එකකින් අගයක් ලබාගැනීමට භාවිතා කරණ ව්‍යාකරණ විධිය පහත දැක්වේ:

```
================================================

int value = arrayname[index number];

================================================
```

ඉහත, value නැමැති int variable එකට array එකෙහි index number නමින් දැක්වෙන ස්ථානයේ ඇති අගය ආදේශ වේ. Index number එක ඇතුල් කල යුත්තේ array එකෙහි නමට ඉදිරියේ ඇති කොටු වරහන් තුල බව නිරීක්ෂණය කරන්න.

3. Array එකකට අගයන් ඇතුල් කිරීම

මෙය ඉහත ක්‍රියාවට විරුද්ධ අවස්ථාවයි. පහතින් එය දැක්වේ:

```
================================================

int value = 10;
arrayname[index number] = value;

================================================
```

ඉහතදී, value හි අගය array එකෙහි index number යනුවෙන් දැක්වෙන ස්ථානයට ආදේශවේ.

Arrays සම්බන්ධයෙන් ඉහත දැක්වූ ක්‍රියාවන් 2 ක ප්‍රායෝගිකව පෙන්වාදෙන පරිගණක වැඩසටහනක් පහත දැක්වේ:

```
================================================

#include<iostream.h>

void main()
{
      int numbers[10];

      for(int x=0; x<10; x++){
            numbers[x] = x;
      }

      for(int y=0; y<10; y++){
            cout<< numbers[y] << endl;
      }
```

```
}
```

==========

වැඩසටහනෙහි පලමුව number නැමැති ධාරිතාව 10 ක් වන array එකක් define කර ඇත. මේනිසා මෙම array එකතුල ගබඩා කල හැකි දත්ත වල උපරිම index number එක වන්නේ 9 බව ඔබට පෙනේ. එසේවන්නේ පෙර කියූ පරිදි array index එක ආරම්භවන්නේ 0 සිට නිසාය. ආදේශයේදී for තුල index number එක ලෙස භාවිතා කර ඇත්තේ x හි අගයයි. ආදේශවන්නේද x හි අගයයි:

```
numbers[x] = x;
```

for loop එක ක්‍රියාත්මක වීමේදී array එකට 0 සිට 9 දක්වා අගයන් 10 ආදේශවේ. index 0 ස්ථානයට ආදේශවන්නේ 0 යි. index 1 ස්ථානයට ආදේශවන්නේ 1 යි. අවසානයේ index 9 ස්ථානයට ආදේශවන්නේ 9 යි:

```
numbers[0] = 0;
numbers[1] = 1;
....
....
numbers[9] = 9;
```

දෙවැනි for loop එක index number එක ලෙස y භාවිතා කර array එකෙහි ඇති දත්ත ලබාගෙනිමින් ඒවා console window එක මත print කරයි. වැඩසටහන ක්‍රියාත්මක කලවිට ලැබෙන්නේ පහත දැක්වෙන ආකාරයේ output එකකි.

==========

```
0
1
2
3
4
5
6
7
8
9
Press any key to continue
```

==========

ඉහත ලෙස index numbers මගින් array එකක අන්තර්ගත දත්ත සමග කටයුතු කිරීම **“Array indexing”** ලෙස හැදින්වේ.

ඉහත වැඩසටහනෙහි අගයන් ආදේශය for loop එකක් භාවිත නොකර සෘජුව සඟල වරහන් භාවිතයෙන් සිදුකලහොත් එය පහත ආකාරය ගනියි:

==========

```
#include<iostream.h>

void main()
{
      int numbers[10] = {0,1,2,3,4,5,6,7,8,9};

      for(int y=0; y<10; y++){
            cout<< numbers[y] << endl;
      }

}
```

==========

char arrays

char array එකක් යනු characters වලින් සෑදුනු array එකක්ය. මේගැන විශේෂයෙන් කථා කිරීමේ හේතුව නම් char arrays වල විශේෂ ප්‍රයෝජනයක් තිබීමයි. පෙරදී, strings භාවිතාවන්නේ characters සමූහයකින් සෑදි character පේලි, නැතිනම් වාක්‍යයන් ගබඩා කිරීමට බව ඔබ දුටුවා. char array එකක් යනු මෙම string data type එකට ආදේශකයකි. නිවැරදිවම කිවහොත් string එකක් ඇත්තෙන්ම සෑදී ඇත්තේ char array එකක් මගිනි. නමුත් මෙය සිදුවන්නේ අපට නොපෙනෙන ලෙසිනි. උදාහරණයක් ලෙස ඔබ පහත දැක්වෙන string එක නිර්මාණය කලා යැයි සිතන්න.

```
string message = “System error!”;
```

ඉහත message string එක යනු පහත දැක්වෙන message char array එකම වනු ඇත:

```
char message[13] = {‘S’, ‘y’, ‘s’, ‘t’, ‘e’, ‘m’, ‘ ’, ‘e’, ‘r’, ‘r’, ‘o’, ‘r’,
‘!’};
```

Array එකෙහි “System” හා “error!” යන වචන බණ්ඩ යුගල අතර හිඩැස තබා ඇත්තේ හිස් character එකක් මගින් බව නිරීක්ෂණය කරන්න. නමුත් char array එකක් සෑදීමට, වඩා පහසු, නිවැරදි ක්‍රමයක් ඇත. පහත දැක්වෙන්නේ මෙයයි:

```
char message[] = "System error!";
```

එහිදී සගල වරහන් ඉවත්වී ඇති අතර, වෙන වෙනම characters පෙලක් වෙනුවට සෘජුවම text string එකක් array එකට ආදේශ කර ඇත. තවද array එකෙහි ධාරිතාව දක්වන කොටු වරහන් තුල සංඛ්‍යාවක් සඳහන්ව නොමැත. එනම්, මෙලෙස char array එකක් සෑදීමේදී අප එහි ධාරිතාව සඳහන් කලයුතු නැත. char arrays සම්බන්ධ තවත් ලක්ෂණයක් තිබේ. අප System error! යන්න array එකට ආදේශ කල විට, එම text string එක අවසානවන තැනට එක් ස්ථානයකට ඔබ්බෙන් ඉබේම හිස් character එකක් ආදේශවේ. එනම් ‘ ! ’ ඇති ස්ථානයට ඔබ්බෙන් මෙම හිස් character එක array එක තුල තැන්පත්වේ. මෙහි ප්‍රයෝජනය නම්, එමගින් array එකතුල ඇති text string එක නිමවන ස්ථානය හඳුනාගැනීමට පහසු වීමයි. මෙම හිස් character එක හැඳින්වෙන්නේ **“null character”** නැතිනම් **“null terminator”** යන නමිනි. මෙය හඳුනාගැනීමේ ලකුණ ‘ \0 ’ වේ.

නමුත් ඔබට ඇතැම්විට නැගෙන ප්‍රශ්ණයක් වනු ඇත්තේ, string සහ char arrays යනුවෙන් සමාන අකාරයේ data type දෙකක් සෑදීමට හේතුවුනේ කුමක්ද යන්නයි. C++ යනු C පරිගණක භාශාවේ කුඩා සොහොයුරාය. එනම්, C++ නිපදවා ඇත්තේ C භාශාව පදනම වශයෙන් තබාගෙනය. char arrays යනු මුල් C නිර්මාපකයා විසින් text strings ගබඩා කිරීම පිණිස නිර්මාණය කරන ලද data type එකය. නමුත් char arrays හි භාවිතයේ ඇති අපහසු බව දුටු පසු කාලීන C++ නිර්මාපකයා විසින් string යන data type එක text strings සමග කටයුතු කිරීම පහසු කිරීම සඳහා හඳුන්වා දුන්නේය. නමුත් ඔහු C තුල තිබූ char arrays භාවිතය C++ තුලද තිබෙන්නට හැරියේය.

පහත වැඩසටහනින් දැක්වෙන්නේ char array එකක් සාදන ආකාරය හා එහි null character එක භාවිතා කර ඇති ආකාරයයි:

==========

```
// char array.cpp

#include <iostream.h>

void main()
{
```

```
  char message[] = "Thank you!";
  int i = 0;
  while(message[i] != '\0')
  {
    cout << message[i];
    i++;
  }
  cout<<endl;
}
```

==========

ඉහත වැඩසටහනෙහි “Thank you” යන අගයෙන් යුත් char array එකක් සාදා ඇති අතර එහි ඇති text string එක while loop එකක් භාවිතා කර array indexing මගින් print කර ඇත. while මගින් text string එකෙහි අවසානය සඳහා පරීක්ෂා කර ඇති ආකාරය බලන්න:

```
while(message[i] != '\0')
```

message array එකතුල යම් ස්ථානයකදී null terminator එක හමුවුන විට, loop එක බිඳිගොස් “Thank you!” යන්න print කරමින් වැඩසටහන නිමාවට පත්වීම සිදුවන බව ඔබට පෙනෙනු ඇත. මෙලෙස null terminator එක මගින් char array එකක් අවසන් වන ස්ථානය සොයා ගත හැකිය.

7. Pointers

අප variables භාවිතා කර පරිගණක වැඩසටහන් ලියන ආකාරය ඉහතදී සලකා බැලුවෙමු. variables භාවිතා කරන්නේ පරිගණක මතකයේ ඇති කිසියම් දත්තයක් නියෝජනය කිරීම සඳහා බව එහිදී දුටිමු. pointers භාවිතා කරන්නේද මේ කාරණයටමය. නමුත් එක් වැදගත් වෙනසක් pointers සහ variables අතර ඇත.

මෙම වෙනස්කම කුමක්දැයි හඳුනාගන්නට නම් variables හැසිරෙන ස්වභාවය පැහැදිලි කරගත යුතුය. variables හි ස්වභාවය පැහැදිලි කරගත හැකි හොඳම ආකාරයක් නම්, තම argument එක ලෙස variable එකක් ලබාගන්නා function එකක් පිලිබඳව සලකා බැලීමයි. පහත දැක්වෙන වැඩසටහන දෙස බලන්න:

==========

```
//usingVariables.cpp

#include<iostream.h>

void change(int dat);

void main()
{
      int data = 100;
```

```
	change(data);
	cout << data << endl;
}

void change(int dat)
{
	dat = 200;
}
```

==========

ඉහත වැඩසටහනෙහි දැක්වෙන change නැමැති function එකින් සිදුවන්නේ, එය argument එක ලෙස ලබාගන්නා variable එකෙහි අගය වෙනස් කර වෙනත් අගයක් එයට ආදේශ කිරීමයි. පලමුව, main function එක තුල අගය 100 ක් වන data නැමැති variable එකක් තනා ඇති අතර මෙම variable එක change තුලට pass කර ඇත. දැන්, change function එක මගින් එම pass කල අගය 200 බවට වෙනස් කරන බව ඔබට දැකිය හැකියි. ඔබ මෙම වැඩසටහන ක්‍රියාත්මක කලහොත් main හි cout මගින් print කරන්නේ 100 බව දැකගත හැක. එනම්, change function එක, data මගින් එයට pass කල අගය වෙනසකට ලක් කලා යැයි කියා, data හි අගය වෙනස් වූයේ නැත. මෙයට හේතුව දැකගැනීමට නම්, function එකක් තුලට variable එකක් pass කරන විට සිදුවන්නේ කුමක්දැයි සොයාබැලිය යුතුය.

Function එකක් තුලට කිසියම් variable එකක් pass කරන විට සිදුවන පලමුවන දෙය නම් RAM මතකය තුල ඇති එම variable එකෙහි **අනුපිටපතක්** RAM තුල නිර්මාණය වීමයි. ඉන්පසු, function එක තුලට අගය pass වීම සිදුවේ. නමුත් අගය ලෙස pass වන්නේ ඉහත නිපදවුනු **මුල් දත්තයේ අනුපිටපතයි**. - **සැබෑ මුල් දත්තය නොවේ**. මෙනිසා function එක එම දත්තයේ අගය වෙනස් කලද වෙනසට ලක්වන්නේ එම අනුපිටපතෙහි අගය පමණයි. ඉහත වැඩසටහනෙන් ලබාදෙන ප්‍රතිඵලයට හේතුව ඉහත කියූ ක්‍රියාවයි. change function එක ලබාගන්නේ data variable එක නොව එහි අනුපිටපතකි. මෙම අනුපිටපත නම් dat variable එකයි. මේනිසා එය මගින් වෙනස් කරන්නේ dat හි අගයයි. data variable එකෙහි අගය වෙනසකට භාජනය නොවී 100 ලෙසම පවතියි.

ඔබ ඉහත කියවූ කරුණින් variables සම්බන්ධ ප්‍රධාන ලක්ෂණයක් අනාවරණය වේ. එය නම් variables මගින් පරිගණක දත්ත වලට සපයන ආරක්ෂාවයි. ඔබ මොහොතකට පෙර දුටු පරිදි, variables පමණක් භාවිතයට ගන්නා function එකකට පරිගණකයේ ඇති දත්ත සමග සෘජුව කටයුතු කිරීමේ හැකියාවක් නොමැත. එයට සැමවිටම ක්‍රියාකිරීමට සිදුවන්නේ දත්ත වල අනුපිටපත් සමගය. මේනිසා function එකකට පරිගණකය තුල ඇති සැබෑ දත්ත සමග සිතුවක් කිරීමට නොහැකිය. මෙම දත්ත සමග “සිතුවක් කිරීමට” ඇති නොහැකියාව නිසා variables භාවිතයෙන් පමණක් කිසිවෙතුට virus සහ worms වැනි වැඩසටහන් නිර්මාණය කිරීමට නොහැකිවී ඇත. (virus සහ worms යනු, අවසරයකින් තොරව කිසියම් කෙනෙකුට අයත් පරිගණකයක ඇති දත්ත වෙනස් කිරීමේ, මකාදැමීමේ හෝ වෙනත් කිසියම් අනවසර පරිගණක ක්‍රියාවක් සිදුකිරීමේ හැකියාව ඇති පරිගණක වැඩසටහන්ය). නමුත් කවරෙකු හෝ variables පමණක් භාවිතයෙන් පරිගණක වැඩසටහන් ලියයි නම් ඔහුට පරිගණකයේ දත්ත සමග සෘජු සම්බන්ධතාවක් පැවැත්වීමට variables මගින් ඉඩ නොසැලසීම හේතුවෙන් ඔහුට C++ වැනි පරිගණක භාශාවකින් නියම ප්‍රයෝජනය ගැනීමට නොහැකිව යයි.

ඉහත සඳහන්, Variables සම්බන්ධයෙන් ඇති මෙම සීමිත බවෙන් මිදීම සඳහා C++ තුල ඇති ඒකකය වන්නේ **pointer** ය**.** Pointers මගින් ඔබට variables මගින් ඉඩ නොලැබෙන සියල්ල කිරීමේ අවස්ථාව ලබාදේ. එමෙන්ම pointers හේතුවෙන්, variables මගින් පරිගණක දත්ත සඳහා යෙදෙන ආරක්ෂක කපොලු ඉවත්වී යයි.

අප මෙතැන් සිට pointers හැසිරෙන ස්වභාවය වටහාගැනීමට උත්සාහ ගනිමු.

Pointers වල ක්‍රියාව නම් පරිගණක දත්ත වෙත සෘජුව සම්බන්ධවීම බව ඔබ මොහොතකට පෙර කියවූවා. සැබෑ ලෝකයේ දී කිසියම් පුද්ගලයෙකු සමග කිසියම් සම්බන්ධතාවක් පැවැත්වීමට නම් ලොව තුල ඔහු සිටින ස්ථානය ප්‍රථමයෙන් සොයාගත යුතු ය. මෙසේම, පරිගණක දත්ත හා සම්බන්ධතාවක් පැවැත්වීමටද, පලමුව අප එම දත්ත RAM මතකය තුල පිහිටි ස්ථාන සොයාගත යුතුය. නමුත් මෙම ස්ථාන අප සොයාගන්නේ කෙසේද?

මෙම පොතෙහි මුල් භාගයේදී, පරිගණකයේ දත්ත අඩංගු වන්නේ RAM තුල බවද, එහි අඩංගු දත්ත වලට ලිපිනයන් හෙවත් memory addresses ඇති බවද සඳහන් විණි. (මෙය ඔබේ මතකයෙන් ගිලිහී ඇත්නම් කරුණාකර නැවත “RAM” නැමැති කොටස කියවන්න). කිසියම් පුද්ගලයකු ජීවත්වන ස්ථානය සොයාගැනීමට ඔහුගේ තැපැල් ලිපිනය පාවිච්චියට ගතහැකි සේ පරිගණක දත්ත ඇති ස්ථාන සොයාගැනීමටද ඉහත කියූ

ඒවායේ “මතක ලිපින” හෙවත් memory addresses භාවිතා කල හැකිය. පරිගණක වැඩසටහනකදී, කිසියම් variable එකක් මගින් නියෝජනය කෙරෙන දත්තයක memory address එක ලබාගැනීම ඉතා පහසුයි. Pointers සම්බන්ධව කරුණු 2ක් සලකා බලමින් අප මේ ගැන කථා කරමු.

1. පරිගණක දත්තයක memory address එක ලබාගැනීම

Variable එකක නමට පෙර ‘&’ ලකුණ යෙදීමෙන් ඔබට එම variable එක නියෝජනය කරන දත්තයේ ලිපිනය ලැබේ. පහත දැක්වෙන්නේ මෙයයි:

```
=======================================================

#include<iostream.h>

void main()
{

  int data = 100;
  cout<< &data << endl;

}

=======================================================
```

ඉහත, cout විධානය අඩංගු පේළියෙහි data හි නමට ඉදිරියෙන් ‘&’ ලකුණ යොදා ඇත. එයින් ලැබෙන්නේ data හි memory address එක හෙවත් data නැමැති variable එක, RAM තුල පිහිටි ස්ථානයේ ලිපිනයයි. ඔබ මෙම වැඩසටහන ක්‍රියාත්මක කලොත් පහත වැනි දහසයේ පාදයේ හෙවත් hexadecimal ඍණයට අයත් සංඛ්‍යාවක් print වෙනු දක්නට ලැබේවි. (ඔබට ලැබෙන්නේ වෙනස් අගයක් වියහැකිය. නමුත් මෙහිදී නිරීක්ෂණය කලයුතු දෙය නම් මෙම memory address එකෙහි ස්වභාවයයි. එය හැමවිටම hexadecimal වර්ගයේ සංඛ්‍යාවකි.)

```
=========

0012FF7C

=========
```

2. Pointer එකක් declare කිරීම සහ memory address එකක් ආදේශ කිරීම මගින් එම pointer එක define කිරීම.

Variable එකක් තුල ගබඩා කල හැක්කේ කිසියම් අගයක් වූ සේම, pointer එකක ගබඩා කල හැක්කේ memory address එකකි. අප “දත්ත වර්ගයන්” නැමැති පරිච්ඡේදයේදී කථාකල ආකාරයට, පරිගණක දත්තයන්, int, char වැනි දත්ත වර්ග වලට බෙදෙන නිසා ඒවායේ addresses ගබඩා කිරීමට තැනිය යුත්තේද අදාල දත්ත වර්ගයට අයත් pointers ය. මේනිසා අපට int pointer, char pointer, string pointer ආදිය පිලිබඳ කථා කිරීමට සිදුවේ. Pointer එකක් නිර්මාණයට හෙවත් declare කිරීම සඳහා වන කේතයද එක් කරුණකින් හැරෙන්නට variable declaration අවස්ථාවේදී යෙදෙන කේතයට බොහෝ සමානයි. වෙනස නම් pointer declaration හිදී pointer එකෙහි නමට ඉදිරියෙන් ‘ * ’ (asterisk) සලකුණ යෙදීමට අවශ්‍ය වීමයි. පහතින් දැක්වෙන්නේ මෙලෙස int වර්ගයේ pointer එකක් declare කර ඇති ආකාරයයි:

```
=======================================================
```

```
int *p;
```

==========

මේ අනුව, p යනු int pointer එකක් ලෙස හැඳින්වේ.

දැන් අප p pointer එක declare කර ඇති නිසා මීලඟට එය define කල යුතුය (එනම්, pointer එකට අගයක් ආදේශ කොට එහි පැවැත්ම පරිගණක මතකය තුල ස්ථිර කල යුතුය). Pointers තුල ගබඩා කලහැක්කේ memory addresses වීම නිසා pointer defenition අවස්ථාවේදී අප එයට ආදේශ කලයුත්තේ memory address එකක් වේ. පහතින් දැක්වෙන්නේ මෙයයි:

==========

```
int *p;

int dat = 120;
p = &dat;
```

==========

ඉහත, dat නමින් int variable එකක් define කර ඇති අතර තුන්වැනි පෙලෙහිදී '&' භාවිතා කිරීමෙන් dat හි address එක p pointer එකට ආදේශ කර ඇත. මෙහිදී ඔබ නිරීක්ෂණය කල යුතු ප්‍රධාන දෙයක් නම් ආදේශයේදී pointer එක (p) ඉදිරියෙන් ' * ' සලකුණ නොයෙදෙන බවය. Pointer එක හුදෙක් p ලෙස භාවිතා කර ඇත. මේ අනුව, pointer එකක් define කරන විට එයට ' * ' ලකුණ යෙදෙන්නේ නැත.

දැන් p තුල address එකක් ගබඩා වී ඇති නිසා අපට එම address එකට අදාල දත්තය සමග p pointer එක මාර්ගයෙන් ඍජුව සම්බන්ධ විය හැක. එනම් මේ අවස්ථාවේදී, p pointer එක, dat සමග dat හි memory address එක මාර්ගයෙන් ඍජුව සම්බන්ධවී ඇත. දැන් අපට p මගින්, dat හි අගය ලබාගැනීම වැනි දේ සිදුකල හැක. පහත දැක්වෙන්නේ එය සිදුකරන ආකාරයයි:

==========

```
int *p;

int dat = 120;
p = &dat;

int number;
number = *p;
cout << number << endl;
```

==========

ඉහත වැඩසටහනේදී, number නැමැති int variable එකක් නිමවා ඇති අතර, `number = *p` යන කේතය මගින් සිදුවන්නේ number තුලට dat හි අගය ඇතුල් වීමයි. මෙහිදී p pointer එක, dat වෙනුවට ආදේශකයක් ලෙස ක්‍රියාකරයි. එනම්, `number = *p` යන කේතය වෙනුවට `number = dat` යන්න යෙදුවහොත් සිදුවන්නේද එකම දෙය වන number වෙත dat හි අගය ආදේශ වීමයි. මෙය සිදුවන්නේ p තුල ගබඩාවී ඇති dat හි memory address එක නිසාය. Pointer එකෙහි කාර්යය නම් එය තුල ඇති මෙම address එක භාවිතා කොට එම address එක හිමි දත්තය සමග සම්බන්ධයක් ඇති කිරීමයි (හෙවත් point කිරීමයි. Pointer එකට එම නම ලැබී ඇත්තේ මේ හේතුවෙනි). මෙහිදී නැවත, p භාවිතා වන්නේ ' * ' (asterisk) ලකුණ සමග බව ඔබට පෙනෙනු ඇත. `number = *p` යන කේතයේ *p මගින් dat හි අගය ලබා දෙයි. මේනිසා දැන් number තුල ඇත්තේද cout මගින් print වන්නේද 120 අගයයි.

ඉහත දැක්වූ pointer එක declare කිරීම, pointer එකට memory address එකක් ආදේශ කර define කිරීම, සහ pointer එක මගින් අදාල අගයන් ලබාගැනීම යන අවස්ථා තුනෙන්, pointer එක සමග ' * ' (asterisk) ලකුණ භාවිතා වන්නේ පලමු හා තෙවන අවස්ථා වලදී පමණක් බව ඔබට දැකිය හැක. Pointer definition හි ආදේශයේදී ' * ' සලකුණ භාවිතා වන්නේ නැත.

දැන්, මේ වනවිට ඕනෑම ආරම්භකයෙකුට මෙන්ම ඔබටද හැගෙනු ඇත්තේ මෙම pointers භාවිතයේදී යොදාගන්නා ව්‍යාකරණ(syntax) මඳින් මඳ සංකීර්ණ බවට පත්වන බවයි. මේනිසා මෙම ව්‍යාකරණ සිහිතබා ගැනීමේ ක්‍රමයක් වශයෙන්, පහත, pointer syntax දැක්වෙන සටහන ඔබට ඉදිරිපත් කෙරේ. එම සටහන කොලයක ලියා ඔබේ පරිගණකය අසල බිත්තියේ අලවා ගන්න. එහි ඇති ව්‍යාකරණ ඔබේ ඇසට හුරුවන තුරු එයට බිත්තියේ එල්ලී සිටීමට ඉඩ හරින්න.

```
int i = 10, j; // int variables

int *ip;        // int pointer

ip = &i;  // int variable එකෙහි address එක int pointer එකට ආදේශ
          // කිරීම

j = *ip;  // int pointer එකෙහි අඩංගු address එක හිමි int variable
          //   එකෙහි අගය (j තුලට) ලබාගැනීම

*ip = 20; // int pointer එකෙහි අඩංගු address එක හිමි int variable
          //   එකෙහි (i හි) අගය (20 බවට) වෙනස් කිරීම
```

ඉහත දැක්වූයේ pointers හා සම්බන්ධ ව්‍යාකරණ වලින් කොටසක් පමණි. ඉතිරි කරුණු මේ අවස්ථාවේදී අප සාකච්චා කිරීමට නොයමු.

මේ වනවිට අප විමසා බලා අවසන් කලේ pointers භාවිතා කර දත්ත වෙත ඍජුව ලගාවීමේ මූලික ක්‍රියාවලියයි. නමුත් pointers හි නියම ක්‍රියාකාරීත්වය දැකීමට නම් අප pointers භාවිතා කර ලියනු ලැබූ function එකක් කවරාකාරදැයි පරීක්ෂා කර බැලිය යුතුය. මේ සඳහා, අප අවසානයට සලකා බැලූ change නැමැති function එක අඩංගු usingVariables යනුවෙන් හැඳින්වුණු වැඩසටහන pointers භාවිතයෙන් නැවත සකසා පහතින් ඉදිරිපත් කර ඇත:

```
=========================================================

//usingPointers.cpp

#include<iostream.h>

void change(int *dat);

void main()
{
      int data = 100;
      change(&data);
      cout << data << endl;
}

void change(int *dat)
{
      *dat =  200;
}

=========================================================
```

ඉහත usingPointers නැමැති වැඩසටහන පිලිබඳ කථා කිරීමට පලමුව, අප pointers භාවිතා නොකර ලියන ලද usingVariables වැඩසටහනෙහි සිදුවූයේ කුමක්දැයි ඉක්මනින් නැවත විමසා බලමු. කරුණාකර දැන් එම වැඩසටහන ඇති පිටුවට ආපසු ගොස් නැවතවරක් එය පිලිබඳ මතකය අලුත් කරගන්න.

එහිදී change නැමැති function එක data හි අගය තම argument එක ලෙස ලබාගන්නා අතර එහි අගය වෙනස් කරයි. නමුත් Function එක තුලට pass වන්නේ data හි අනුපිටපතක් පමණක් නිසා මුල් data variable එකෙහි අගය වෙනස්වීමකට ලක් නොවේ. මෙනිසා main තුල ඇති cout විධානය මගින් print වන්නේ 100 අගයමයි. දැන් අප ඉහත දැක්වෙන usingPointers නැමැති වැඩසටහන දෙස නෙත් යොමු කලහොත්, එහිදී change function එක තම argument එක ලෙස ලබාගන්නේ variable එකක් නොව pointer එකක් බව ඔබට පෙනෙනු ඇති. මෙය ඔබට change හි function declaration එකෙහිදී change හි වරහන් තුල ඇති `int *dat` යන්නෙන් පැහැදිලිවේ.

```
==========================

void change(int *dat);

==========================
```

එනම්, අප change function එකට pass කලයුත්තේ memory address එකකි. මෙය සිදුකර ඇති ආකාරය main function එකතුල දැකිය හැක. පලමුව data නැමැති int variable එකක් සාදා ඇති අතර ඉන්පසු එහි address එක pass කරමින් change වෙත call කර ඇත.

```
==========================

change(&data);

==========================
```

change තුල සිදුවන්නේ කුමක්ද? change function එක, `*dat = 200` යන කේතය මගින්, dat තුල ඇති address එක හිමි data නැමැති variable එක වෙත ලගාවී එහි අගය තීරණාත්මක ලෙස වෙනස් කරයි. මෙය variable එකකට කල හැකි දෙයක් නොවන අතර pointer එකක හැසිරීමේ ප්‍රධාන ලක්ෂණය මෙයයි. ඒ අනුව අවසානයේදී cout මගින් data හි අගය print කරනු ඇත්තේ 100 නොව 200යි.

Arrays සඳහා pointers

Arrays යනු මොනවාදැයි ඔබ **“Arrays”** පරිච්ඡේදයේදී ඉගෙනගත්තා. ඔබ උගත් මෙම arrays සඳහා pointers සෑදීම පිලිබඳව දැන් සලකා බලමු.

කිසියම් array එකකට pointer එකක් සෑදුවිට එම pointer එක භාවිතා කොට array එකතුල ඇති සියලු දත්තයන් සමග සම්බන්ධ විය හැකියි. Array එකක් සඳහා pointers සෑදීම දෙආකාරයකට සිදුකල හැක. පහත දැක්වෙන්නේ මෙම ආකාරදෙක භාවිතාකොට, int array එකක් සඳහා pointers දෙකක් සාදා ඇති ආකාරයයි.

<u>Arrays සඳහා pointers සෑදීම</u>

```
=========================================

int numbers[5] = { 23, 32, 44, 68, 11 };
int* numberP1 = &numbers[0];  // (1)
int* numberP2 = numbers;      // (2)

=========================================
```

(1) අවස්ථාවේදී සිදුවන්නේ කුමක්දැයි ඔබට ඇතැම් විට වැටහෙනු ඇත. එහිදී numberP1 නැමැති pointer එක define කර ඇති අතර එයට numbers නම් array එකෙහි index 0 වන ස්ථානයේ ඇති දත්තයෙහි memory

address එක ආදේශ කර ඇත. numbers[0] යන්නෙන් index 0 වන දත්තය දක්වන අතර &numbers[0] යන්න මගින් එම දත්තයේ memory address එක ලබාදේ.

ඉහත දැක්වූයේ array pointer එකක් සෑදීමේ පලමු ක්‍රමයයි. එනම්, අදාල array එකෙහි දත්ත වර්ගයට අයත් pointer එකක් සාදා එයට array එක තුල ඇති පලමු දත්තයේ address එක එයට ආදේශ කිරීමය.

(2) අවස්ථාවේදී සිදුවන්නේ කුමක්ද? එහිදී, "&" සහ වරහන් ඉවත් කර ඇත. ඇත්තෙන්ම මෙහිදී සිදුවන්නේද (1) අවස්ථාවේදී සිදුවූ දෙයමයි. එනම් numberP2 යන pointer එකට numbers array එකෙහි පලමු දත්තයේ address එක ආදේශවේ. මෙය හුදෙක් (1) අවස්ථාවේ සිදුවන දෙයම සරලව කිරීමකි. මේනිසා මෙම දෙආකාරයෙන් ඔබ කැමැති ආකාරයක් array pointers සෑදීමේදී අනුගමනය කරන්න.

Array pointer එකක් ප්‍රයෝජනයට ගනිමින් array එකක් තුල ඇති දත්ත සමග කටයුතු කිරීම

arrays සඳහා pointers සාදන්නේ කෙසේදැයි ඔබට දැන් වැටහේ. දැන් අප සූදානම් වන්නේ එම සෑදූ pointers, arrays සම්බන්ධයෙන් භාවිතා කරන්නේ කෙසේදැයි සොයා බැලීමටයි.

ඔබ මොහොතකට පෙර array pointer එකක් නිපදවන අවස්ථාවේදී සිදුකලේ array එකෙහි 1වැනි දත්තයේ address එක අඩංගු pointer එකක් නිර්මාණය කිරීමය. දැන් ඔබ දැනගතයුතු වන්නේ මෙම pointer එක මගින්, array එකෙහි ඇති අනෙකුත් දත්ත ලබාගැනීම, ඒවා වෙනස් කම් වලට ලක් කිරීම ආදී දේ සිදුකරන ආකාරය පිලිබඳවයි. මුලින්ම අප array එකක 2 වැනි දත්තය හෙවත් index 1 ස්ථානයේ ඇති variable එක වෙත array pointer එකක් මගින් පිවිසෙන්නේ කෙසේදැයි බලමු. පහත දැක්වෙන්නේ int array එකක් සඳහා pointer එකක් සෑදූ අවස්ථාවකි:

```
================================================

      #include<iostream.h>

      void main()
      {

        int numbers[5] = { 23, 32, 44, 68, 11 };
        int* number = numbers;
        cout<< number << endl;

      }

================================================
```

ඔබ උගත් පරිදි දැන් number නැමැති pointer එකතුල අඩංගු වන්නේ numbers නැමැති array එකෙහි පලමු දත්තය (index 0) වන 23 අගය, RAM තුල පවතින ස්ථානයේ memory address එකයි. දැන් අපට අවශ්‍ය වන්නේ එම pointer එක මගින් array එකෙහි 2 වන දත්තය වෙත පිවිසීමටයි. මෙය සිදුකරන්නේ කෙසේද?
අප කලයුතු දෙය වන්නේ ඉහත 23 අගය RAM හි පවතින ස්ථානයට පසුව ඇති දත්තය වෙත pointer එක දිශාගත (point) කිරීමය. එවිට pointer එක 32 අගය වෙත සම්බන්ධවනු ඇත. Pointer එක නිවැරදි ස්ථානයට දිශාගත කිරීමට නම් අප කලයුතු එකම දෙය වන්නේ pointer එක තුල ඇති memory address එක, 32 අගය පවතින ස්ථානයේ address එකට සමාන කිරීමය. වෙනත් වචන වලින් කිවහොත් pointer එක ඊලග දත්තය වෙත "පැන්නිය" යුතුය.

Array එකක් තුල අඩංගු දත්ත, RAM මතකය තුල සංවිධානය ඇත්තේ එක පෙලක් ලෙස ලීග ලීගින් වීම හේතුවෙන් ("Arrays" පරිච්ඡේදයේ "Array එකක අඩංගු දත්ත සහ ඒවායේ memory addresses" යන කොටසෙහි මෙය විස්තර කෙරිණි), මෙම "පැන්නීම" සිදුකිරීම පහසු දෙයක් වී ඇත. ඔබ කලයුත්තේ pointer එකට 1ක් එකතු කිරීමයි. පහත මෙය සිදුකරන අයුරු දැක්වේ:

```
================================================

#include<iostream.h>

void main(){
```

```
	int numbers[5] = { 23, 32, 44, 68, 11 };
	int* number = numbers;

	number++;
	cout<< *number << endl;
}
```

==========

ඉහත, number pointer එකෙහි අගය 1කින් ඉහල දමා ඇති අතර, මේ හේතුවෙන් එය array එකෙහි 2 වැනි දත්තය වෙතට දිශාගතවනු ඇත. දැන් cout මගින් නිවැරදි ලෙසම 32 අගය print වෙනු වැඩසටහන ක්‍රියාත්මක කිරීමේදී ඔබට පෙනෙනු ඇත. ඔබට array එකෙහි මීලග දත්තය ලබාගැනීමට අවශ්‍ය නම් සිදුකල යුත්තේ number හි අගය තවත් 1කින් ඉහල දැමීමයි. එවිට එය තුල අන්තර්ගත වනු ඇත්තේ 44හි address එකයි. මෙලෙස ඔබට ලබාගැනීමට අවශ්‍ය දත්තය, array එකෙහි පිහිටි ස්ථානය අනුව pointer එකෙහි අගය ඉහල දැමීය යුතුයි.

ඉහත ඔබ දුටුවේ pointer arithmatics භාවිතා කරනු ලැබීමේ එක් අවස්ථාවකි. ඔබ එහිදී pointer එකක අගය වෙනස් කිරීමෙන් array එකක් තුල ඇති දත්ත කරා ලගාවෙන ආකාරය දුටුවා. දැන් අප pointer arithmatics භාවිතා කොට ඉහත පරිදි ලබාගන්නා ලද දත්තයන් සමග, සරල ගණිතමය ක්‍රියා සිදුකරන්නේ කෙසේදැයි පරීක්ෂා කර බලමු.

උදාහරණයක් ලෙස ඉහත numbers නැමැති array එකෙහි තෙවන දත්තයෙහි අගය 44 සිට 50 දක්වා ඉහල දැමීය යුතුයැයි සිතමු. මෙය සිදුකරන ආකාරය පහත ලෙස දැක්විය හැක:

==========

```
#include<iostream.h>

void main()
{

  int numbers[5] = { 23, 32, 44, 68, 11 };
  int* number = numbers;
  number+=2;

  cout<< *number << endl;

  (*number)+=6;

  cout<< *number << endl;

}
```

==========

ප්‍රථමයෙන් සිදුවන්නේ පෙර පරිදිම number array එක වෙත pointer එකක් නිපදවීම මගින් බව ඔබට පෙනේ. ඉන්පසු, pointer එකෙහි අගය 2 කින් ඉහල දමා ඇත්තේ එය array එකෙහි තෙවන දත්තය වෙත සම්බන්ධ කිරීම පිණිසය. මීලඟට එම දත්තයේ අගය වන 44 print වේ. දැන්, ඔබට හමුවන්නේ pointer arithmatics භාවිතා කිරීමෙන් ලියනු ලැබූ expression එකකි. එහි වරහන් භාවිතා වන බව ඔබට පෙනේ(expression එකකදී වරහන් භාවිතය පිලිබඳව **“Operators”** පරිච්ඡේදයේ සඳහන් විණි). මෙම expression එකෙහිදී සිදුවන්නේ කුමක්දැයි බලමු.

`*number+=6` යන ආකාරයට ඉහත expression එකෙහි වරහන් ඉවත් කොට ලිවූහොත්, precendence නියාමයන්ට අනුව පලමුව සිදුවිය යුත්තේ plus equal (+=) operator එක, pointer (*) operator එකට පෙර ක්‍රියාත්මක වීමය. * operator එක expression එකෙහි මුලින් තිබුනද එය ක්‍රියාත්මක විය යුතු වන්නේ += operator එකට පසුවය. ඒ අනුව, පලමුව number හි අගයට 6 ක් එකතුවනු ඇත. නමුත් ඔබට පෙනෙන පරිදි අපට අවශ්‍ය ගණිත කර්මය මෙය නොවේ. මෙයින් සිදුවන එකම දෙයනම් pointer එක array එකෙහි ස්ථාන 6

ක් ඉදිරියෙන් වන ස්ථානයට ගෙනයාමය. ඉන්පසු එම ස්ථානයේ ඇති අගය print වෙනු ඇත. දැන් expression එක වරහන් යොදා ලියූවිට සිදුවන දේ බලන්න. * සහ number යුගලය වරහන් තුල අන්තර්ගත කිරීම මගින් (*number), expression එකෙහි පලමුව සිදුවන දෙය ලෙස, number pointer එක වෙත මේ මොහොතේදී සම්බන්ධවී ඇති අගය ලබාගැනීම සිදුවේ. දැන් += operator එක ක්‍රියාත්මක වී එම ලබාගත් අගයට 6 ක් එකතුවේ. එනම් 44 වෙත 6 ක් එකතුවී array එකෙහි එම ස්ථානයේ අගය 50 ක් බවට පත්වේ. මෙය වැඩසටහනේ දෙවැනි cout විධානය මගින් තහවුරු වනු ඇත.

ඔබට ඉහත කරුණු තවදුරටත් පැහැදිලිවීම පිණිස පහත වැඩසටහනද ඉදිරිපත් කෙරේ. එය for loops භාවිතා කරමින්, arrays තුල අඩංගු වන දත්තය වලට ඇතුල්වෙමින් එම දත්ත වෙනස්කම් වලට භාජනය කරනු ලබනු ඔබට දැකියහැකි වනු ඇත:

```
=====================================================

#include<iostream.h>

void main()
{

  int Lotto[10] = { 21, 2, 61, 20, 86, 44, 37, 10, 0, 1 };
  int* number = Lotto;

  // 1st, print the array
  for(int a=0; a<10; a++){
  cout<< *number++ << endl;
  }
  cout<< "\n";

  // copy the array to a temporary array
  int temp[10];
  for(int b=0; b<10; b++){
        temp[b] = Lotto[b];
  }

  number = Lotto; // point the pointer back to the
                  // 1st element of the array

  // swap the data in the array
  int c = 9;
  for(int d=0; d<10; d++){
      Lotto[d] = temp[c];
      c--;
  }

  for(int e=0; e<10; e++){
  cout<< *number++ << endl;
  }

}

=====================================================
```

ඉහත වැඩසටහන ඔබ මෙතෙක් දුටු වැඩසටහන් වලට වඩා සුලු වශයෙන් සංකීර්ණ එකක් බව ඔබට පෙනෙනු ඇත. නමුත් මෙය පැහැදිලි කරගැනීම ඉතා පහසුයි. එහි පලමුවෙන්ම සිදුවන්නේ Lotto නමින් array එකක් නිර්මාණය කිරීම බව ඔබට පෙනේ. ඉන්පසු සුපුරුදු පරිදි එම array එක සඳහා pointer එකක් නිමවා ඇත. ඉන්පසු සිදුවන්නේ array එකෙහි ඇති අගයන් සියල්ල for loop එකක් මගින් print වීමය. අවසාන අගය print වූ පසු එය **newline character (\n)** එක මගින් Terminal window එකෙහි නව හිස් පේළියක් හෙවත් new line එකක් සාදයි. ඉන් අනතුරුව, Lotto array එකෙහි length එකට සමාන length එකක් ඇති temp නමින් අලුත් array එකක් සාදා ඇති අතර මෙම temp array එක තුලට Lotto array එකෙහි සියලු දත්ත for loop එකක් ආධාරයෙන් කොපි කරනු ලැබේ. අනතුරුව number pointer එක Lotto array එකෙහි පලමු දත්තය වෙත නැවතත් දිශාගත කරනු ලැබේ. මක්නිසාදයත් එම pointer එක දැන් දිශාගතවී තිබෙන්නේ array එකෙහි අවසාන දත්තයට හෙයිනි. මුලින්ම array එකෙහි දත්ත print කරන විටදී pointer එකෙහි අගය වෙනස්වූ බව ඔබට මතක ඇති. මීලගට විශේෂ දෙයක් සිදුවේ. මීලග for loop එකින් Lotto තුල ඇති දත්ත පෙලගැස්ම

කණපිට පෙරලයි. එනම්, මුලින් ඇති අගටත්, අග ඇති දත්ත මුලටත් ගෙනයාම සිදුවේ. මෙය සිදුවන්නේ ඉහත අප නිපදවූ temp array එකෙහි අධාරයෙනි. For loop එක තුල ඇති පලමු expression එකෙහි අඩංගු arrays වල කොටු වරහන් තුල ඇති අගයන් දෙස බැලූවිට ඔබට මෙය සිදුවන ආකාරය පැහැදිලි වනු ඇත. Lotto හි වරහන් තුලට d යොදා ඇති අතර, temp හි වරහන් තුලට c යොදා ඇත. d හි අගය for loop එක ක්‍රියාත්මක වීමත් සමග 9 දක්වා වැඩිවන අතර, c හි අගය 0 දක්වා අඩුවනු ඇත. temp තුල ඇත්තේ Lotto තුල ඇති අගයන්ම බව ඔබට මතක ඇති. මේ නිසා, Lotto තුල ඇති මුල් අගයන් temp තුල ඇති පසු අගයන් සමගද අග ඇති අගයන් මුල් අගයන් සමගද මාරුවීමෙන් Lotto තුල ඇති අගයන්ගේ පෙලගැස්ම කණපිට පෙරලෙනු ඇත. අවසානයේදී මෙම කණපිට හැරුණු අගයන් print වේ. මෙම වැඩසටහන ක්‍රියාත්මක කල විට පහත output එක ලැබෙනු ඇත:

```
==============================

21
2
61
20
86
44
37
10
0
1

1
0
10
37
44
86
20
61
2
21

==============================
```

මුලින් දිස්වන්නේ Lotto හි දත්ත මුලින් පිහිටි ආකාරය වන අතර ඉන්පසුව ඇත්තේ අගයන් කණපිට පෙරලී ඇති අයුරුයි.

8. Object Oriented Programming

(වස්තු මූලික ක්‍රමලේඛණය)

(Object Oriented Programming පිලිබඳව මෙම පරිච්ඡේදයෙහි එන්නේ ඉතා මූලික හැඳින්වීමක් පමණක් බව සලකන්න. එය පිලිබඳ සම්පූර්ණ විස්තර සලකා බැලීමට එක් පරිච්ඡේදයකින් නොහැකිය.)

Object oriented programming යනු කුමක්ද?

මෙම පොතෙහිදී මෙතෙක් ඔබ දුටු ක්‍රමලේඛණයන් අයත් වූයේ **procedural programming** ඝණයටයි. මෙවැනි පරිගණක ක්‍රමලේඛණයන් බොහෝවිට ක්‍රියාත්මක වන්නේ ක්‍රමලේඛණයේ ඉහල සිට පහලට, අඛණ්ඩ ආකාරයකිනුයි. එමෙන්ම, මෙතෙක් අප දුටු වැඩසටහන් වල පදනම වූයේ functions ය. මේවා procedural programming හි ලක්ෂණයි. දැන් අප සූදානම් වන්නේ මෙම මානයෙන් ඔබ්බට ගමන් කර නව දෘශ්‍යකෝණයකින් පරිගණක වැඩසටහන් ලිවීම දෙස බැලීමටය. මෙම නව දැක්ම හෙවත් නව ක්‍රමලේඛණ විධිය **Object Oriented Programming** නමින් හැඳින්වේ.

පරිගණක වැඩසටහනක් යනු ක්‍රමලේඛකයෙකු විසින් කිසියම් ගැටලුවකට ඉදිරිපත් කල විසිදුම ලෙස හැඳින්විය හැක. මෙම ගැටලු බොහෝවිට සැබෑ ලෝකයේ වස්තු මූලික කොටගෙන ඇතිවේ. උදාහරණයක් වශයෙන් කිසිවෙකු මෝටර් ධාවනය කිරීම පරිගණකය මගින් ඉගැන්වීම සඳහා virtual reality ඝණයේ පරිගණක මෘදුකාංගයක් නිර්මාණය කිරීමට සිතයි නම්, ඔහුට මොටර් රථය නැමැති වස්තුවද, ට්‍රැපික් සංඥා නැමැති වස්තුවද, පදිකයා යන වස්තුවද, වෙනත් මොටර්රථ ධාවනය කිරීම හා සම්බන්ධ නොයෙක් දේද තම වැඩසටහන තුල නිමවීමට සිදුවනු ඇති. එහිදී සිදුවන්නේ සැදිය යුතු පරිගණක වැඩසටහනට අදාල සැබෑ ලෝකයෙහි පවතින වස්තූන් එම පරිගණක වැඩසටහන තුල නිර්මාණය කිරීමයි. මෙලෙස සැබෑලෝකය තුල ඇති වස්තු පාදක කොටගෙන පරිගණක වැඩසටහන් නිර්මාණය කිරීම object oriented programming ලෙස මූලික වශයෙන් සැලකිය හැකිය.

Object oriented programming භාවිත කරන්නේ ඇයි?

Procedural programming හිදී ඔබට කිසියම් function එකක ඇති පරිගණක කේතයන් නැවත පාවිච්චි කිරීමට සිදුවන අවස්ථාවක් සලකන්න. නමුත් එය නැවත පාවිච්චි කිරීමේදී එම function එකෙහි ක්‍රියාකාරීත්වය මඳක් වෙනස් කිරීමට ඔබට අවශ්‍ය යැයඳ සිතන්න. මෙවිට ඔබ කරන්නේ කුමක්ද? ඔබ Function එක copy කර, එය වෙනත් ස්ථානයක Paste කර, එයට අවශ්‍ය වෙනස්කම් සිදුකර, එයට අලුත් නමක් දෙනු ඇත. මෙම ක්‍රියාව මගින් දෙආකාරයක පාඩු සිදුවේ. පලමුවැන්න නම් මෙම copy-paste කිරීමේදී ක්‍රමලේඛණය වඩ වඩා විශාල වීමය. තවද මෙයින් වැඩසටහන සැමතැනම **program bugs** හෙවත් වැරදි ලෙස ක්‍රමලේඛණය කර ඇති තැන් ඇතිවීමේ සම්භාවිතාව ඉහල යයි. නමුත් object oriented programming භාවිතා කිරීමෙන් මෙම copy-paste අවුල ලිහියයි. එහිදී ඔබට **Inheritance** නැමැති object oriented ක්‍රමවේදය මෙවැනි අවස්ථාවන් වලදී භාවිතා කල හැකිය. Inheritance යනු, පවතින පරිගණක කේතයන් copy-paste නොකර, මුල් කේතයන්ටද හානියක් සිදු නොකර, අවශ්‍ය වෙනස්කමඳ සහිතව යලි එම කේතයන් පාවිච්චියට ගැනීමේ විශිෂ්ඨ ක්‍රමෝපායකි.

තවද, object oriented programming මගින් පරිගණක ක්‍රමලේඛකයක් වඩා පැහැදිලි වන අතර, පසුව වැඩසටහන නවීකරණය කිරීමද වඩා පහසු දෙයක් බවට පත් වේ.

වස්තුවක් හෙවත් object එකක් යනු කුමක්ද?

සැබෑ ලෝකයේ වස්තුවක් ගත්විට එයට හිමි විශේෂ ලක්ෂණ ඇත. එමෙන්ම එම වස්තුවට හිමි ක්‍රියාකාරීත්වයන්ද ඇත. සැබෑ ලෝකයේ වස්තුවකට උදාහරණයක් ලෙස ගුවන්යානයක් සලකන්න. ගුවන්යානයකට නියමිත හැඩයක් ඇත. එයට වර්ණයක් ඇත. එය මගින් ගුවන් යානයකට විශේෂ වූ හඬක් නිපදවයි. මේවා ගුවන්යානය නැමැති වස්තුවෙහි ලක්ෂණය. අප මෙම ලක්ෂණ properties යනුවෙන් හඳුන්වමු. තවද, ගුවන්යානයකට නිශ්චිත ක්‍රියාකාරීත්වයන්ද ඇත. එයට තම වේගය පාලනය කලහැකිය. තමාගේ සිට පොලොවට ඇති දුර (altitude) පාලනය කලහැකිය. තම ගමන්මග වෙනස් කලහැකිය. මෙම ක්‍රියාකාරීත්වයන් අප ගුවන්යානයේ functions ලෙස හඳුන්වමු. C++ මගින් වස්තුවක් නිර්මාණය කිරීම යනු මෙම properties (ලක්ෂණ) සහ functions (ක්‍රියාකාරීත්වයන්) එක් කොට එක් ඒකකයක් නිමවීමය. C++ තුල වස්තුවක් නිර්මාණය කිරීමට අප **class** නැමැති keyword එක පාවිච්චි කරමු.

දැන් අප ඉහත සඳහන් ගුවන්යානය නැමැති වස්තුව C++ මගින් නිමවන්නේ කෙසේදැයි බලමු:

```
============================

class Plane
{

};

============================
```

ඉහත දැක්වූයේ ගුවන්යානය නැමැති object එක C++ මගින් නිපදවීම ආරම්භ කල හැකි ආකාරයය. එහිදී පලමුව class යන keyword එක යොදා ඇත. ඉන්පසු තනන වස්තුවෙහි නම(plane) සඳහන් කර ඇත. ඉන්පසුව function එකකදී මෙන් සගල වරහන් යොදා අවසානයේදී colon එකක් තබා ඇත. මෙලෙස object එකක් තැනීම ඔබගේම දත්ත වර්ගයක් නිර්මාණය කිරීම ලෙසද හැඳින්වේ. එනම් දැන් Plane යන්න int හෝ char මෙන් දත්ත වර්ගයකි. එය “user define data type” හෙවත් “පාවිච්චි කරන්නා විසින් නිමවූ දත්ත වර්ගයක් ලෙස හඳුන්වයි.

අප Plane යන object එක සෑදුවද, ඔබට පෙනෙන පරිදි ඒතුල කිසිවක් අඩංගු නොවේ. සැබෑ වස්තුවකට නම් එයට අයත් ලක්ෂණ ඇත. ගුවන්යානයකට ඇති property එකක් වන්නේ එහි වේගයයි. තවද, කිසියම් මොහොතකදී එය ගමන් කරන දිශාවද එහි ලක්ෂණයක් වනු ඇති. එයට පොලොවෙහි සිට ඇති දුර හෙවත් altitude එක තවත් property එකකි. මෙම properties අප Plane තුලට දැන් අන්තර්ගත කරන්නේ මෙසේයි:

```
============================

class Plane
{

  private:
     int Speed;
     int Direction;
     int Altitude;

};

============================
```

දැන් Plane සතු ක්‍රියාකාරීත්වයන් එයට අන්තර්ගත කරමු. ඒවානම්, එහි වේගය පාලනය කිරීමේ හැකියාව, එය ගමන් කරන දිශාව පාලනය කිරීමේ හැකියාව සහ එහි altitude එක පාලනය කිරීමේ හැකියාව යන ක්‍රියාකාරීත්වයන්ය:

```
==========================================

class Plane
{

  private:
     int Speed;
     int Direction;
     int Altitude;

  public:
     void ChangeSpeed(int spd);
     void ChangeDirection(int dir);
     void ChangeAltitude(int alt);

};

==========================================
```

දැන් අප Plane object එක define කර අවසානයි. සිදුවූයේ කුමක්ද?:

Procedural programming හිදී අප සිදුකලේ කිසියම් දෙයක් සිදුකිරීම සඳහා වෙන වෙන ම variables හා function define කිරීමයි. නමුත් මෙහිදී කිසියම් වස්තුවකට අයත් functions හා variables, එම වස්තුව තුල අන්තර්ගත කර ඇත. එනම්, අප Plane නමින් class එකක් හෙවත් object එකක් සාදා, එයට අයත් දත්ත ඒතුලට අඩංගු කර ඇත.

තවද ඔබ දකින අලුත් දෙයක් වන්නේ මෙම **private** හා **public** නැමැති keywords යුගලයි. මෙම keywords හැඳින්වීමට පොදුවේ **access control keywords** යන නම භාවිතවේ. private මගින් සිදුවන්නේ වැඩසටහනෙහි වෙන තැනක සිට ඍජුව, මෙම Plane object එකතුල ඇති variables භාවිතා කිරීමට හෝ වෙනස් කිරීමට ඇති හැකියාව නැති කිරීමයි. ඒවා මෙහිදී Plane object එකෙහි "පෞද්ගලික" දේ බවට පත්වේ. public keyword එකින් පවසන්නේ එම keyword එක යටතේ ඇති සියල්ල, සියල්ලට විවෘත බවයි. දැන්, මෙලෙස කිරීමේ ඇති අර්ථය කුමක්ද? Plane object එකට පිටතින් ඇති කිසිවකට, object එකතුල ඇති variables හිටි අඩියේ තමාට අවශ්‍ය පරිදි වෙනස් කිරීමට ඉඩනොදීමෙන් එම object එකෙහි සුරක්ෂිත බව තහවුරුවේ. නමුත් එම variables වෙනස් කිරීමට නීත්‍යානුකූල ක්‍රමයක්ද මෙහිදී Plane මගින් සපයා ඇත. එනම් ඒතුල ඇති functions වල ක්‍රියාකාරීත්වයයි. ChangeSpeed() මගින් එහි Speed variable එක වෙනස් කරනු ඇත. එමෙන්ම ChangeDirection() මගින් එහි Direction variable එකද ChangeAltitude() මගින් එහි Altitude variable එකද වෙනස් කරනු ඇත. මෙම functions කොතැනක සිට වුව පාවිච්චි කලහැකිය. මන්දයත් ඒවා declare කර ඇත්තේ public keyword එක යටතේ නිසාය. මෙය හරියටම ගුවන්යානයක අභ්‍යන්තර ක්‍රියාකාරීත්වයන් එහි නියමුවාට ඍජුව පාලනය කිරීමට නොහැකි බවට සමානය. ඔහුට හිටිඅඩියේ ගුවන්යානයට යාහැකි උපරිම වේග සීමාව වෙනස් කල නොහැකිය. එය සිදුකල හැක්කේ ගුවන්යානය නිපදවූ සමාගමෙහි කාර්මික ශිල්පීන් හටය. නියමුවාට කලහැක්කේ ගුවන්යානයේ cockpit එකෙහි හිඳගෙන, ඔහුට සපයා ඇති ලීවර හා බොත්තම් පාවිච්චි කිරීමෙන් එය පාලනය කිරීම පමණය. මෙහිදී, ගුවන්යානය නිපදවූ ශිල්පීන්ට අනුරූප වන්නේ Plane class එක තැනූ ඔබය. ගුවන් නියමුවාගේ කාර්යයට අනුරූප වන්නේ ඔබ Plane හි functions භාවිතයට ගන්නා අවස්ථාවය.

Access control keywords පිලිබඳව තව දුරටත් මඳක් ඉදිරියේදී සලකා බලමු.

දැන් අප ඉහත දැක්වූ functions, define කරමු**:**

```
==============================================
```

```
void Plane::ChangeSpeed(int spd)
{
     Speed = spd;
     cout<< Speed <<endl;
}

void Plane::ChangeDirection(int dir)
{
     Direction = dir;
     cout<< Direction <<endl;
}

void Plane::ChangeAltitude(int alt)
{
     Altitude = alt;
     cout<< Altitude <<endl;
}

================================================
```

Procedural programming හිදී function එකක් define කරන ආකාරයට වඩා මෙම function definitions වල විශාල වෙනසක් දැකිය හැකි නොවේ. වෙනසකට ඇත්තේ function name එකට මුලින් එම function එක අයත් object එකෙහි නම සඳහන් වී තිබීම පමණි. Object එකෙහි නමද function name එකද වෙන්වන්නේ **“ :: ”** සලකුණිනි. සාමාන්‍යයෙන් ඔබ Visual C++ තුල object name එක type කර ඉදිරියෙන් **::** සලකුණ type කලවිට එම object එකට අදාල variables හා functions වල නම් Visual C++ මගින් පෙන්නුම් කරනු ඇති. ඔබ කලයුත්තේ දිස්වන කුඩා menu එකින් ඔබට අවශ්‍ය item එක මත click කිරීමය.

ඉහත function definitions මගින් සිදුවන දේ ඔබට පැහැදිලි ඇති. ඒවා මගින් Plane object එකෙහි variables වල අගයයන්, pass කරනු ලබන arguments වල අගයන් බවට පත් කෙරේ. ඉන්පසු එම වෙනස්වූ අගයන් print වේ.

දැන් මෙහිදී සිදුවූ දේ යලි කෙටියෙන් කිවහොත්**:** Plane object එක මගින් එහි අඩංගු අභ්‍යන්තර දත්ත හෙවත් variables සෘජුව වෙනස් කිරීමේ අයිතිය භාහිර ලෝකයෙන් ඉවත් කර ඇති අතර, ඒවා නිසි පරිදි, නීත්‍යානුකූලව පමණක් වෙනස් කිරීමට functions define කර ඇත. C++ object එකකට ඇති මෙම දත්ත පිටත ලෝකයෙන් සැගවීමේ හැකියාව **Data Abstraction** ලෙස හැඳින්වේ. Object එකක් සතු දත්ත, class නැමැති එක් ඒකකයට ගොනු කිරීම **Encapsulation** ලෙස හැඳින්වේ.

Encapsulation හි ප්‍රධාන ප්‍රයෝජන 2ක් දැක්වේ**:**

(1) Modulraity - Object එකකට ස්වාධීන ඒකකයක් ලෙස ක්‍රියාකිරීමට ඇති හැකියාව. object oriented පරිගණක වැඩසටහනක object එකක් එම වැඩසටහනෙහිම අනෙක් objects වෙතින් වෙන්වූ ඒකකයක් ලෙස පැවතීමට හැකිවීම නිසා, අනෙක් objects මත රඳා සිටීමට එයට සිදු නොවේ - ගුවන්යානයක් ගුවන්තොටුපලක ඇති අනෙක් වස්තු වලින් වෙන්වූ දෙයකි.

(2) Information hiding – දත්ත සැගවීමේ හැකියාව. Object එකකට private හෙවත් පෞද්ගලික දත්ත තබාගැනීමට හැකි නිසා එම දත්ත අනෙක් objects වල සම්බන්ධ වීමකින් තොරව, ස්වාධීනව වෙනස් කරගත හැක. - ගුවන්යානයකට යාහැකි උපරිම වේගය එහි නිශ්පාදන සමාගම විසින් වෙනස් කිරීම, ගුවන් මගියා ගේ (ගුවන් මගියා නැමති object එකෙහි) හවුල් වීමකින් තොරව සිදුකල හැකිය. එහිදී යානයට යාහැකි උපරිම වේගය යානයේ පෞද්ගලික දත්තයක් වේ.

දැන් අප ඉහත කියූ private දත්ත වල හැසිරීම පිලිබඳව උදාහරණයක් පරීක්ෂා කරමු. මේ සඳහා අප ගුවන්යානයක ගුවන් නියමුවා දැක්වීමට Pilot යන object එක තනා එම object එක මගින් Plane object එක සම්බන්ධව කලහැකි දේ සහ කල නොහැකි දේ පිලිබඳව විමසමු.

පහත දැක්වෙන්නේ Pilot object එක නිමවා ඇති ආකාරයයි:

```
================================================

class Pilot
{

  public:
       void ChangePlaneSpeed(Plane p, int spd);
       void ChangePlaneDirection(Plane p,int dir);
       void ChangePlaneAltitude(Plane p, int alt);

};

================================================
```

මීලඟට Pilot object එකෙහි functions define කරමු**:**

```
================================================

void Pilot::ChangePlaneSpeed(Plane p, int spd)
{
      p.ChangeSpeed(spd);
}

void Pilot::ChangePlaneDirection(Plane p,
int dir)
{
      p.ChangeDirection(dir);
}

void Pilot::ChangePlaneAltitude(Plane p,
int alt)
{
      p.ChangeAltitude(alt);
}

================================================
```

ඉහත සෑම function එකක්ම තම argument එක ලෙස Plane object එකක් ලබාගන්නා බව ඔබට පෙනෙනු ඇති. මේ අනුව ඉහතින්ද කී පරිදි object එකක් යනු ඇත්තෙන්ම ක්‍රමලේඛකයා විසින් define කරන ලද data type එකක් වේ. Functions තුලදී, එම pass කරනු ලැබූ object එක පාවිච්චි කර ඇති ආකාරය බලන්න. එහිදී p යනු Plane object එකෙහි instance එකක් ලෙස හැඳින්වේ. එය, int i ලෙස variable එකක් declare කිරීමේදී, i යනු int හි instance එකක් යැයි කීම හා සමානය. Plane තුල ඇති functions වලට call කර ඇත්තේ p සහ function name එක අතර තිතක්(period) තැබීමෙනි. Visual C++ තුල, ඔබ object instance එක type කර, තිතක් type කල විට object එක තුල ඇති functions සහ properties VC++ මගින් දක්වනු ඇත.

මෙලෙස Pilot object එක මගින් Plane object එකතුල ඇති දත්ත, එතුල ඇති functions මාර්ගයෙන් වෙනස්කිරීමට හැකි බව ඔබට ඉහත පෙනෙන්නට ඇත. මේ අයුරින් පරිගණක වැඩසටහනක ඇති එක් object එකක සිට වෙනත් object එකක functions වලට call කිරීම එම objects අතර massages send කිරීම හෙවත් පණිවුඩ හුවමාරු කරගැනීම ලෙස හැඳින්වේ. Object oriented program එකක් යනු ඇත්තෙන්ම objects රැසක් අතර ඇති පණිවුඩ හුවමාරු කරගැනීමකි.

මේවන විට objects සම්බන්ධව කටයුතු කිරීමේදී භාවිතා වන සලකුණු 2 ක් ඔබ දුටුවා. ඒවානම්, **“ :: ”** සහ **“ . ”** යන යුගලයි. **“ :: ”** පාවිච්චි වන්නේ ඉහත දැක්වුණු පරිදි, declare කරන ලද object එකක් තුල ඇති

functions define කිරීමේදීය. **“ . ”** පාවිච්චි වන්නේ object instance එකක් මගින් එම object එක තුල ඇති function එකකට call කිරීමේදී හෝ object එක තුල ඇති variable එකක් ලබාගැනීමේදීය.

දැන් ඔබට main() තුල සිට Plane හි functions වලට, Pilot object එක මගින් පහත පරිදි call කල හැකිය. එහිදී ඔබ කලයුත්තේ පලමුව Pilot සහ Plane object දෙකෙහි instance දෙකක් සෑදීම වන අතර ඉන්පසුව **“ . ”** මගින් Pilot හි function වලට ඇතුල් විය යුතුය:

```
================================

void main()
{
  Pilot pt;
  Plane pl;
  pt.ChangePlaneSpeed(pl, 500);
}
```

පහත දැක්වෙන්නේ ඉහත ඔබ දුටු කේතයන් සියල්ල ගොනු කර සෑදූ සම්පූර්ණ වැඩසටහනයි. මෙම වැඩසටහන VC++ තුල type කර එය ක්‍රියාකරන ආකාරය බලන්න:

```
================================================

#include<iostream.h>

class Plane
{

  private:
      int Speed;
      int Direction;
      int Altitude;

  public:
      void ChangeSpeed(int spd);
      void ChangeDirection(int dir);
      void ChangeAltitude(int alt);

};


void Plane::ChangeSpeed(int spd)
{
     Speed = spd;
     cout<< Speed <<endl;
}

void Plane::ChangeDirection(int dir)
{
     Direction = dir;
     cout<< Direction <<endl;
}

void Plane::ChangeAltitude(int alt)
{
     Altitude = alt;
     cout<< Altitude <<endl;
}

class Pilot
{
```

```
  public:
      void ChangePlaneSpeed(Plane p, int spd);
      void ChangePlaneDirection(Plane p, int dir);
      void ChangePlaneAltitude(Plane p, int alt);
};

void Pilot::ChangePlaneSpeed(Plane p, int spd)
{
      p.ChangeSpeed(spd);
}

void Pilot::ChangePlaneDirection(Plane p, int dir)
{
      p.ChangeDirection(dir);
}

void Pilot::ChangePlaneAltitude(Plane p, int alt)
{
      p.ChangeAltitude(alt);
}

class FighterPlane:Plane
{

};

void main(){

  Pilot pt;
  Plane pl;
  pt.ChangePlaneSpeed(pl, 500);
}
```

==========

Inheritance

ඉහත ද සඳහන්වූ පරිදි, ඔබට කිසියම් පරිගණක කේතයන් රැසක් වෙනස්කම් සහිතව නැවත නැවත භාවිතා කිරීමට අවශ්‍ය නම් object oriented programming හි අන්තර්ගත inheritance නම් ක්‍රමවේදය යොදාගත හැක. අප මෙතෙක් දුටු ගුවන්යානා උදාහරණයම යොදාගෙන මෙය වටහාගැනීමට උත්සාහ කරමු.

උදාහරණයක් ලෙස ඔබට සාමාන්‍ය ගුවන්යානයකට අමතරව යුධ ගුවන්යානයක් ද තැනීමට අවශ්‍යයැයි සිතන්න. සාමාන්‍ය ගුවන්යානයකට මෙන්ම යුධ ගුවන්යානයකට ද පොදුවූ ලක්ෂණ මෙන්ම එකිනෙකින් වෙනස් වූ ලක්ෂණද ඇත. නමුත් යුධ ගුවන්යානයකට ඇත්තේද සාමාන්‍ය ගුවන්යානයක පදනමම වේ. මේනිසා ඔබට සාමාන්‍ය ගුවන්යානය පදනම් කොටගෙන යුධ ගුවන්යානයක් නිමවිය හැකිය. ඔබ කලයුත්තේ සාමාන්‍ය ගුවන්යානයක් ගෙන එයට අලුතින් අන්තර්ගත කලයුතු ලක්ෂණ එකතුකර යුධ ගුවන්යානය තැනීමයි. මෙලෙස එක් object එකක ලක්ෂණ ආවේණි කරමින් වෙනත් object එකක් නිර්මාණය කිරීම **Inheritance** ලෙස හැඳින්වේ. යුධ යානාව තැනීමේදී ඔබට සාමාන්‍ය යානයේ නොමැති මිසයිල විධිමේ හැකියාව එයට අන්තර්ගත කිරීමට අවශ්‍ය වනු ඇත. මෙම වෙනස මෙම යානා දෙක අතර ඇති ප්‍රධාන වෙනස ලෙස සලකමින් දැන් අප යුධ ගුවන්යානය තනන්නේ කෙසේදැයි බලමු:

==========

```
class FighterPlane: public Plane
{

};
```

==========

ඉහත දැක්වෙන්නේ යුධ ගුවන්යානය හෙවත් `FighterPlane` object එක, `Plane` object එකින් inherit හෙවත් ප්‍රවේණි කිරීමේ 1වන අධ්‍යරයි. Code එකෙහි දැක්වෙන පරිදි inherit කිරීමේදී භාවිත වන ව්‍යාකරණ පිලිවෙල මෙසේයි**:** මෙහිදී අප declare කරන්නේ නව object එකක් නිසා පලමුව යෙදිය යුත්තේ **class** keyword එකයි. දෙවනුව යෙදිය යුත්තේ නව object එකෙහි නමයි. ඉහත මෙය `FighterPlane` වේ. තෙවනුව, **" : "** සලකුණ යෙදිය යුතුය. මෙම සලකුණින් දැක්වෙන්නේ, එම සලකුණට පෙර ඇති object එකෙහි(`FighterPlane`) ලක්ෂණ සලකුණට පසු ඇති object එකින් (`Plane`)ආවේණිය(inherit) වියයුතු බවයි. හතරවනුව inherit කලයුතු පැරණි object එකෙහි නම, ඒ සඳහා භාවිත කල යුතු access type එක සමග යෙදිය යුතුය. ඉහතදී ආවේණි කලයුතු object එක වන්නේ `Plane` ය. Plane ට ඉදිරියෙන් එහි access type එක ලෙස public යොදන්නේ, plane හි තිබෙන සියලු දත්තයන් පාවිච්චි කිරීමේ අයිතිය FighterPlane class එකට තිබිය යුතු බව දැක්වීමටය. ඔබ එහිදී private keyword එක පාවාච්චි කලා නම් FighterPlane හට Plane හි ඇති functions පාවිච්චි කිරීමට නොහැකිවීයයි.

මෙහිදී මුල් class එක **"Base class"** ලෙසද, මුල් class එකින් ආවේණි කර සැදූ නව class එක **"Derived class"** ලෙසද object oriented programming හිදී හඳුන්වනු ලැබේ.

දැන් ඔබ object oriented programming හි භාවිත වන තවත් සලකුණක් වන **" : "** දුටුවා. මෙය හඳුන්වන්නේ "is based on sign" වශයෙනි. ඒඅනුව ඔබ මේ වනවිට මතකයේ තබාගතයුතු සලකුණු ගනන 3 කි**:**
(1) " :: " (2) " . " (3) " : " නැවතත් මෙම සලකුණු භාවිතා වන අවස්ථාවන් කෙටියෙන් සිහිකල හොත්, (1) භාවිතාවන්නේ object එකක ඇති functions define කරන විටයි. (2) භාවිතාවන්නේ object එකක් තුල ඇති function වලට call කරන විට හෝ variables ලබා ගන්නා විටයි. (3) භාවිතාවන්නේ මොහොතකට පෙර දුටු පරිදි පැරණි object එකක ආවේණියෙන් නව object එකක් තනන විටය.

මේ අනුව, FighterPlane object එකට Plane object එකට හිමි සියලුම දත්ත හිමිය. යුධ ගුවන්යානයට සාමාන්‍ය යානයට කල හැකි වේගය වෙනස් කිරීම, දිශාව වෙනස් කිරීම, altitude එක වෙනස් කිරීම ආදී සියල්ල කලහැකිය. ඒ අනුව ඔබට main() හි සිට, Plane තුල ඇති functions වලට, ඒවා FighterPlane තුල ඇත්තා සේ සලකා call කල හැකි වනු ඇත. එනම් මෙහිදී අප Plane තුල ඇති functions ම, FighterPlane තුලදී නැවත භාවිතයට ගනිමු. ඒ අනුව Inheritance භාවිතා කිරීමෙන් අපට පරිගණක කේතයන් copy-paste නොකර මෙසේ යලි භාවිතයට ගත හැකිය. නමුත් මෙය සාමාන්‍ය function එකකට නැවත නැවත call කිරීමක් නොවේ. අප මෙහිදී Plane තුල ඇති functions නැවත ප්‍රයෝජනයට ගත්තේ ඒවා FighterPlane ට අයත් අලුත්ම functions ලෙසයි. පහත දැක්වෙන්නේ මෙයයි:

```
====================================

void main(){

     FighterPlane fp;
     Pilot pl;
     pl.ChangePlaneSpeed(fp, 200);

}

====================================
```

ඉහතදී ඔබ දකින දෙයක් වන්නේ, **ChangePlaneSpeed** මගින් මෙහිදී ලබා ගන්නේ Plane object එකක් නොව FighterPlane object එකක් බවයි. නමුත් ChangePlaneSpeed මගින් ඉල්ලන්නේ Plane object එකක් නොවේද? Object oriented programming හි තවත් වාසියක් නම් මෙයයි. එනම්, ඔබට Base class object එකක් අවශ්‍ය තැනකට, Derived class object එකක් වුවද pass කල හැකිය. එසේනම් දැන් **ChangePlaneSpeed** තුලදී call වන්නේ FighterPlane තුල ඇති ChangeSpeed function එකකටය.

මෙය සිත්ගන්නා සුලු ලක්ෂණයකි. පැහැදිලිවම, FighterPlane තුල දැනට කිසිම function එකක් අත්තර්ගතව නොමැති බව ඔබට එහි declaration එක දෙස බැලීමෙන් පැහැදිලි වේ. මෙහිදී, main() තුල සිට

FighterPlane instance එකක් සාදා, Pilot object එකක මාර්ගයෙන් FighterPlane හි ChangeSpeed වෙත call කිරීම සිදුවේ. නමුත් FighterPlane තුල එවැනි function එකක් දක්නට නැත. එසේනම් එය Plane තුල ඇති ChangeSpeed වියයුතුද? ඇත්තෙන්ම දැන් එය Plane තුල ඇති ChangeSpeed නොවේ. එය FighterPlane මගින් Plane තුල සිට ආවේණික කරගත්, දැන් තමා සතු කරගෙන සිටින වෙනම function එකකි. ඒනිසා මෙහිදී ChangeSpeed මගින් සිදුවන්නේ Plane තුල ඇති Speed හි අගය වෙනස් කිරීම නොවේ. සිදුවන්නේ FighterPlane තුල ඇති Speed හි අගය වෙනස් කිරීමයි. Object oriented programming භාවිතයේ ඇති කාර්යක්ෂම බවට එක් සාධකයක් නම් මෙයයි. පෙර කියූ පරිදිම මෙලෙස Inheritance මගින් ඔබට පවතින දත්තයන් copy-paste කිරීමකින් තොරව නැවත නැවත පාවිච්චි කිරීමේ හැකියාව ලැබේ.

පහත දැක්වෙන්නේ ඉහත ඔබ මෙතෙක් දුටු කේතයන් ගොනු කොට තනනු ලැබූ සම්පූර්ණ වැඩසටහනයි. එය VC++ තුල type කර ක්‍රියාත්මක කර සිදුවන දේ නැවත පැහැදිලි කරගන්න:

```
================================================

#include<iostream.h>

class Plane
{

  private:
      int Speed;
      int Direction;
      int Altitude;

  public:
      void ChangeSpeed(int spd);
      void ChangeDirection(int dir);
      void ChangeAltitude(int alt);

};


void Plane::ChangeSpeed(int spd)
{
      Speed = spd;
      cout<< Speed <<endl;
}

void Plane::ChangeDirection(int dir)
{
      Direction = dir;
      cout<< Direction <<endl;
}

void Plane::ChangeAltitude(int alt)
{
      Altitude = alt;
      cout<< Altitude <<endl;
}

class Pilot
{
  public:
      void ChangePlaneSpeed(Plane p, int spd);
      void ChangePlaneDirection(Plane p, int dir);
      void ChangePlaneAltitude(Plane p, int alt);
};

void Pilot::ChangePlaneSpeed(Plane p, int spd)
```

```
{
     p.ChangeSpeed(spd);
}

void Pilot::ChangePlaneDirection(Plane p, int dir)
{
     p.ChangeDirection(dir);
}

void Pilot::ChangePlaneAltitude(Plane p, int alt)
{
     p.ChangeAltitude(alt);
}

class FighterPlane:public Plane
{

};

void main(){

  FighterPlane fp;
  Pilot pl;
  pl.ChangePlaneSpeed(fp, 200);

}
```

==========

නමුත් යුධ යානයට සාමාන්‍ය ගුවන් යානයට කලහැකි දේට අමතරව මිසයිල විඩිමේ අමතර හැකියාව ද ඇත. මෙම හැකියාව FighterPlane object එක වෙත ලබාදෙන ආකාරය දැන් බලමු:

==========

```
class FighterPlane:  public Plane
{
  public:
     void FireMissile();
};
```

==========

ඔබ කලයුත්තේ FighterPlane තුලට මිසයිල විදීමේ හැකියාව ලබාදෙන නව function එකක් ඉහත දැක්වෙන පරිදි ඇතුල් කිරීමය. පහත FireMissile function එක define කර ඇත:

==========

```
void FighterPlane::FireMissile(){

     cout<< "Missile fired!" << endl;

}
```

==========

එමෙන්ම, පහත පරිදි Pilot තුලද නව Function එකක් define කිරීම සුදුසුය:

==========

```
class Pilot
{
public:
     void ChangePlaneSpeed(Plane p, int spd);
     void ChangePlaneDirection(Plane p, int dir);
     void ChangePlaneAltitude(Plane p, int alt);
     void FireMissile();
};

void Pilot::FireMissile()
{
     FighterPlane fp;
     fp.FireMissile();
}
```

==========

දැන් ඔබට main() හි සිටීමින්, මිසයිලයක් මුදාහරින ලෙස යුධ යානයට පහත පරිදි කිවහැකිය:

==========

```
void main(){

     Pilot pl;
     pl.FireMissile();

}
```

==========

දැන් මුලු වැඩසටහන පහත පරිදිය:

==========

```
#include<iostream.h>

class Plane
{

private:
     int Speed;
     int Direction;
     int Altitude;

public:
     void ChangeSpeed(int spd);
     void ChangeDirection(int dir);
     void ChangeAltitude(int alt);

};


void Plane::ChangeSpeed(int spd)
{
```

```
     Speed = spd;
     cout<< Speed <<endl;
}

void Plane::ChangeDirection(int dir)
{
     Direction = dir;
     cout<< Direction <<endl;
}

void Plane::ChangeAltitude(int alt)
{
     Altitude = alt;
     cout<< Altitude <<endl;
}

class Pilot
{
public:
     void ChangePlaneSpeed(Plane p, int spd);
     void ChangePlaneDirection(Plane p, int dir);
     void ChangePlaneAltitude(Plane p, int alt);
     void FireMissile();
};

void Pilot::ChangePlaneSpeed(Plane p, int spd)
{
     p.ChangeSpeed(spd);
}

void Pilot::ChangePlaneDirection(Plane p,
int dir)
{
     p.ChangeDirection(dir);
}

void Pilot::ChangePlaneAltitude(Plane p, int alt)
{
     p.ChangeAltitude(alt);
}

class FighterPlane:public Plane
{
     public:
     void FireMissile();

};

void FighterPlane::FireMissile(){
     cout<< "Missile fired!" << endl;
}

void Pilot::FireMissile()
{
     FighterPlane fp;
     fp.FireMissile();
```

```
}


void main(){

     Pilot pl;
     pl.FireMissile();

}

============================
```

Access control - public, private, protected keywords
(Object එකක් තුල ඇති දත්ත වෙත ඇතුල්වීම පාලනය කිරීම)

කිසියම් object එකක් තුල ඇත්තේ variables හා function බව ඔබ දනියි. Object එකක් තුල ඇති මෙම දත්ත වලට, object එකින් පිටත සිට ලගාවීමේ හැකියාව පාලනය කිරීම C++ object එකක ඇති මූලික ලක්ෂණයකි. ඔබ ඉහත තැනකදී දුටු information hiding නැමැති සංකල්පය යථාර්ථයක් වන්නේ මෙම හැකියාව නිසාය. ඒ හැකියාව නම්, object එකකට පිටතින් ඇති යමකට, පාවිච්චි කිරීමට හැකියාව ඇත්තේ object එක තුල ඇති කවර දත්දයන්ද සහ පාවිච්චි කිරීමේ හැකියාව නැත්තේ කවර දත්තයන්ද යනුවෙන් තීරණය කිරීමයි. මේ සඳහාය public, private හා protected යන access control keywords භාවිතා වන්නේ. public හා private යන යුගලය පිලිබඳව මේ වන විට ඔබ යම් තරමක් කියවා ඇති නමුත් එහි ක්‍රියාකාරීත්වය අත්දැක නැත. පහත දැක්වෙන්නේ Plane තුල ඇති private දත්තයක් වන altitude හි අගය main() හි සිට සෘජුව වෙනස් කිරීමට ගත් උත්සාහයකි:

```
================================================
void main(){

     Plane p;
     p.Altitude = 1300;
}
================================================
```

ඔබ මෙම වෙනස්කම වැඩසටහනෙහි main() function එකට සිදුකර වැඩසටහන compile කිරීමට උත්සාහ කලහොත් ඔබට error එකක් ලැබෙනු ඇති අතර වැඩසටහන compile නොවනු ඇත. එම error එකින් කියවෙන දෙය නම් main() හි සිට (හෙවත් Plane object එකට පිටතින් සිට) Plane තුල ඇති private variable එකක් වන Altitude පාවිච්චි කිරීමට නොහැකි බවයි. අප මෙහිදී සිදු කිරීමට උත්සාහගත් දෙය, හරියටම කිසිවෙකු ගුවන්යානයක ඇති පාලක යතුරු භාවිතා නොකර එහි උන්නතාංශය පාලනය කිරීමට සිතීම වැනිය. නමුත් නිසැකවම එය ක්‍රියාවේ යෙදවීමට ඔහුට නොහැකි වෙයි. ගුවන්යානයේ උන්නතාංශය පාලනය කිරීමට හැක්කේ ගුවන්යානයේ ඇති නිසි යතුරු හෝ ලීවර පාවිච්චි කිරීමෙන් පමණක් විය යුතුය. එය අනුරූපී වන්නේ Plane තුල ඇති ChangeAlitutude function එකට කරන call කිරීමකටයි. ඉහත වැඩසටහනේදී මෙය සිදුකරන්නේ Pilot object එක මගිනි. ඔහු එය කරනු ලබන්නේ තමා තුල ඇති public function එක වන ChangePlaneAltitude මගිනි. මෙම ක්‍රමය සහ main() තුල සිට කෙලින්ම Plane හි functions සඳහා call කිරීම හැරෙන්නට Altitude හෝ Plane හි වෙනත් කිසිම private variable එකක් වෙනස් කිරීමේ වෙනත් කිසිම ක්‍රමයක් ඉහත වැඩසටහන තුල නොමැත. ඔබට Plane හි ඇති දත්ත වෙනස් කිරීම සඳහා අවසර දීමට අවශ්‍ය වන්නේ අතිශයින්ම Pilot object එකට පමණක්ම නම් ඔබ කල යුතු වන්නේ, Plane හි ඇති functions ඉවත් කර, Pilot හි functions වල සිට සෘජුව Plane හි දත්ත වෙනස් කල හැකි ආකාරයට වැඩසටහන සැකසීමය. එවිට main හි සිට Plane හි දත්තයන් වෙනස් කිරීමට නොහැකි වී යයි. මේ අනුව object oriented programming මගින් object එකක් තුල ඇති දත්ත වලට සැපයෙන ආරක්ෂාව ඔබට වැටහෙනු ඇත.

Object එකක් තුල “private:” යන්න යටතේ declare කර ඇති දත්තයන්ට object එකට පිටින් සිට ඇතුල්වීමට නොහැකි බවද “public:” යන්න යටතේ ඇති දත්ත වලට ඇතුල් වීමට සැමටම හැකිබවද ඉහත ඔබ දුටුවා. 3න් වැනි access control keyword එක වන protected ද බොහෝ දුරට private keyword එකට සමානයි. Object එකක් තුල “protected:” යන්න යටතේ ඇති දත්ත වලටද පිටතින් සිට ඇතුල්වීමේ අවස්ථාවක් නැත. නමුත් **protected** keyword එකෙහි ඇති වෙනස නම්, object එකක protected යටතේ define කර ඇති දත්ත වලට ඇතුලු වීමේ අවසරය, එම object එකින් inherit කර නිපදවනු ලැබූ derived objects වලට පමණක් ලබාදීමට එයට හැකිවීමයි. මෙය පැහැදිලි කර ගැනීම සඳහා පහත දැක්වෙන පරිදි, Plane class එකට **protected:** keyword එක යටතේ LandingGears නැමැති bool variable එකක් declare කරන්න. LandingGears හි අගය true කිරීමෙන් ගුවන් යානයේ ගොඩබැසීම සඳහා වන ගියර පද්ධතිය ක්‍රියාත්මක කර ඇති බව දක්වන අතර එය false කිරීමෙන් යානය ගොඩබැසීමට සූදානම් නැති බව දැක්වෙනු ඇත:

```
==========================================

class Plane {

  private:
     int Speed;
     int Direction;
     int Altitude;

  public:
     void ChangeSpeed(int spd);
     void ChangeDirection(int dir);
     void ChangeAltitude(int alt);

  protected:
     bool LandingGears;

};

==========================================
```

දැන් පහත පරිදි ඔබ මෙම protected variable එකෙහි අගය main() තුල සිට true කිරීමට උත්සාහ ගතහොත් එයින් error එකක් නිපදවෙනු ඇත. මෙනිසා protected දත්තද private මෙන්ම object එකට පිටත සිට භාවිත කල නොහැකි බව පෙනේ.

```
==========================================

void main()
{
     Plane p;
     p.LandingGears = true;
}
==========================================
```

නමුත් ඔබ Plane හි derived object එක වන FighterPlane තුල සිට එය කිරීමට උත්සාහ කරන්න. එය සාර්ථක වනු ඇත. පහත දැක්වෙන්නේ FighterPlane තුල සිට LandingGears හි අගය true කිරීමට ගත් උත්සාහයයි. මේ සඳහා FighterPlane තුල SetLandingGears නැමැති නව function එක declare කර ඇත:

```
==========================================
```

```
class FighterPlane: public Plane
{

  public:
     void FireMissile();
     void SetLandingGears(bool st);

};

==========================================
```

SetLandingGears() මගින් LandingGears variable එක නියමිත true හෝ false අගයට පත්කෙරේ:

```
==============================================

void FighterPlane::SetLandingGears(bool st){

     LandingGears=st;

}

==============================================
```

Constructors

Plane object එක තුල කිසියම් මොහොතක ගුවන්යානයේ වේගය ලබාගැනීමට GetSpeed නැමැති function එක ඇතැයි සිතන්න. පහත, GetSpeed function එක Plane class එක තුලට ඇතුල් කර ඇත:

```
==========================================

class Plane
{

  private:
     int Speed;
     int Direction;
     int Altitude;

  public:
     void ChangeSpeed(int spd);
     void ChangeDirection(int dir);
     void ChangeAltitude(int alt);
     int GetSpeed();

};

==========================================
```

GetSpeed function එක මෙසේ define කරමු:

```
==========================================

int Plane::GetSpeed(){
```

```
     return Speed;

}
```

==========

GetSpeed() මගින් Speed හි දැන් පවතින අගය return කරනු ඇත.
දැන් main() තුල සිට අපට ගුවන්යානයේ වේගය මෙලෙස ලබාගත හැක.

==========

```
void main(){
     Plane p;
     int sp = p.GetSpeed();
     cout << sp << endl;
}
```

==========

නමුත් මෙහිදී එක් ගැටලුවක් පැනනගියි. ඔබ වැඩසටහන ක්‍රියාත්මක කිරීමෙන් පසු පලමුවෙන් සිදුකරන්නේ මෙලෙස වේගය සෙවීම යැයි සිතන්න. එවිට Speed හි අගය ලෙස ලැබෙන්නේ කවරක්ද? බින්දුවද? නැත. එවිට සිදුවන්නේ Speed variable එක කිසියම් අගයකට define කිරීමට පෙර එයින් අගයක් ලබා ගැනීමට උත්සාහ කිරීමයි. ඔබ කිසියම් variable එකක් define කර නැතිවිට එහි කිසිම අර්ථයක් ඇති සංඛ්‍යාවක් අන්තර්ගත නොවේ. මේනිසා ඔබ ඉහත මෙන් variable එකක් define කිරීමට පෙර එහි අගය ලබාගතහොත් එය අර්ථයක් නැති ක්‍රියාවක් වනු ඇති අතර එය වැරදි ක්‍රියාවක්ද වේ. මේනිසා පරිගණක වැඩසටහනක් ආරම්භයේදීම එහි ඇති variables සියල්ල ස්වයංක්‍රියව define කිරීමට මාර්ගයක් තිබිය යුතුය. මෙම ක්‍රියාව, **variable initialisation** ලෙස හැඳින්වේ. Class variables, initialise කිරීමේ එම ක්‍රමය නම් **constructor** functions භාවිතා කිරීමය.

ඇත්තෙන්ම constructor functions ඔබ දැනටමත් එසේ කරනවාදැයි නොදැන පාවිච්චියට ගෙන ඇත. ඔබ C++ පරිගණක වැඩසටහනක් තුල class එකක් සෑදූ විට ඔබට නොපෙනෙන ලෙස එහි constructor function එක define වේ. මෙය සිදුවන්නේ ඔබ class එකෙහි නම type කරන අවස්ථාවේදීය. එම class එකෙහි instance එකක් සෑදූ විට සිදුවන පලමු වන දෙය නම් මෙම නොපෙනෙන constructor function එකට නොපෙනෙන පරිදිම call වීමය. එනම් constructor function එකට call වීම සිදුවන්නේ ස්වයංක්‍රියව, class instance එක නිපදවන අවස්ථාවේදීයි. තවද, constructor එකක නම වන්නේ එහි class එකෙහි නමමය. ඒ අනුව Plane class එක තුල ඇති constructor function එකෙහි නම වන්නේද Plane ය. Constructor function එකකට return type එකක් නොමැත. එමෙන්ම මුල් (default) අවස්ථාවේදී එයට කිසිම argument එකක් ද නොමැත. මේ අයුරින්, Plane class එකෙහි constructor එක declare වනු ඇත්තේ පහත පරිදිය:

```
Plane();
```

නමුත් Constructor එක වැඩසටහන තුල රහසේම පවතින අතර එයට රහසේම call වෙයි. එසේනම් ඉහත සඳහන් වූ පරිදි අප මෙම constructor එක අපගේ class එක තුල ඇති variables initialise කිරීම පිණිස යොදාගන්නේ කෙසේද? ඉතා පහසුයි. අප කලයුත්තේ constructor එක අපට අවශ්‍ය ආකාරයෙන් වෙනස් කර එය අප විසින්ම define කිරීමයි. Constructor එකක් භාවිතා කර Plane class එකෙහි ඇති Speed, Direction, Altitude යන variables, initialise කරන ආකාරය පහත විස්තරවේ.

පලමුවෙන්ම කල යුතු වන්නේ Plane class එක තුල අප විසින්ම constructor එක declare කිරීමයි :

==========

```
class Plane {
```

```
private:
     int Speed;
     int Direction;
     int Altitude;

public:
     Plane(); // The constructor
     void ChangeSpeed(int spd);
     void ChangeDirection(int dir);
     void ChangeAltitude(int alt);
     int GetSpeed();

protected:
     bool LandingGears;

};
```

==========

Plane හි public - access controller keyword එක යටතේ constructor එක declare කර ඇති අයුරු ඔබට පෙනේ. දැන් කලයුත්තේ constructor එක define කිරීමය:

==========

```
Plane::Plane(){

     Speed = 0;
     Direction = 0;
     Altitude = 0;
}
```

==========

Constructor definition එක තුල සියලු variable වලට මුල් අගයන් ලබා දී ඒවා initialise කර ඇති ආකාරය ඉහත දැක්වේ. දැන් ඔබ main() හි හෝ වැඩසටහනෙහි වෙනත් කවර හෝ අඳුරු මුල්ලක සිට Plane හි instance එකක් සෑදුවද එම සාදන මොහොතේදී පලමුව සිදුවන දෙය නම් ස්වයංක්‍රියව මෙම constructor එකට call වී එහි ඇති variables නිසි පරිදි initialise වීමය. මේ නිසා ඔබ මුලින්ම කරන දෙය වන්නේ GetSpeed() මගින් ගුවන්යානයේ වේගය ලබාගැනීම වුවද ඔබට එහි වේගය ලෙස අර්ථයක් සහිත අගයක් වන 0 ලැබේ. වේගය බින්දුව වීමෙන් පැවසෙන්නේ ගුවන්යානය ගමන් නොකරන බව ඔබට පැහැදිලිය. ඔබ constructor එක පාවිච්චි නොකලා නම් ඔබට Speed ලෙස බොහෝවිට ලැබෙනු ඇත්තේ අර්ථ විරහිත ඍණ අගයකි.

පහත දැක්වෙන්නේ constructor එක යොදා වෙනස් කරන ලද මුලු පරිගණක වැඩසටහනයි. දැන් GetSpeed() මගින් මුල් වේගය ලෙස 0, return කරනු ඇත.

==========

```
#include<iostream.h>

class Plane
{

private:
     int Speed;
     int Direction;
     int Altitude;
```

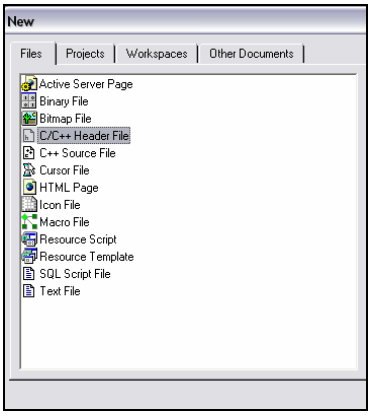
```
public:
      Plane();
      int GetSpeed();
      void ChangeSpeed(int spd);
      void ChangeDirection(int dir);
      void ChangeAltitude(int alt);

protected:
      bool LandingGears;

};

Plane::Plane(){
      Speed = 0;
      Direction = 0;
      Altitude = 0;
}

void Plane::ChangeSpeed(int spd)
{
      Speed = spd;
      cout<< Speed <<endl;
}

void Plane::ChangeDirection(int dir)
{
      Direction = dir;
      cout<< Direction <<endl;
}

void Plane::ChangeAltitude(int alt)
{
      Altitude = alt;
      cout<< Altitude <<endl;
}

int Plane::GetSpeed(){
      return Speed;
}

class Pilot
{
public:
      void ChangePlaneSpeed(Plane p, int spd);
      void ChangePlaneDirection(Plane p, int dir);
      void ChangePlaneAltitude(Plane p, int alt);
};

void Pilot::ChangePlaneSpeed(Plane p, int spd)
{
      p.ChangeSpeed(spd);
}

void Pilot::ChangePlaneDirection(Plane p, int dir)
{
      p.ChangeDirection(dir);
}

void Pilot::ChangePlaneAltitude(Plane p, int alt)
{
```

```
	p.ChangeAltitude(alt);
}

class FighterPlane:public Plane
{
	public:
	void FireMissile();
	void SetLandingGears(bool st);
};

void FighterPlane::FireMissile(){
	cout<< "Missile fired!" << endl;
}

void FighterPlane::SetLandingGears(bool st){
	LandingGears=st;
}

void main(){

	Plane p;
	int sp = p.GetSpeed();
	cout << sp << endl;

}
```

==========

Header files

ඉහත වැඩසටහන ඔබ නිමකරනු ලැබුයේ එකම C++ file එකකින්ය. නමුත් බොහෝවිට Object oriented programe එකක අඩුම වශයෙන් files 2 ක් හෝ අඩංගුය. පලමුවැන්න නම් C++ file එක වන අතර දෙවැන්න හඳුන්වන්නේ Header file එකක් වශයෙනි. (ඇත්තෙන්ම ඔබ Library එකක් ලෙස වැඩසටහනට include කරන්නේ මෙම header files ය) Header file එකක extension වන්නේ **“ .h ”** යන්නය. මෙම file එක වැඩසටහනක classes declare කිරීම සඳහා යොදාගත හැක. ඉහත දී නම් ඔබ කලේ classes සියල්ල එම classes define වීම සිදුවන C++ file එක තුලම declare කිරීමයි. නමුත් සාමාන්‍යයෙන් එම classes වෙනම header file එකක් තුල declare කිරීම සිදුවේ. මෙය සිදු කිරීමට ඔබ Visual C++ හි **File>new** menu එකින් **“C/C++ Header File”** නැමැති item එක මගින් නව header file එකක් ඔබේ වැඩසටහනට ඇතුල් කල යුතුය (රූපය 8.1)



(රූපය 8.1)

ඉන්පසු මෙම නව header file එක තුල ඔබේ සියලු classes declare කල හැක. එමෙන්ම ඔබේ වැඩසටහනට #include කලයුතු සියලු libraries ද මෙම header file එක තුලට #include කල හැක. ඉහත වැඩසටහනෙහි Plane class එක සඳහා ඔබ “Plane.h” යනුවෙන් header file එකක් සෑදුවහොත් එහි අඩංගු කල යුතු දෑ පහත පරිදි වනු ඇත.

==========

```
// Plane.h

#include<iostream.h>

class Plane
{

private:
      int Speed;
      int Direction;
      int Altitude;

public:
      Plane();
      int GetSpeed();
      void ChangeSpeed(int spd);
      void ChangeDirection(int dir);
      void ChangeAltitude(int alt);

protected:
      bool LandingGears;

};
```

==========

දැන් ඔබ කල යුත්තේ පහත පරිදි මෙම header file එක ඔබේ වැඩසටහනෙහි C++ file එක තුලට #include කිරීමයි. ඔබ විසින් සාදන ලද header file එකක් include කිරීමේදී සිදුකල යුතු වෙනස වන්නේ library එකක් include කිරීමේදී #include සමග යෙදෙන ‘<’ සහ ‘>’ යන සලකුණු වෙනුවට double quaots(“ ”) යෙදීමයි:

==========

```
#include “Plane.h” // including a header file

Plane::Plane(){
      Speed = 0;
      Direction = 0;
      Altitude = 0;
}

void Plane::ChangeSpeed(int spd)
{
      Speed = spd;
      cout<< Speed <<endl;
}

..............
..............
..............
```

```
void main(){
      Plane p;
      int sp = p.GetSpeed();
      cout << sp << endl;

}

================================================
```

ඉහත දැක්වෙන පරිදිම, FighterPlane සහ Pilot classes වල අඩංගු declarations ද header files තුලට ඇතුලු කර ඒවාද C++ file එකට include කල හොත්, C++ file එක තුල මීලීගට අඩංගු කල යුත්තේ function definitions සහ main() function එක පමණි. Class declarations සහ include කරන ලද libraries ඇත්තේ C++ file එකට include කරන ලද header files තුලය. සාමාන්‍යයෙන් වැඩසටහනක ඇති සෑම class එකක් සඳහාම header file එකක් තැනීම සුදුසුය.

9. C++ Libraries

මෙම පොතෙහි මුල් භාගයේදී Libraries ගැන ඔබ යන්තමින් ඉගෙන ගත්තා. Libraries තුල අඩංගු වන්නේ ලියා ගබඩා කර තබන ලද C++ පරිගණක කේතයන් බවද libraries අපගේ වැඩසටහන් තුල

පාවිච්චියට ගැනීමෙන් එම කේතයන් අපගේ වැඩසටහන් තුල භාවිතා කිරීමට හැකියාව ලැබෙන බවද ඔබ දුටුවා.

Visual C++ තුල අපට පාවිච්චියට ගත හැකි libraries රැසක් ම ඇත. නමුත් බොහෝවිට මේවා අප පාවිච්චියට ගන්නේ ඒවා අප ප්‍රයෝජනයට ගන්නවාදැයි හෝ නොදැනය. එලෙස සිදුවන්නේ, අප කිසියම් library එකක් අපගේ වැඩසටහනට ඇතුලත් කල විට එම library එකට සම්බන්ධ කල වෙනත් libraries ද ඉබේම වැඩසටහන තුලට පැමිණීමයි. එසේනම් අප එවැනි libraries බොහෝවිට පාවිච්චි කරන්නේ සෘජුව නොවේ. නමුත් සෘජුවම අප ලියන වැඩසටහන් වලට ඇතුල් කල යුතු libraries ඇත.

C++ Library එකක ඇත්තෙන්ම අඩංගු වන්නේ ලියා ගබඩා කර තබන ලද **classes** ය. මේ අනුව කිසියම් C++ library එකක් භාවිතා කරන විට අප සිදුකරන්නේ එම library එක තුල අඩංගු Classes තුල අන්තර්ගත functions සහ වෙනත් දත්ත පාවිච්චියට ගැනීමය. නමුත් C වර්ගයේ libraries තුල classes නොමැත.

දැන් අප Visual C++ තුල අන්තර්ගත වන Libraries කිහිපයක් පිලිබඳව සළකා බලමු.

fstream හෙවත් file stream library

File stream library එක තුල අන්තර්ගත වන්නේ දෘඪ තැටියේ ඇති files සම්බන්ධව කටයුතු කිරීමේදී භාවිතයට ගැනෙන functions ආදියයි. ඔබ විසින් ලිවීමට අදහස් කරන කිසියම් පරිගණක වැඩසටහනකදී දෘඪ තැටියේ ඇති files සම්බන්ධව කටයුතු කිරීමට අවශ්‍ය නම්, ඔබ **fstream** library එක එම වැඩසටහනට අන්තර්ගත කල යුතුය. fstream තුල **fstream** (file stream), **ofstream** (output file stream) සහ **ifstream** (input file stream) යන classes ත්‍රිත්වය අන්තර්ගතය. අප files සම්බන්ධ කටයුතු සිදුකරන්නේ මෙම classes තුල අඩංගු functions භාවිතයෙනි.

පහත දැක්වෙන්නේ fstream භාවිතා කරමින් ලියනු ලැබූ කුඩා වැඩසටහනකි. එම වැඩසටහන ප්‍රථමයෙන් " test " නමින් txt file එකක් ඔබේ පරිගණකයේ C:\ තැටියෙහි නිපදවනු ඇති අතර ඉන්පසු එය කුඩා message එකක් එම file එක තුල ලියනු ඇත:

```
================================================

#include<fstream.h>

void main()
{
     fstream file;
     file.open("C:\\test.txt", ios::app);
     file << "...Appending data";
}

================================================
```

මෙම වැඩසටහන compile කර Run කරන්න. ඉන්පසුව C:\ drive එක විවෘත කර එහි test නමින් text file එකක් නිමවී ඇති අයුරු බලන්න. එම file එක විවෘත කලහොත් `"...Appending data"` යන්න එහි ලියවී ඇති බව ඔබට පෙනෙනු ඇති.

දැන් වැඩසටහනෙහි සිදුවන්නේ කුමක්දැයි බලමු. පලමුව **#include** keyword එක මගින් fstream.h හෙවත් file stream library එක වැඩසටහන තුලට අන්තර්ගත කර ඇත.
`fstream file;` යන කේතය මගින් සිදුවන්නේ fstream හි class instance එකක් file නමින් තැනීමය.
දැන් `file.open("C:\\test.txt", ios::app);` යන කේතය මගින්, fstream class එකෙහි open functions එකට call වේ. open මගින් C:\ මත "test.txt" file එක නිපදවයි. එහි 1වැනි argument එක වන්නේ නිපදවිය යුතු file එකෙහි path එකයි. ඉහත එය `"C:\\test.txt"` ය. දෙවැනි argument එක වන්නේ නිපදවන file එකට දත්ත ඇතුල් කරන විට එය සිදුවිය යුතු ආකාරයයි. ඉහත ios::app මගින් දක්වන්නේ දත්ත ඇතුල් තරන සෑම විටම එම දත්ත, දැනට file එක තුල ඇති දත්ත වලට append කරන ලෙස හෙවත් "අගට අමුණන" ලෙසයි. එය සිදුවන්නේ `file << "...Appending data"`

කේතයෙනි. මෙහිදී file එක තුලට දත්ත output කරන්නේ << operator එක මගිනි. ios::app මගින් දත්ත append වීම සිදුවන හෙයින්, ඔබ ඉහත වැඩසටහන ක්‍රියාත්මක කරන සෑම අවස්ථාවකදීම "…Appending data" යන්න file එකතුල දිගින් දිගට එක පෙලක් ලෙස ලියැවෙනු ඇත. ios::app හි app යනු ios නැමැති class එක තුල අඩංගු වන, සංඛ්‍යාත්මක අගයක් රැගත් variable එකකි. ඉහත පරිදි එය ofstream තුලට pass කල විට file එකට දත්ත append වීම සිදුවේ. ඔබට අවශ්‍ය නම් මෙම ක්‍රියාකාරීත්වය වෙනස් කිරීමට හැකිය. උදාහරණයක් ලෙස ios::out variable එක යෙදීමෙන් වන්නේ, ඉහත වැඩසටහන ක්‍රියාත්මක වීමේදී, දැනටමත් අදාල නමින් file එකක් ඇත්නම් එහි දැනට ඇති දත්ත මකාදැමීමය. පහත දැක්වෙන්නේ මේ ආකාරයෙන් open වෙත pass කල හැකි variables ය. මේවාට පොදුවේ "File creation flags" නැමැති නම භාවිත වේ.

`Ios::app`	අලුත් දත්ත file එකෙහි අගට append කරයි.
`Ios::binary`	File එක binary දත්ත ලෙස විවෘත කරයි.
`Ios::nocreate`	දක්වා ඇති file එක දැනටමත් දෘඩ තැටිය මත පවතියි නම් පමණක් එය විවෘත කරයි.
`Ios::noreplace`	දක්වා ඇති file එක දැනට දෘඩ තැටිය මත නොපවතියි නම් පමණක් එය විවෘත කරයි.
`Ios::out`	දත්ත ලිවීම සඳහා විවෘත කරයි. දැනට ඇති දත්ත මකාදමයි.
`Ios::trunk`	දත්ත ලිවීම සඳහා විවෘත කරයි. දැනට ඇති දත්ත මකාදමයි.

ඉහත යොදා ඇති "විවෘත කිරීම" යන්නෙන් දැක්වෙන්නේ file එක පාවිච්චි කරන්නා විසින් විවෘත කිරීමක් නොවන බව සලකන්න. File එකකට C++ මගින් කිසිවක් සිදුකිරීමට පෙර එය C++ වැඩසටහන විසින් "විවෘත කල යුතුයි". ඉහත විවෘත කිරීම ලෙස හැඳින්වෙන්නේ එයයි.

File එකක් විවෘත කිරීමට හැකි හෝ නොහැකඳැයි දැනගැනීම

මේ සඳහා පහත කේත කොටස භාවිතා කල හැකිය. පහත **fail** function එක මගින්, file එක open කල නොහැකිනම් true අගය නිපදවනු ඇත:

```
========================================

fstream file;
file.open("C:\\test.txt", ios::app);

if(file.fail()){

    cout<< "File open error!" << endl;

}

========================================
```

Files කියවීම

පහත දැක්වෙන වැඩසටහනේදී, while loop එකක් මගින් file එකක අඩංගු characters සියල්ල, එක් වරකදී character එක බැගින් char variable එකක් තුල ගබඩා කරණු ලැබේ. එහිදී file එකින් දත්ත ලබා ගැනෙන්නේ හෙවත් file එකින් දත්ත කියවනු ලබන්නේ **get** function එක මගිනි. get මගින් file එකෙහි ඇති characters එක බැගින් කියවා එය c තුලට ඇතුල් කරයි. ඉන්පසු cout මගින් එම character එක Terminal window එක මතට output කරයි.

```
================================================

#include<fstream.h>
#include<iostream.h>

void main()
{
     fstream file;
     file.open("C:\\test.txt",ios::in|ios::out);
     char c;
     while(!file.eof()){
          file.get(c);
          cout<< c <<endl;
     }
}

================================================
```

string library

string library එක ඔබ ලියන වැඩසටහනකට අන්තර්ගත කිරීමෙන්, ඔබට වැඩසටහන තුල **string data type** එකට අයත් variables භාවිතයට ගැනීමට හැකියාව ලැබේ. ඔබ මීට ඉහතදී char arrays භාවිතා කරන ආකාරය දුටුවා. char arrays භාවිතා වන්නේ text strings සමග කටයුතු කරන විටදී බවද දුටුවා. නමුත් ඔබ දුටු පරිදි char arrays භාවිතා කිරීම මඳක් දුශ්කර ය. Text strings යනු පරිගණක වැඩසටහනකදී භාවිතයට ගැනෙන සුලභ දෙයකි. මෙවැනි සුලභව භාවිතා වන දෙයක් දුශ්කර අයුරින් පාවිච්චියට ගැනීමට සිදුවීම නුසුදුසුය. මෙයට පිලියමකි මෙහිදී අප පාවිච්චි කිරීමට සූදානම් වන string data type එක.

string එකක් යනු ඇත්තෙන්ම අභ්‍යන්තර ලෙස සකසන ලද char array එකක්ය. නමුත් string එකක් යනු හුදෙක් char array එකක් පමණක්ම නොවන අතර එය C++ class එකකි. මේ බව string නමින් library එකක් C++ තුල තිබීම මගින් පැහැදිලි වේ.

මෙම **string class** එක තුල ඇත්තේ char arrays භාවිතයේදී ඔබට මුහුණපෑමට සිදුවන අපහසුතා මගහැරීමට නිර්මාණය කරන ලද functions ය. දැන් අප මෙම function මොනවාද සහ ඒවායින් සිදුකිරීමට හැක්කේ කවරක්දැයි විමසා බලමු.

පහත දැක්වෙන්නේ string library එක වැඩසටහනකට ඇතුල් කර string type variable එකක් define කරන ආකාරයයි:

```
========================================

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

void main(){
```

```
     string str = "The Millenium";
     cout<< str << endl;
}
```

==========

string data type එක භාවිතා කරනවිට මුලින් කලයුතු දේ 2ක් ඇත. පලමුවැන්න නම් මෙහිදී වැඩසටහන තුලට #include මගින් ඇතුල් කලයුතු වන්නේ **iostream.h** file එක නොව **iostream** file එක වීමයි. දෙවනුව, string library එක #include කිරීමෙන් පසුව ඉහත වැඩසටහනෙහි දැක්වෙන පරිදි, **using namespace std;** නැමැති කේතය type කල යුතුය. මේ ක්‍රියා දෙක සිදුකිරීමෙන් අනතුරුව ඔබේ වැඩසටහන තුල strings පාවිච්චියට ගත හැක. ඉහත වැඩසටහන Run කල විට **string** str හි ගබඩා වී ඇති “The Millenium” යන text string එක print වනු ඇත.

ඉහත වැඩසටහනෙහි සිදුකර ඇත්තේ str නමින් string class එකෙහි instance එකක් සෑදීමය. string class එක තුල ඇති functions වලට call කිරීමට නම් අප සාමාන්‍යයෙන් classes සම්බන්ධයෙන් සිදුකරන පරිදිම, මෙම str instance එක මගින් function calls සිදුකල යුතුය. දැන් අප මෙතැන්සිට string class එක තුල ඇති මෙම functions හඳුනා ගැනීම අරඹමු.

<u>append function</u>

append function එක භාවිතා කරන්නේ කිසියම් string එකකට තවත් string එකක් සම්බන්ධ කිරීමට සිදුවන අවස්ථාවලදීය. append function එකෙහි අවස්ථා කීපයක් ඇත. මෙසේ වන්නේ class string තුල append නමින් යුත් functions කීපයක් තිබීම හේතුවෙනි. මේ අවස්ථා අතුරින් 2ක් පිලිබඳ සලකා බැලීම සුදුසුය.

1. append (***string*** *str*)

append function එකෙහි මෙම අවස්ථාවේදී අප එයට pass යුතු වන *str* නැමැති argument එක වන්නේ, මුල් string එකට append විය යුතු දෙවැනි string එකයි. පහත උදාහරණය බලන්න:

==========

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

void main(){
     string str1 = "The Millenium";
     string str2 = " & The X-Files";
     str1.append(str2);
     cout<< str1 << endl;
}
```

ඉහතදී **str1** හා **str2** නැමැති strings 2 ක් සාදා ඇත. අපට මෙහිදී අවශ්‍ය වන්නේ str1 string එකෙහි අගට str2 string එක සම්බන්ධ කිරීමයි. මේ සඳහා str1 මත append function එකට call කර ඇති ආකාරය බලන්න: str1.append(str2). append වෙත pass කර ඇති argument එක වන්නේ str2 string එකය. මෙයින් සිදුවන්නේ str1 තුල ඇති text string එක වන “The Millenium” යන්න අගට str2 තුල ඇති “ & The X-Files” යන text string එක append හෙවත් සම්බන්ධ වීමය. එවිට function call එකෙහි ප්‍රථිපලය ලෙස str1 string එකෙහි අඩංගු වන string එක “The Millenium & The X-Files” යන්න බවට පත් වනු ඇත.

නමුත් මෙය append function එකට call කිරීමකින් තොරවද සිදුකල හැකිය. මේ සඳහා අප පාවිච්චියට ගන්නේ සුපුරුදු **“ + ” operator** එකයි. පහත මෙම අවස්ථාව දැක්වේ:

==========

```
#include<iostream>
```

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

void main(){
     string str1 = "The Millenium";
     string str2 = " & The X-Files";
     str1 = str1 + str2;
     cout<< str1 << endl;
}
```

==========

එනම් අප integer data හෙවත් සංඛ්‍යා සම්බන්ධයෙන් භාවිතා කරනු ලබන + operator එක මෙහිදී strings සම්බන්ධයෙන්ද භාවිතාවට ගත හැකිය.

දැන් append හි 2 වන අවස්ථාව සලකා බලමු.

2. append (***string*** *str*, ***int*** *position*, ***int*** *numberOfChars*)

append function එකෙහි මෙම 2 වැනි අවස්ථාවේදී එක් string එකකට වෙනත් string එකක තෝරාගත් කොටසක් පමණක් සම්බන්ධ කිරීමේ හැකියාව ඇතිවේ. Function එකට pass කල යුතු පලමු argument එක වන්නේ මුල් string එකට සම්බන්ධ කල යුතු 2 වැනි string එකයි ***(string*** *str)*. 2 වැනි argument එක නම් *str* string එකෙන් append කිරීම ආරම්භ කල යුතු ස්ථානයයි (***int*** *position*). තෙවැනි argument එක නම්, *position* argument එකින් දැක්වෙන ස්ථානයේ සිට *str* string එකින් append කල යුතු characters සංඛ්‍යාවයි(***int*** *numberOfChars*). මෙය වඩාත් පැහැදිලි කරන උදාහරණයක් පහත දැක්වේ. එහිදී සිදුවන්නේ str1 හි අන්තයට, str2 හි 2 වන character එකෙහි සිට 8 වැනි characters එක දක්වා ඇති කොටස සම්බන්ධ කිරීමයි:

==========

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

void main(){
     string str1 = "The Millenium";
     string str2 = " & The X-Files";
     str1.append(str2, 2,8);
     cout<< str1 << endl;
}
```

==========

ඉහත, str1 වෙත str2 හි “The X-F” යන කොටස append වේ. මෙය str2 හි 2 වන character එකෙහි සිට 8 වන character එක දක්වා ඇති කොටසයි. str2 හි 2 වන character එක යනු එහි “T” ට පෙර ඇති හිඩැසයි. ඇතැම් විට මෙම හිඩැස ඇත්තේ 2 වන ස්ථානයේ නොව තෙවන ස්ථානයේ නොවේදැයි ඔබට සිතෙන්නට පුලුවන. නමුත් ඇත්තෙන්ම string එකක character index එක ආරම්භ වන්නේ බින්දුවෙනි. එසේ වන්නේ string එකක් යනු ඇත්ත වශයෙන්ම character array එකක් වීම නිසාය. ඔබ දන්නා පරිදිම array එකක index එක ආරම්භ වන්නේ 0 වෙනි. මේනිසා අපට str2 හි “T” ට පෙර ඇති හිඩැස හමුවන්නේ එහි 2 වන ස්ථානයේදීයි. මේ අනුව අවසානයේදී str1 හි අගය වන්නේ “The Millenium” හා “The X-F” එකතු වීමෙන් සැදෙන “The Millenium The X-F” යන්නය.

<u>assign function</u>

assign function එක මගින් සිදුවන්නේ පවතින string එකක ඇති දත්තය මකා දමා එයට වෙනත් string එකක් හෝ string එකින් කොටසක් ආදේශ කිරීමය. assign function එකටද අවස්ථා කීපයක් ඇත.

1. assign (***string*** *str*)

assign හි මෙම අවස්ථාවේදී සිදුවන්නේ *str* හි ඇතිදේ හුදෙක් වෙනත් string එකක් වෙත ආදේශ කිරීමය. මෙය පහත දැක්වේ:

```
========================================

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

void main(){
     string str1 = "The Millenium";
     string str2 = " & The X-Files";
     str1.assign(str2);
     cout<< str1 << endl;
}

========================================
```

ඉහතදී str2 හි ඇති දේ str1 ට ආදේශ වේ. මෙහෙයින්, assign වීම අවසන් වූ විට str1 හා str2 එකිනෙකට සමාන වනු ඇත.

2. assign (***string*** *str*, ***int*** *position*, ***int*** *numberOfChars*)

assign function එකෙහි arguments පිලිබඳ විස්තර පහත දැක්වේ.

Arguments

string *str*:
මුල් string එකට assign කල යුතු 2 වැනි string එක.

int *position:*
str string එකින් assign කිරීම ආරම්භ කල යුතු ස්ථානය.

int *numberOfChars:*
int *position* argument එකින් දැක්වෙන ස්ථානයේ සිට, මුල් string එකට, *str* string එකින් append කල යුතු characters සංඛ්‍යාව.

උදා:

```
========================================

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

void main(){
     string str1 = "The Millenium";
     string str2 = " & The X-Files";
```

```
     str1.assign(str2,2,8);
     cout<< str1 << endl;
}
```

==========

ඉහත උදාහරණයේදී str1 වෙත str2 තුල ඇති text string එකෙහි index 2 - 8 කොටස ආදේශවේ. ඒ අනුව str1 = " The X-F" වනු ඇත.

at function

at මගින් string එකක කිසියම් ස්ථානයක ඇති character එකක් ලබා දේ.

at (***int*** *position*)

Arguments

int *position***:**
Character එක ලබාගත යුතු ස්ථානයේ index number එක.

පහත උදාහරණයේදී str1 හි 4 වන ස්ථානයේ ඇති character එක(M) ලබා ගෙන එය Terminal window එක මත print කෙරේ:

==========

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

void main(){
     string str1 = "The Millenium";
     cout<< str1.at(4) << endl;
}
```

==========

compare function

1. compare (***string*** *str*)

මෙම function එකින් strings යුගලක් සමාන දැයි පරීක්ෂා කල හැක. strings යුගල සමාන නම් function එක මගින් 0 අගය return කරනු ලබන අතර, සමාන නොවේ නම් 1 හෝ -1 අගය return කරනු ලැබේ.

Arguments

string *str**:***
පලමු string එකට සමානදැයි පරීක්ෂා කල යුතු 2 වැනි string එක.

උදා:
==========

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

void main(){
     string str1 = "The Millenium";
     string str2 = " & The X-Files";
     string str3 = "The Millenium";

     int check = str1.compare(str3); //(1)
     cout<< check << endl;
     check = str1.compare(str2); //(2)
     cout<< check << endl;
}

========================================
```

ඉහත (1) අවස්ථාවේදී int check හි අගය 0 වනු ඇත්තේ str1 හා str3 සමාන නිසාය. (2) අවස්ථාවේදී str1 හා str2 සමාන නොවන නිසා check හි අගය 1 වේ.

නමුත් නැවතත් මෙය string function එකක් භාවිතා නොකර == **operator** එක භාවිතා කර ද සිදුකල හැකිය. නමුත් මෙහිදී ලැබෙන්නේ පෙර අවස්ථාවට විරුද්ධ වන output එකක් බව ඔබට වැටහෙනු ඇත:

උදා:

```
========================================

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

void main(){
     string str1 = "The Millenium";
     string str2 = " & The X-Files";
     string str3 = "The Millenium";

     bool check = (str1 == str3); //(1)
     cout<< check << endl;
     check = (str1 == str2); //(2)
     cout<< check << endl;
}

========================================
```

2. compare (***int*** *pos*, ***int*** *charN*, ***string*** *str*)

මෙයින් එක් string එකක කොටසක් තවත් string එකකට සමාන දැයි පරීක්ෂා කල හැක. strings යුගල සමාන නම් function එක මගින් 0 අගය return කරනු ලබන අතර, සමාන නොවේ නම් 1 හෝ -1 අගය return කරනු ලැබේ.

Arguments

int *pos:*
str හි, පරීක්ෂාව සඳහා භාවිතා කල යුතු කොටසේ ආරම්භක ස්ථානය.

***int** charN:*
str හි පරීක්ෂා කිරීම සඳහා ගැනෙන characters ගණන.

***string** str:*
පරීක්ෂාවට ලක් කෙරෙන 2 වැනි string එක.

පහත උදාහරණයේදී සිදුවන්නේ str2 හි index 14 සිට ඇති characters 13 (“The Millenium”) , str1 ට සමානදැයි පරීක්ෂා කිරීමය. මෙහිදී එය සමාන වන නිසා check හි අගය 0 වේ.

```
================================================

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

void main(){
string str1 = "The Millenium";
     string str2 = "The X-Files & The Millenium";

     int check = str2.compare(14, 13, str1);
     cout<< check << endl;
}

================================================
```

empty function

මෙම function එක මගින් පරීක්ෂාවල ලක් කරන්නේ කිසියම් string එකක් හිස් string එකක්ද යන්නයි. මෙයට කිසිම argument එකක් pass කල යුතු නැත. පරීක්ෂා කරනු ලබන string එක හිස් නම්, empty මගින් true, return වේ.

උදා:

```
========================================

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

void main(){
     string str1 = "The Millenium";
     string str2 = "";

     bool check = str1.empty();
     cout<< check << endl; //(1)
     check = str2.empty();
     cout<< check << endl; //(2)
}

========================================
```

ඉහත (1) අවස්ථාවේදී, str1 තුල text string එකක් ඇති නිසා, check = false (0) වන අතර (2) අවස්ථාවේදී, str2 තුල කිසිවක් නොමැති නිසා check = true (1) වේ.

erase function

erase function එක මගින් string එකක කොටසක් මකා දැමීය හැක.

erase (***int*** *pos*, ***int*** *charN*)

Arguments

int *pos:*
අදාල string එකින් දත්ත මකා දැමීම ඇරඹිය යුතු ස්ථානය.

int *charN:*
මකා දැමිය යුතු characters ගණන.

උදා:

```
========================================

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

void main(){
string str1 = "The Millenium";

     str1.erase(4,4);
     cout<< str1 << endl;
}

========================================
```

ඉහත, str1 හි 4 වන character එකෙහි සිට characters 4 ක් මකා දැමීම සිදුවේ. එවිට ඉතිරි වන්නේ “The enium” යන්නයි.

find function

find මගින් string එකක ඇති තනි character එකක් හෝ character string එකක් සඳහා පරීක්ෂා කල හැක. අදාල string එක තුල පරීක්ෂාවට ලක් කෙරෙන character string එක තිබේ නම්, function එක එම character string එක හමුවන ස්ථානයේ index number එක return කරනු ඇත.

find (***string*** *str*)

Arguments

string *str:*
අදාල string එකෙහි තිබෙඳැයි පරීක්ෂා කල යුතු character string එක.

උදා:

```
========================================

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

```
void main(){
     string str1 = "The Millenium";
     string str2 = "Millenium";

     int pos = str1.find(str2);
     cout<< pos << endl;
}
```

==========

මෙහිදී str1 තුල str2 පවතියි දැයි පරීක්ෂා කෙරෙන අතර, str1 තුල “Millenium” යන්න හමුවන ස්ථානය වන index 4 අගය pos තුල ගබඩා වනු ඇත.

insert function

මෙමගින් එක් string එකක ඕනෑම ස්ථානයකට වෙනත් string එකක් ඇතුල් කල හැක.

insert (***int*** *pos*, ***string*** *str*)

Arguments

int *pos:*
මුල් string එකට *str* string එක ඇතුල් කල යුතු ස්ථානය.

string *str:*
ඇතුල් කල යුතු string එක.

උදා:
==========

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

void main(){
     string str1 = "The Millenium";
     string str2 = "new ";

     str1.insert(4,str2);
     cout<< str1 << endl;
}
```

==========

ඉහත, str1 තුල 4 වන ස්ථානයට str2 ඇතුල් වෙයි. ඒ අනුව නව str1 වනු ඇත්තේ “The new Millenium” යන්නය.

length function

මෙමගින් string එකක ඇති characters සංඛ්‍යාව හෙවත් එහි length එක ලබාගත හැක.

length ()

Arguments
මෙහි arguments නැත.

උදා:

```
=========================================

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

void main(){
     string str1 = "The Millenium";

     int len = str1.length();
     cout<< len << endl;
}

=========================================
```

ඉහත, str1 හි characters සංඛ්‍යාව වන 13 ලැබේ.

[] operator

මෙම operator එක භාවිතයෙන් string එකක යම් ස්ථානයක ඇති character එකක් ලබාගත හැකිය.

උදා:

```
=========================================

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

void main(){
string str1 = "The Millenium";

     char c = str1[6];
     cout<< c << endl;
}

=========================================
```

ඉහත, c තුලට str1 හි 6 වන ස්ථානයේ ඇති “l” ලබා ගැනේ.

replace function

replace function එක මගින් එක් string එකක ඕනෑම ස්ථානයක ඇති character string එකක් වෙනුවට වෙනත් string එකක් ආදේශ කල හැක.

1. replace (***int*** *pos*, ***int*** *charN*, ***string*** *str*)

Arguments

int *pos:*
str string එක මගින්, මුල් string එක replace කිරීම ආරම්භ කල යුතු ස්ථානය.

int *charN:*
replace වන විට පලමු string එකින් ඉවත් කල යුතු characters ගණන.

string *str:*
ආදේශ කල යුතු string එක.

පහත උදාහරණයේදී, str1 හි "Millenium" යන්න වෙනුවට "X-Files" යන්න ආදේශ වනු ඇත. එනිසා str1 = "The X-Files" බවට පත්වනු ඇත:

```
========================================

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

void main(){
     string str1 = "The Millenium";
     string str2 = "X-Files";
     str1.replace(4,9,str2);
     cout<< str1 << endl;
}

========================================
```

2. replace (***int*** *pos1*, ***int*** *charN1*, ***string*** *str*, ***int*** *pos2*, ***int*** *charN2*)

Arguments

int *pos1:*
str string එක මගින්, මුල් string එකෙහි replace කිරීම ආරම්භ කල යුතු ස්ථානය.

int *charN1:*
replace වන විට මුල් string එකින් ඉවත් කල යුතු characters ගණන.

string *str:*
ආදේශ කල යුතු string එක.

int *pos2:*
str string එකෙහි ආදේශය ආරම්භ කල යුතු ස්ථානය.

int *charN2:*

str string එකින් ආදේශ කල යුතු characters සංඛ්‍යාව.

පහත උදාහරණයේදී, str1 හි “Millenium” යන්න වෙනුවට str2 හි “X-Files” යන කොටස එයට ආදේශ වනු ඇත. එනිසා str1 = “The X-Files” බවට පත්වනු ඇත:

```
========================================

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

void main(){

     string str1 = "The Millenium";
     string str2 = "The X-Files";
     str1.replace(4,9,str2,4,7);
     cout<< str1 << endl;
}

========================================
```

substr function

substr function එක මගින් string එකක කොටසක් නව string එකක් ලෙස ලබාගත හැක.

substr (***int*** *pos*, ***int*** *charN*)

Arguments

int *pos:*
අදාල string එක වෙන් කිරීමට අවශ්‍ය ස්ථානය.

int *charN:*
string එකින් වෙන් කරගත යුතු characters සංඛ්‍යාව.

උදා:

```
========================================

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

void main(){
     string str1 = "The Millenium";
     string str2 = str1.substr(4,9) ;
     cout<< str2 << endl;
}

========================================
```

ඉහත උදාහරණයේදී str1 හි “Millenium” යන කොටස, substr මගින් ඡේදනය කර, එය නව string එකක් ලෙස str2 වෙත ආදේශ කෙරේ.

swap function

swap function එක මගින් එක් string එකක ඇති දත්තය වෙනත් string එකක ඇති දත්තය සමග හුවමාරු කල හැක.

substr (***string*** *str*)

Arguments

string *str:*
මුල් string එක සමග දත්ත හුවමාරු කල යුතු 2 වැනි string එක.

උදා:

```
========================================

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

void main(){
     string str1 = "The Millenium";
     string str2 = "The X-Files";
     str1.swap(str2);
     cout<< str1 << endl;
     cout<< str2 << endl;
}

========================================
```

ඉහතදී, str1 හා str2 යන strings දෙකෙහි අඩංගු text strings යුගල හුවමාරු කෙරේ. ඒ අනුව අවසානයේදී str1 හි තිබිය යුත්තේ “The X-Files” වන අතර str2 හි තිබිය යුත්තේ “The Millenium” වේ.

ctime library

ctime library එක තුල අඩංගු වන්නේ කාලය සම්බන්ධ දත්ත සමග කටයුතු කිරීම සඳහා නිර්මාණය කෙරුණු functions ය. උදාහරණයක් ලෙස පහත වැඩසටහන අද දිනය සහ කාලය console window එක මත පෙන්වයි:

```
========================================

#include<ctime>
#include<iostream.h>

void main()
{
     time_t tnow = time(0);
     cout<< asctime(gmtime(&tnow));
}

========================================
```

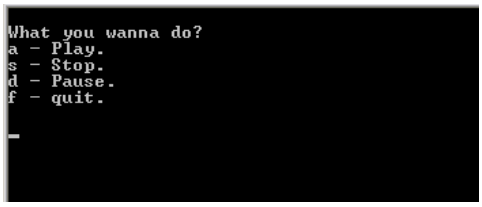
ඉහත, time_t යනු කාලය ගබඩා කිරීම පිණිස C++ තුල ඇති දත්ත වර්ගයයි. ඒ අනුව, දැන් කාලය ගබඩා කිරීමට time_t type එකින් tnow variable එක සාදා ඇත. **time** function එක මගින් දැන් වේලාව ලබාගෙන එය tnow තුල ගබඩා කෙරේ. ඉන්පසු gmtime හා asctime මගින් now තුල ඇති කාලය text string එකක් බවට පත් කොට Terminal එක මත දිස් කරවයි.

10. C++ මගින් සරල Media Player එකක් සෑදීම

මෙය අපගේ අවසන් පරිච්ඡේදයයි. මෙහිදී අප සලකා බැලීමට යන්නේ C++ මගින් සරල **Media Player** එකක් සාදන ආකාරයයි. මෙහි අප සෑදීමට අදහස් කරන පරිගණක වැඩසටහන ඇත්ත වශයෙන්ම "සුපිරි" මෘදුකාංගයක් නොවනු ඇත. ඔබට මෙය අන්තර්ජාලය තුල අලෙවි කොට මුදල් ඉපයීම වැනි දෙයක් කල හැකි නොවනු ඇත! මෙය ඉදිරිපත් කිරීමේ අදහස හුදෙක් ඉහත පරිච්ඡේද වලදී ඉගෙන ගත් සංකල්ප පාවිච්චි කර ප්‍රයෝජනවත් පරිගණක වැඩසටහන් සාදන ආකාරය පිලිබඳ මුල් වැටහීමක් ඔබට ලබාදීමයි. එමෙන්ම C++ පිලිබඳ ඔබගේ උද්‍යෝගය වැඩි කිරීමයි.

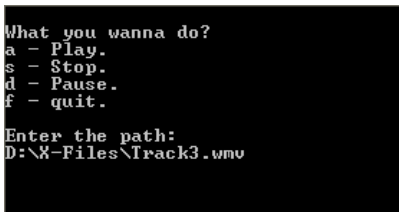
දැන්, අප සෑදීමට සූදානම් වන media player එක මගින් ඔබට mp3, wav, mpeg, avi සහ wmv යන formats වලින් යුතු media files ධාවනය කිරීමට හැකිවනු ඇත. මෙය සරල Object oriented program එකක් වනු ඇති අතර, inheritance සහ අනෙකුත් සංකීර්ණ Object oriented ක්‍රමවේදයන් භාවිතා කර නොමැති නිසා නවක ක්‍රමලේඛකයන්ට වැඩසටහන වටහා ගැනීමට පහසු වනු ඇත.

අපගේ Media player program එකට ඇත්තේ පහත දැක්වෙන ආකාරයේ සරල DOS interface එකකි (මතු පිටකි).



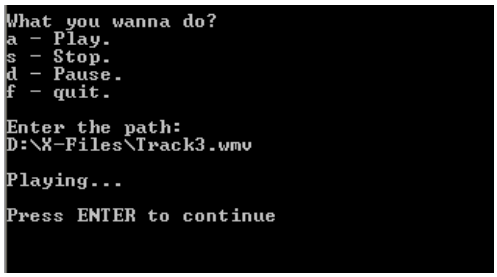
(රූපය 10.1)

ඔබ වැඩසටහන ධාවනය කල විට පලමුවෙන්ම දිස්වන menu එක ඉහත ආකාරය ගනු ඇත. ඔබට audio හෝ video file එකක් play කිරීමට අවශ්‍ය වූ විට කල යුත්තේ keyboard එකෙහි A යතුර එබීමයි. එවිට වැඩසටහන මගින් පහත දැක්වෙන පරිදි file path එකක් විමසනු ඇත.



(රූපය 10.2)

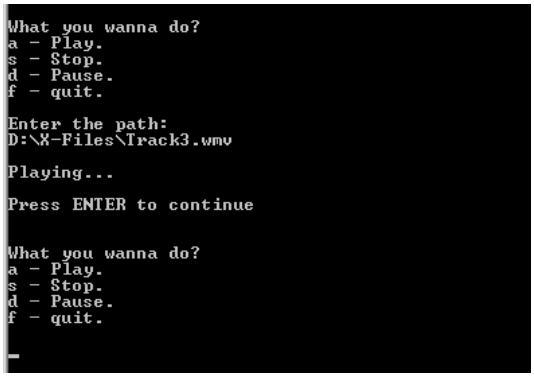
ඔබ file path එක වැඩසටහනට නිවැරදිව සපයා Enter button එක press කල විට එම file එක play වීම, වැඩසටහන මගින් "Playing…" නැමැති message එක දිස් කරමින්, රූපය 10.3 හි පෙනෙන ලෙස අරඹනු ඇත. File එක video එකක් නම් video එක දිස්කිරීම සඳහා අමතර window එකක් ස්වයංක්‍රියව විවෘත වනු ඇත.



(රූපය 10.3)

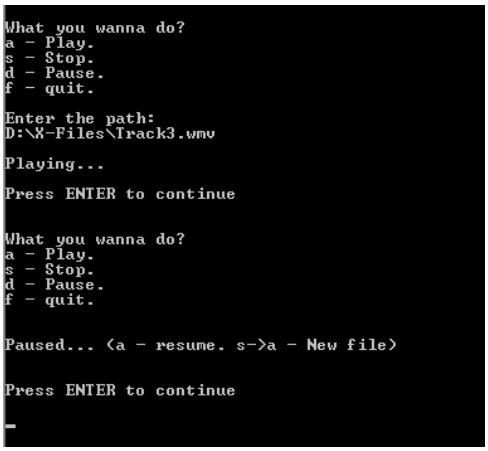
[මෙහිදී සැලකිය යුතු දෙයක් නම් media file එකෙහි නම එක් කොටසකින් පමණක් යුතු වීමට අවශ්‍ය වීමයි. කොටස් කීපයකින් යුතු නමක් ඇති file එකක්(උදා: Tears in heaven.mp3) ධාවනය කිරීමට මෙම වැඩසටහනට බොහෝවිට නොහැකි වනු ඇත. මෙයට ප්‍රතිකර්මයන් ඇති නමුත් මෙහිදී එම උපක්‍රම ගැන සාකච්චා කිරීමට නොයන්නෙමු.]

නැවත main menu එක ලබා ගැනීමට ඔබ Enter button එක press කල යුතුය. එවිට පහත ලෙස නැවතත් menu එක වැඩසටහන මගින් දිස් කරයි.



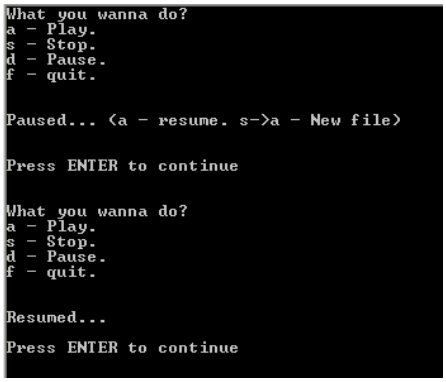
(රූපය 10.4)

දැන්, දැනට play වෙමින් පවතින media එක, D එබීම මගින් pause කල හැකිය. එවිට "Paused…" යනුවෙන් message එකක් print වේ (රූපය 10.5).



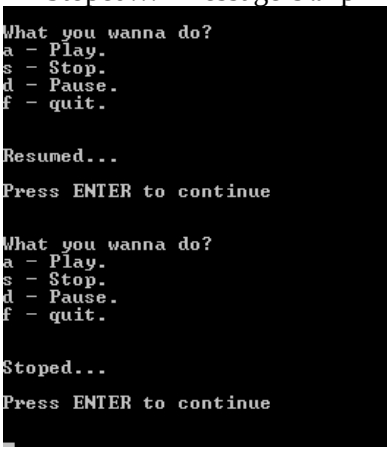
(රූපය 10.5)

නැවතත් play වීම resume කිරීමට නම් A ද නව file එකක් play කිරීමට නම් S ඔබා ඉන්පසුව A ද එබිය යුතුය. Resume කල විට "Resumed…" message එක print වේ (රූපය 10.6).



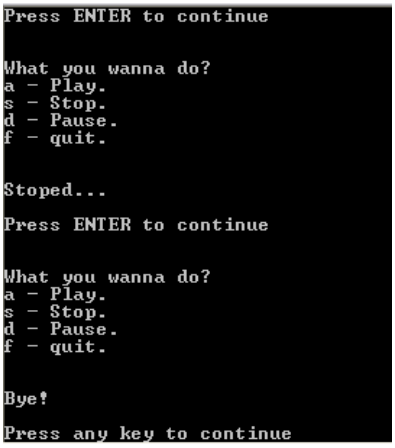
(රූපය 10.6)

S මගින් file එක stop කල හැකිය. එවිට "Stoped…" message එක print වේ (රූපය 10.7).



(රූපය 10.7)

වැඩසටහනින් ඉවත්වීමට නම් F එබිය යුතුය. එවිට "Bye!" යන message එක දිස් කරමින් වැඩසටහන නිම වනු ඇත (රූපය 10.8).



(රූපය 10.8)

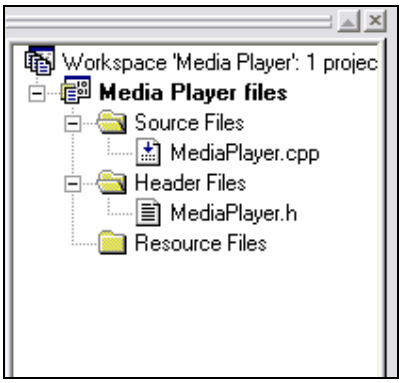
වැඩසටහන ක්‍රියාකරන්නේ ඉහත දැක්වුනු ආකාරයටයි. දැන් අප මෙම වැඩසටහන ලියන්නේ කෙසේදැයි බලමු.

Project Media Player

1. Visual C++ මගින් Media Player නමින් නව **Win32 Console Application** project එකක් සාදන්න. Project එක සාදන විට "An empty project" නැමැති option එක select කිරීමට මතක තබාගන්න.

2. දැන් "MediaPlayer" නමින් නව **C++ file** එකක් project එක තුලට ඇතුල් කරන්න.

3. "MediaPlayer" නැමැති **C/C++ header file** එකක්ද project එකට ඇතුල් කරන්න. (header files පිලිබඳව "Object Oriented Programming" නැමැති පරිච්ඡේදයේ අවසන් භාගයේදී විස්තර කෙරිණි). දැන් ඔබේ Project workspace එකෙහි FileView window එක රූපය 10.9 ආකාරය ගනු ඇත.



(රූපය 10.9)

4. දැන් **MediaPlayer.h** file එක double click කර එය විවෘත කර ගන්න.

5. MediaPlayer.h තුලට පහත දැක්වෙන පරිදි, **conio, Shlobj, iostream, string** යන libraries include කරන්න.

```
====================================

#include<conio.h>
#include<Shlobj.h>
#include<iostream>
#include<string.h>

====================================
```

conio library එක අප මෙහිදී include කරන්නේ ඒ තුල ඇති **getch()** නැමැති function එක අපගේ වැඩසටහනේදී පාවිච්චියට ගැනීම පිණිසය. **Shlobj** library එකින් අප ලබා ගන්නේ **wsprintf, mciSendString** හා **MessageBox** යන functions ය. මෙම functions පිලිබඳව පසුව විස්තර කෙරේ.

6. අප string library එක පාවිච්චි කරන නිසා මීලගට “using namespace std” යන්න යෙදිය යුතුය.

```
====================================

#include<conio.h>
#include<Shlobj.h>
#include<iostream>
#include<string.h>
using namespace std;

====================================
```

7. දැන් පහත තද කලු අකුරින් දැක්වෙන code එක type කරන්න.

```
====================================

#include<conio.h>
#include<Shlobj.h>
#include<iostream>
#include<string.h>
using namespace std;

#pragma comment(lib, "winmm.lib")

====================================
```

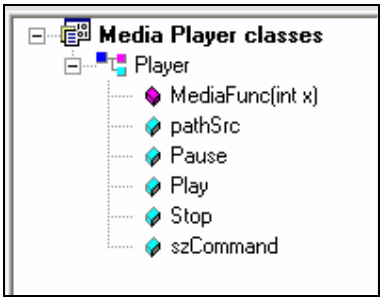
මෙම නව code එක මගින් සිදුකරන්නේ **winmm.lib** නැමැති library file එක වැඩසටහනට ඇතුල් කිරීමය. මෙය දැනට අප include කර ඇති Shlobj library එක සමග ක්‍රියාකරන අතර winmm.lib නොමැති නම් අපට ඉදිරියේදී හමුවීමට නියමිත mciSendString function එක ක්‍රියා නොකරනු ඇත.

8. Media Player වැඩසටහන සරල object oriented program එකක් ලෙස සැකසිය යුතු නිසා, වැඩසටහන තනි object එකකින් පමණක් යුතුව නිමවීමට අදහස් කෙරේ. මෙම object එක **Player** නමින් නම් කරමු. Player class එක තුල MediaFunc නමින් යුතු, තම argument එක ලෙස integer variable එකක් ලබා ගන්නා function එකක් ද Stop, Play, Pause යන boolean variables ත්‍රිත්වයද mediaCommand සහ pathSrc යන character arrays නොහොත් character strings යුගලද අප අනතර්ගත කල යුතුයි (එම variables වල හා function එකෙහි කාර්යයන් පසුව විස්තර වේ). Player class එක දැන් MediaPlayer.h **header file** එක තුල පහත පරිදි declare කරන්න.

```
   ====================================

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

```
#include<iostream>
#include<string.h>
using namespace std;

#pragma comment(lib, "winmm.lib")

class Player{
 public:
   void MediaFunc(int x);
   bool Stop;
   bool Play;
   bool Pause;
   char mediaCommand[50];
   char pathSrc[50];
};

====================================
```

ඉහත player class හි declaration එක type කිරීම අවසන් වූ විට workspace එක තුල ඇති classView window එකෙහි නව **Player class** එක හා එය තුල අන්තර්ගත function එක සහ variables පහත ලෙස දිස්වනු ඇත.



(රූපය 10.10)

Player class එකෙහි අන්තර්ගත function එකෙහි හා variables වල කාර්යයන් මොනවාදැයි දැන් අප මූලික වශයෙන් හඳුනාගනිමු.

- **MediaFunc(int *x*)** – මෙම function එකෙහි කාර්යය නම් එයට pass කරනු ලබන integer එකට අනුව ක්‍රියාවන් සිදුකිරීමය. එනම්, ඔබ එයට 1 අගය pass කලහොත් function එක මගින් ඔබ දක්වන media file එක play කිරීම ආරම්භ කල යුතුය. ඔබ pass කරන්නේ 2 නම්, එය media file එක ධාවනය වීම නැවැත්විය යුතුය (stop). Pass කරන අගය 3 නම් media file එක pause කල යුතුය.

- **pathSrc** – මෙම char array එක අප භාවිතා කරන්නේ play කිරීමට අවශ්‍ය media file එකෙහි file path එක ගබඩා කිරීමටය.

- **mediaCommand** – මෙහි කාර්යය පසුව සලකා බලමු.

- **Play, Pause, Stop** – මෙම Boolean variables ත්‍රිත්වය භාවිතා වන්නේ media player එකෙහි state එක හෙවත් දැනට එය media එකක් play කරමින් සිටිනවාද, Pause කර ඇතඳ එසේත් නැතිනම් කිසිවක් නොකරමින් සිටිනවාද යනුවෙන් වාර්ථා කිරීම පිණිසය.

ඉහත දැක්වූ functions සහ variable පිලිබඳව ඉදිරියේදී ඔබට වඩා හොඳ වැටහීමක් ලැබෙනු ඇත.

9. මීලඟට කල යුත්තේ ඉහත declare කරන ලද MediaFunc function එක define කිරීමයි. මෙය සිදුකිරීම සඳහා අප MediaPlayer.cpp file එකට යායුතුය. මක්නිසාදයත් Header files තුල කිසිවක් define කල

නොහැකි අතර ඒවා ඇත්තේ declarations සඳහා පමණක් හෙයිණි. MediaPlayer.cpp file එක open කිරීම සඳහා workspace එකෙහි MediaPlayer.cpp file එක මත double click කරන්න.

10. MediaFunc function එක define කිරීමට පලමුවෙන් MediaPlayer.h header file එක MediaPlayer.cpp file එක තුලට ඇතුල් කොට සිටිය යුතුය. මෙමගින් MediaFunc හි declaration එක වැඩසටහන තුලට ඇතුල් වේ. පහත code එකින් මෙය සිදුවේ. එය MediaPlayer.cpp තුල type කරන්න:

```
====================================

#include "MediaPlayer.h"

====================================
```

11. දැන් function එක define කිරීම අරඹමු. පහත දැක්වෙන්නේ එහි පලමු අඩියරයි (class functions define කිරීම පිලිබඳව අප “Object oriented programming” පරිච්ඡේදයේ සලකා බැලුවෙමු).

```
====================================

#include "MediaPlayer.h"

void Player::MediaFunc(int x){

}
====================================
```

දැනට MediaFunc function එක හිස් function එකක් බව ඔබට පෙනේ.

12. ඔබ ඉහත කියවූ පරිදි MediaFunc function එකෙහි කාර්යය නම් එයට pass කරනු ලබන argument එකෙහි අගයට අනුව කාර්යයන් සිදුකිරීමයි. මේනිසා function එක තුල පලමුවෙන් ම කලයුතු දේ නම් function එක තුලට pass කරන ලද variable එකෙහි අගය කුමක්දැයි පරීක්ෂා කර බැලීමයි. එවිට එක් එක් අගයන් pass කලවිට සිදුකල යුත්තේ කුමක්දැයි අපට තීරණය කලහැක. මේ සඳහා අප මෙහිදී යොදාගන්නේ **switch – case** program flow control statement එකය (program flow control statements පිලිබඳව අප 4 වන පරිච්ඡේදයේදී ඉගෙනගත්තා). switch-case යෙදූ විට MediaFunc දිස්වෙන ආකාරය පහත දැක්වේ:

```
====================================

#include "MediaPlayer.h"

void Player::MediaFunc(int x){
  switch(x){
    case 1:
     break;

   case 2:
     break;

    case 3:
     break;
  }
}

====================================
```

මෙහිදී switch-case මගින් සිදුවන්නේ MediaFunc function එකට pass කරනු ලබන **int x** argument එකෙහි අගය 1 ද 2 ද එසේ නැතිනම් 3 වේද යන්න පිරික්සීමයි.

13. දැන් අප මෙම switch-case හි case statements තුලට අදාල codes යෙදිය යුතුය. පහත දැක්වෙන්නේ switch-case තුල type කලයුතු පරිගණක කේතයන්ගෙන් කොටසකි.

```
=========================================================

#include "MediaPlayer.h"

void Player::MediaFunc(int x){
  switch(x){
   case 1:

     // if paused or playing, no menu
     if(Pause!=true && Play!=true){
      cout<< "Enter the path: " << endl;
      cin>> pathSrc;
      wsprintf(mediaCommand, "play %s", pathSrc);
      cout<<"\nPlaying..."<<endl;
     }
     Pause = false;
     Stop = false;
     Play = true;
     break;

   case 2:

      Pause = false;
      Stop = true;
      Play = false;
      break;

   case 3:

      Pause = true;
      Stop = false;
      Play = false;
       break;
  }
}

=========================================================
```

දැන් මෙම නව codes වලින් සිදුවන්නේ කුමක්දැයි හඳුනා ගැනීමට කරන උත්සාහය. ඉහලින්ම ඇති **case 1:** වෙතින් අරඹමු.

මුලින්ම ඇති **if** statement එක මගින් **Pause** හා **Play** යන boolean variables, **true** වේදැයි පරීක්ෂා කරන බව ඔබට පෙනේ. මෙමගින්, දැනට කිසියම් media file එකක් play වෙමින් හෝ pause වී හෝ පවතියි දැයි හඳුනාගැනීම සිදුවේ. එය සිදුවන්නේ මෙසේයි **:**

ඉහතින්ද ඔබ කියවූ පරිදි **Pause**, **Stop** හා **Play** variables වල කාර්යය කිසියම් මොහොතක Media Player එක සිදුකරන කාර්යය වාර්ථා කර තැබීමයි.

එනම්,

- Media Player එක file එකක් play කරමින් සිටිනවානම් **Play** variable එකෙහි අගය true විය යුතු අතර Pause හා Stop variables වල අගයන් false විය යුතුය.
- Media Player එක විසින් ධාවනය වෙමින් පැවති file එකක් pause කර ඇත්නම් **Pause** variable එකෙහි අගය true විය යුතු අතර අනෙක් variables යුගලෙහි අගයන් false විය යුතුය.

- Media Player එක කිසිත් නොකර නඳහස්ව ඇත්නම් **Stop** variable එකෙහි අගය true විය යුතු අතර අනෙක් variables යුගල false විය යුතුය.

දැන්, ඔබට ඉහත code එකෙහි පෙනෙන පරිදි සෑම case statement එකක් තුලම මෙම Pause, Stop හා Play variables වල අගයයන් අදාල ලෙස true හෝ false කර ඇත. එනම්, (**MediaFunc** function එක තුලට 1 අගය pass කල විට) **case 1** තුලදී සිදුවිය යුත්තේ file එකක් play කිරීම වන හෙයින්, එතුලදී **Play** හි අගය true කර අනෙක්වා false කර ඇත. එමෙන්ම (**MediaFunc** function එක තුලට 2 pass කල විට) **case 2** තුල සිදුවිය යුත්තේ file එකක් stop කිරීම නිසා එහිදී **Stop** හි අගය පමණක් true කර ඇත. අවසාන වශයෙන් (**MediaFunc** function එක තුලට 3 pass කල විට) **case 3** තුල file එකක් pause වීම සිදුවන නිසා **Pause** හි අගය true කර ඇත. (මෙම play, pause හා stop වීම සිදුවන්නේ කෙසේදැයි ඉදිරියේදී ඔබට දැකගත හැකිය)

එසේනම්, අප සලකා බලමින් සිටි case 1 තුල ඇති if statement එකින් සිදුවන්නේ කුමක්දැයි ඔබට වටහාගත හැක. එනම්, මෙම if තුල පරීක්ෂා කරන තත්වය සත්‍ය වන්නේ MediaPlayer එක විසින් file එකක් play කරමින් හෝ pause කර නොමැතිවිට හෙවත්, **Stop** හි අගය true වන විටදීයි. (මෙම if මගින් **Play** හා **Pause** හි අගයන් false දැයි පරීක්ෂා කිරීම වෙනුවට **Stop** හි අගය true වේදැයි යන්න ද පරීක්ෂා කල හැකියි.)

if මගින් පරීක්ෂා කරනු ලබන තත්වය සත්‍ය වේ නම් මීලගට වහාම සිදුවන්වන්නේ **"Enter the path:"** යන message එක Terminal window එක තුල print කරමින් file path එකක් වැඩසටහන පාවිච්චි කරන්නාගෙන් විමසීම බව ඔබට code එකෙහි ඊලග පේලිය දෙස බැලීමෙන් වැටහේ. එනම්, වැඩසටහන ඔබෙන් file path එකක් විමසන්නේ එය දැනටම file එකක් play කරමින් නොමැතිනම් හෝ pause කර නොමැති නම් පමණකි. මෙහිදී ඔබ, play කිරීම අවශ්‍ය media file එකෙහි file path එක console window එක තුලට type කර, Enter button එක press කල විට, එම type කල path එක cin මගින් **pathSrc** string එක තුලට ඇතුල් කරනු ලබන බව code එකෙහි ඊලග පෙලින් දිස්වේ (pathSrc නැමැති නම සාදාගෙන ඇත්තේ path සහ source යන වචන යුගල එක් කර, කෙටි කිරීමෙනි).

මීලගට හමුවන්නේ **wsprintf** නැමැති function එක සහිත කේතයයි. wsprintf function එක අඩංගු වන්නේ **Shlobj** library එක තුල බව ඔබ මීට කලින් දුටුවා. දැන් අප මෙම wsprintf function එක ක්‍රියා කරන්නේ කෙසේදැයි වටහා ගනිමු:

<u>wsprintf function</u>

int wsprintf (**char*** *buffer*, **char*** *format*, **char*** *source*)

මෙම function එක මගින් සිදුකරන්නේ *source* නැමැති text string එක format කර (අවශ්‍ය ආකාරයට සකසා) එය *buffer* නැමැති char array එක තුල ගබඩා කිරීමය.

<u>Arguments</u>

char* *buffer*:
Format කරන ලද text string එක ගබඩා කෙරෙන char array එකට යොමුවුන pointer එකක්.

char* *format*:
Text string එක format කල යුතු ආකාරය දැක්වෙන char array එකට යොමුවුන pointer එකක්. (මෙම format වීම සිදුවන ආකාරය මොහොතකින් සලකා බැලේ)

char* *source*:
Format කිරීමට ලක් වන char array එකට යොමුවුනු pointer එකකි.

wsprintf මගින් සිදුවන දේ පැහැදිලි කරගැනීමට පෙර එයට arguments pass කිරීම හා සම්බන්ධ කරුණු ගැන මඳක් සලකා බැලීම සුදුසුය. පහත දැක්වෙන කරුණු පිලිබඳ අවධානය යොමු කරන්න.

ඉහත ඔබ දුටු පරිදි wsprintf වෙත call කිරීමේදී අප එයට pass කල යුතු වන්නේ char arrays හෙවත් text strings සඳහා pointers ය. කිසියම් array එකකට pointer එකක් සාදාගත හැකි ක්‍රමය වන්නේ එම array එකෙහි index 0 ස්ථානයේ ඇති දත්තයේ memory address එක, අදාල data type එකින් යුතු pointer එකකට ආදේශ කිරීම බව ඔබ **“Pointers”** නැමැති පරිච්ඡේදයේ **“Arrays සඳහා pointers සෑදීම”** නම් කොටසේදී දුටුවා. ඉහත wsprintf function එකෙහි argument pass කිරීමේදී පාවිච්චි කල යුත්තේ මෙම මූලධර්මයයි. wsprintf තම සියලු arguments ලෙස ලබාගන්නේ char arrays සඳහා pointers ය. මේ නිසා අප එයට pass කල යුතු වන්නේ ඉහත කියූ පරිදි char arrays වල index 0 ස්ථාන වල addresses ය. මෙය හුදෙක් එම char array එකෙහි නම wsprintf තුලට pass කිරීමෙන් සිදු කල හැකිය. මක්නිසාදයක් array pointer එකක් සෑදීම, pointer එකකට හුදෙක් අදාල array එකෙහි නම ආදේශ කිරීමෙන් සිදුකල හැකි වීමයි (මෙය “Pointers” පරිච්ඡේදයෙහි දැක්වුණි). Code එකෙහි සිදුකර ඇත්තේ මෙයයි. ඔබ දැන් code එක දෙස බැලුවහොත් wsprintf වෙත එහි 1 වන හා 3 න් වැනි arguments ලෙස pass කර ඇත්තේ mediaCommand සහ pathSrc යන char arrays යුගලෙහි නම් බව දැකිය හැකියි.

නමුත් එහි 2 වැනි argument එක ලෙස කෙලින්ම text string එකක් යොදා ඇත. ඇත්තෙන්ම මෙහිදී සිදුවන්නේද ඉහත දෙයම වෙනස් ආකාරයකටයි. එහිදී 2 වන argument එක ලෙස variable එකක් භාවිතා නොකර සෘජුවම text string එකක් pass කර ඇත. (pass කර ඇති “play %s” යන්නෙහි අර්ථය කුමක්දැයි ඔබ මොහොතකින් දකිනු ඇත). මෙය, “play %s” යන අගය ගබඩා කරන ලද char array එකක address එකක් pass කිරීමට හරියටම සමානය.

wsprintf මගින් ඇත්තෙන්ම සිදුකරනු ලබන්නේ ඉහත *source* නමින් දී ඇති char pointer එකින් දක්වන char array එක තුල ඇති text string එක අවශ්‍ය අයුරින් format කර *buffer* නමින් දී ඇති pointer එකින් දක්වන char array එක තුල ගබඩා කිරීමය. මෙම format වීම සිදුවන්නේ *format* නමින් දී ඇති pointer එකින් දක්වන char array එක තුල ගබඩාව ඇති text string එකට අනුවය. එනම්, format මගින් දක්වන string එක සිදුකරන්නේ *source* නැමැති string එක format කරන ආකාරය wsprintf function එකට පෙන්වා දීමය. මෙවැනි text strings හඳුන්වනු ලබන්නේ **format-control strings** යන නමිනි. මෙය Media player වැඩසටහනෙහි “play %s” ලෙස දැකිය හැකිය. මෙහිදී format-control string එක ලෙස ගැනෙන්නේ **“ %s ”** යන කොටසය.

Format-control string එකක් හැමවිටම ඇරඹෙන්නේ **%** ලකුණිනි. ඉන්පසුව එන තනි character එකින් දැක්වෙන්නේ wsprintf හි තෙවැනි argument variable එක තුල ඇති දත්තයට සැලකිය යුතු ආකාරයයි. ඉහත format control string එකෙහි තනි character එක ලෙස s යෙදීමෙන් දැක්වෙන්නේ, 3න් වන argument එක string එකක් ලෙස සැලකිය යුතු බවයි. wsprintf function එක මගින් format – control string එකෙහි **%** සලකුණට පෙර ඇති string කොටස, එනම් “play” යන්න අගට **pathSrc** string එකෙහි අඩංගු text string එක අමුණා එය **mediaCommand** තුල ගබඩා කිරීම සිදුවේ. ඒ අනුව, උදාහරණයක් වශයෙන් ඔබ pathSrc තුලට ලබාදී ඇත්තේ `“D:\X-Files\track3.wmv”` යන file path string එක නම් wsprintf function එක මගින් එය `“play D:\X-Files\track3.wmv”` යන string එක බවට හරවා එය csCommnad තුල ගබඩා කරනු ඇත. එනම්, format-control string එක වන “%s” යන්න ඉවත්වී එතැනට wsprintf හි 3 වන argument එක වන string එක ආදේශ වේ. (mediaCommand තුල ගබඩා කල මෙම text string එකෙහි අර්ථය සහ ප්‍රයෝජනය අප මෙම වැඩසටහන ඉදිරියට ලියන විට ඔබට පැහැදිලි වනු ඇත.) wsprintf function එක execute වූ වහාම play වීම ඇරඹී ඇති බව දැක්වීමට “Playing…” යන්න print වේ.

හොඳයි. ඉහත කොටසේදී සිදුකල දේ නැවත කෙටියෙන් සඳහන් කලහොත්, අප සිදුකලේ**:** Media Player වැඩසටහන පාවිච්චි කරන්නා විසින් වැඩසටහන තුලට 1, 2 හෝ 3 අගයයන් ඇතුල් කලවිට, එම අගයයන්ට අනුව ක්‍රියාත්මක වීම පිණිස switch-case code එකක් සැකසීමය. එහිදී අප පලමු case statement එක තුලදී program එක පාවිච්චි කරන්නාගෙන් file path එකක් ලබා ගෙන, එම file path එක pathSrc තුල ගබඩා කලෙමු. ඉන්පසුව, **wsprintf** function එක මගින් “Play” යන string එක අගට pathSrc හි ඇති අගය(file path එක) අමුණා එය mediaCommand යන string එක තුල ගබඩා කලෙමු. මේ සියල්ල සිදුවන්නේ දැනට වැඩසටහන විසින් file එකක් play කරමින් හෝ pause කර නැති විට බවද if statement එකක් මගින් තහවුරු කලෙමු.

දැන් අප play කල යුතු media file එක පරිගණකයේ දෘඩ තැටියෙහි පවතින ස්ථානය දනිමු. එහි file path එක අප pathSrc variable එක තුල ගබඩා කර ඇත. දැන් කල යුත්තේ මෙම file එක ධාවනය කිරීම සඳහා අවශ්‍ය codes ලිවීමය. (මෙය සාර්ථක වීම සඳහා ඔබේ පරිගණකයට Sound card එකක් install කර තිබිය යුතු බවද, පරිගණකයට සම්බන්ධ කර ඇති speaker එකක් ඔබ සතුව තිබිය යුතු බවද මෙහිදී සඳහන් කිරීම අනවශ්‍යය)

පහත දැක්වෙන්නේ media file එකක් play කිරීම සඳහා අවශ්‍ය පරිගණක කේත අඩංගු කර ලියනු ලැබූ නව code එකයි.

```
============================================================

#include "MediaPlayer.h"

void Player::MediaFunc(int x){
  switch(x){
   case 1:

      // if paused or playing, no menu
      if(Pause!=true && Play!=true){
       cout<< "Enter the path: " << endl;
       cin>> pathSrc;
       wsprintf(mediaCommand, "play %s", pathSrc);
       cout<<"\nPlaying..."<<endl;
      }
      Pause = false;
      Stop = false;
      Play = true;
      break;

    case 2:

      Pause = false;
      Stop = true;
      Play = false;
      break;

     case 3:

      Pause = true;
      Stop = false;
      Play = false;
      break;
  }

  if(mciSendString(mediaCommand, NULL, 0, NULL)!=0){

  }
}

============================================================
```

ඉහතදී, නව function එකක් වන **mciSendString** ඔබට හමුවේ. පහත දැක්වෙන්නේ මෙම function එක පිලිබඳ හැඳින්වීමයි.

mciSendString function

mciSendString function එක මගින් සිදුවන්නේ එයට සපයනු ලබන media file එකට media commands(play, stop, pause, etc) සැපයීමයි.

int mciSendString (**char*** *command*, **char*** *returnInfo*, **int** *bsReturn*,
HANDLE *callback)*

Arguments

char* command**:**

මෙම string එකෙහි අඩංගු විය යුත්තේ media file එක සඳහා සපයන command string එකයි.

command string එක පහත format එකට, හෙවත් සැලැස්මට අනුව සැකසී තිබිය යුතුය:
" *MediaCommand, FilePath* " – එනම්, පලමුවෙන් අදාල command එක(play, stop, pause, etc) සඳහන් විය යුතු අතර දෙවනුව, එම command එක සැපයිය යුතු media file එකෙහි path එක සඳහන් විය යුතුය.

char* returnInfo:
int bsReturn:
HANDLE callback:
ඉහත දැක්වෙන arguments ත්‍රිත්වය අප මෙම වැඩසටහනේදී භාවිතා නොකරන හෙයින් ඒවා පිලිබඳ විස්තර මෙහිදී නොසලකා හරිනු ලැබේ. එම arguments භාවිතා නොකරන බව දැක්වීමට, code එකෙහි **mciSendString** function call එක තුලදී ඒවා pass කල යුතු ස්ථාන සඳහා NULL හෝ 0 යන අගයයන් යොදා ඇත. මෙම අගයන් යෙදීමෙන් අදහස් කරන්නේ arguments සඳහා pass කෙරෙන්නේ ශුන්‍ය වූ අගයන් බවයි.

දැන් code එකෙහි mciSendString function එකට call කර ඇති ආකාරය බලන්න. (call වීම if statement එකක් තුල සිදුවන්නේ ඇයි දැයි පසුව විස්තර වේ) පලමුවෙන් එයට pass කර ඇත්තේ දැන් media command එක ගබඩා කර ඇති mediaCommand string එකයි. මෙය ඔබ ඉහතදී දුටු පරිදිම පලමුව media command එකකින්ද, දෙවනුව file path එකකින්ද සමන්විතය (උදා- `"play D:\X-Files\track3.wmv").` mciSendString function එක execute වූ විගස *command* string එකෙහි දැක්වෙන file එක, *command* string එකෙහි ඇති media command එක අනුව play, stop හෝ pause වනු ඇත.

මේ අනුව, code එක තුල ඇති swich-case block එකෙහි පලමු case statement එක true වන්නේ නම්, mediaCommand string එකෙහි අඩංගු media command එක "play" යන්න බවට පත් වන නිසා, එවිට mciSendString function එක මගින් අදාල file එක play කිරීම අරඹනු ඇත.

mciSendString function එකට call කර ඇත්තේ if statement එකක් තුල සිට බව නිරීක්ෂණය කරන්න. මෙයට හේතුව නම් mciSendString function එක නිසි පරිදි ක්‍රියාත්මක නොවනවා නම් එය එසැනින් ම හඳුනාගැනීමට මෙමගින් හැකියාව ලැබෙන නිසයි. mciSendString function එක මගින් return කරනු ලබන්නේ integer එකක් බව ඉහත එහි හැඳින්වීම තුලදී ඔබ දකින්නට ඇති (Functions මගින් අගයන් return කරනු ලබන ආකාරය පිලිබඳව අප **"Functions"** පරිච්ඡේදයේදී සලකා බැලුවෙමු). mciSendString සාර්ථකව ක්‍රියාත්මක වුවහොත්, එනම් එයට ඔබ ලබා දෙන media file එක play කිරීමට හැකි වුවහොත්, එය return කරනු ලබන්නේ 0 අගයයි. mciSendString හි ක්‍රියාකාරීත්වය අසාර්ථක වුවහොත්, එනම් එය file එක play කිරීමට අපොහොසත් වුවහොත් එය මගින් return කරනු ලබන්නේ 0 ට වැඩි අගයකි. මෙම 0 ට වැඩි අගය, file එක play කිරීමට දරන උත්සාහයේදී උද්ගත වන ගැටලුවට හෙවත් Runtime error එක මත රඳා පවතියි. (Runtime errors යන නමින් හැඳින්වෙන්නේ කිසියම් පරිගණක වැඩසටහනක් ක්‍රියාත්මක වන මොහොතේදී එම වැඩසටහනෙහි ක්‍රියාකාරීත්වය අඩාල වීමට හේතුවන errors ය). මේහිදී mciSendString function එක ක්‍රියාත්මක නොවීමට විවිධ හේතඩන් බලපෑ හැකියි. උදාහරණයක් වශයෙන් ඔබ සැපයූ file එක media file එකක් නොවීමට ඉඩ ඇත. එසේත් නැතිනම් එම file එකෙහි media format එක mciSendString function එක නොහඳුනනවා විය හැක. නැතහොත් හුදෙක් ඔබ type කරන ලද file path එකෙහි දෝශයක් තිබෙනවා විය හැක.

if statements එක මගින් සිදුවන්නේ එහි වරහන් තුල ඇති තත්වය සත්‍ය වුවහොත්, එහි සඟල වරහන් තුල ඇති code එක execute කිරීම බව ඔබ **"පරිගණක වැඩසටහනක ක්‍රියාකාරීත්වය පාලනය කිරීම"** යන පරිච්ඡේදයේදී දුටුවා. මෙම "සත්‍ය වීම" යනුවෙන් ඇත්තවශයෙන්ම හැඳින්වෙන්නේ if තුල ඇති expression එකින් 0 ට වඩා වැඩි අගයක් ඉපදීමයි. එසේනම් "අසත්‍ය වීම" යනු expression එකින් 0 අගය නිපදවීමයි (මෙය Boolean data type එක පිලිබඳව සාකච්චා කිරීමේදී සඳහන් විණි). එසේනම් mciSendString නිසි පරිදි ක්‍රියා නොකල විගස මෙම if හි වරහන් තුල අගය true වන නිසා (mciSendString මගින් 0 ට වැඩි අගයක් නිපදවෙන නිසා) if යටතේ ඇති codes, execute හෙවත් ක්‍රියාත්මක වනු ඇත. මෙහෙයින්, අප මීලගට කල යුත්තේ මෙම if, true වේ නම් හෙවත් file එක නිසි පරිදි ධාවනය කිරීමට නොහැකි නම්, ඊලගට වැඩසටහන විසින් කල යුත්තේ කුමක්දැයි තීරණය කිරීමය. මෙම වැඩසටහනෙහි සරලත්වය උදෙසා, අප මෙහිදී සිදු කිරීමට යන්නේ සරල message box එකක් වැඩසටහන පාවිච්චි කරන්නා හට දිස් කරවීමය. මෙම message box එක තුල `"Sorry, can't play your file."` යන message එක දිස්විය යුතුය. මේසඳහා දැන් if තුල අන්තර්ගත වියයුතු codes මොනවාදැයි පරීක්ෂා කර බලමු:

```
============================================================

#include "MediaPlayer.h"

void Player::MediaFunc(int x){
  switch(x){
   case 1:

     // if paused or playing, no menu
     if(Pause!=true && Play!=true){
      cout<< "Enter the path: " << endl;
      cin>> pathSrc;
      wsprintf(mediaCommand, "play %s", pathSrc);
      cout<<"\nPlaying..."<<endl;
     }
     Pause = false;
     Stop = false;
     Play = true;
     break;

   case 2:

     Pause = false;
     Stop = true;
     Play = false;
     break;

    case 3:

     Pause = true;
     Stop = false;
     Play = false;
     break;
  }

  if(mciSendString(mediaCommand, NULL, 0, NULL)!=0){
   system("cls");
   MessageBox(NULL,"Sorry, can't play your file.","HALT!",0);
   Pause = false;
   Stop = false;
   Play = false;
  }
}

  ========================================
```

if තුල පලමුවෙන්ම සිදුවන්නේ Terminal window එකෙහි දැනට දිස්වන සියලු දේ මකා දැමීමය. මෙය **system** නමින් ඇති function එක මගින් සිදුවේ. **system** function එක තම argument එක ලෙස ලබාගන්නේ **DOS** commands වේ. (DOS commands යනු ඔබ Disk Operating System එක යටතේ පරිගණකයක් ක්‍රියාකරවන විට එයට සැපයිය යුතු විධානයන්ය). **system** function එක මගින් සිදුකරන්නේ එයට pass කරනු ලබන DOS command එකට අදාල DOS ක්‍රියාකාරීත්වය සිදු කිරීමයි. අපගේ වැඩසටහනෙහි system function එක තුලට pass කර ඇත්තේ **“cls”** නැමැති DOS විධානයයි. ඔබට DOS commands පිලිබඳ දැනුමක් ඇත්තේ නම්, **“cls”** යනු Terminal window එකෙහි දිස්වන දෑ මකාදමන(clear screen) විධානය බව දන්නවා ඇති. මේ නිසා මෙහිදී සිදුවන්නේ පෙර කී පරිදිම මේ වන විට වැඩසටහන මගින් Terminal window එක තුල දිස්වෙන සියල්ල මකා දැමීමයි.

මීලඟට ඇති code එක පෙර සඳහන් message box එක දිස් කරනු ලබන පරිගණක කේතයයි. පහත දැක්වෙන්නේ **MessageBox** function එකෙහි හැඳින්වීමයි.

MessageBox function

MessageBox function එක මගින් සිදුවන්නේ එයට සපයනු ලබන message string එක අන්තර්ගත වන පරිදි **Windows message box** එකක් විවෘත කිරීමයි.

int MessageBox (**HWND** *hwnd,* **char*** *text,* **char*** *caption,*
int *boxType)*

Arguments

HWND hwnd**:**
මෙම වැඩසටහනෙහිදී මෙම argument එක පාවිච්චියට නොගැනෙන නිසා මෙය නොසලකා හැරේ ։code එකෙහි මෙම argument එක pass කල යුතු ස්ථානයට NULL අගය යොදා ඇත*.

char* text**:**
මෙම char array එකෙහි අඩංගු විය යුත්තේ message box එක තුල දිස්විය යුතු text string එකයි.

char* cpation**:**
මෙය message box එකෙහි title bar එකෙහි දිස්විය යුතු title එකයි.

int boxType**:**
මෙම integer එකින් දැක්වෙන්නේ message box එකෙහි type එක හෙවත් වර්ගයයි.

Windows message boxes වර්ග කීපයකි. මේ සෑම message box type එකකටම int අගයක් ඇත. boxType argument එකින් දැක්වෙන්නේ මෙම වර්ගයයි. පහත දැක්වෙන්නේ message boxes වර්ග සහ ඒවාට අදාල boxType අගයයන්ය.

Message box වර්ගය		boxType අගය
Ok button එක පමණක් සහිත message box එක	-	0
Ok සහ Cancel buttons සහිත	-	1
Abort, Retry සහ Ignor buttons සහිත	-	2
Yes, No සහ Cancel buttons සහිත	-	3
Yes, No buttons සහිත	-	4
Retry සහ Cancel buttons සහිත	-	5
Cancel, Try again සහ Continue buttons සහිත	-	6

වැඩසටහනෙහි අප boxType argument එක වෙත pass කර ඇත්තේ 0 නිසා මෙහිදී විවෘත වනු ඇත්තේ OK button එකින් පමණක් සමන්විත message box එකයි. එය තුල, අප විසින් *text* argument එක සඳහා pass කරනු ලබන string දිස් වනු ඇති අතර, එහි title එක වනු ඇත්තේ *caption* argument එක වෙත pass කරනු ලබන string එකයි.

මීලඟට code එකෙහි සිදුවී ඇත්තේ Pause, Play සහ Stop අගයන් false වීමයි. මෙසේ කිරීමට හේතුව නම්, දැන් file එක play කිරීමට Media Player වැඩසටහන අසමත් වීම හේතුවෙන් නව file එකක් load කිරීම සඳහා මෙම variables reset කිරීම (නැවත මුල් අගයයන්ට ගෙන ඒම) සුදුසු හෙයිණි.

මෙතෙක් අප ලියා ඇති වැඩසටහනින් සිදුවන කාර්යයන් කෙටියෙන් සඳහන් කලහොත්**:**
අප switch-case statement එකක් මගින්, MediaFunc function එකට pass කරනු ලබන අගය කුමක්දැයි බැලීමු. Pass කරනු ලැබූ අගය 1 වූ විට ක්‍රියාත්මක වීම සඳහා case 1 තුල codes ලිවුවෙමු. එම code එකින් සිදුවූයේ යම් අවස්ථාවක file එකක් play වෙමින් හෝ pause වී නොමැතිනම් වැඩසටහන පාවිච්චි කරන්නාට

වැඩසටහනට නව file එකක් ඇතුල් කිරීමට ඉඩ ලබා දීමත්, file එක play කිරීම සඳහා අවශ්‍ය command එක සැකසීමත්, Play variable එකෙහි අගය true කිරීමත්ය. නමුත් ඇත්තෙන්ම එම command එක ලබාගෙන file එක play කරනු ලබන්නේ switch-case block එකට පිටතින් ඇති mciSendString function එක මගිනි. මෙම function එකට call කිරීමේදී function එක නිසි පරිදි ක්‍රියාත්මක නොවුනහොත් සිදුකල යුතු දෙය අප if statement එකක් මගින් තීරණය කලෙමු.

මෙතැන් සිට අප සිදුකල යුතු දේ නම් case 2 සහ case 3 සඳහා අවශ්‍ය කේතයන් ලිවීමය. case 2 තුල ලිවිය යුත්තේ media file එකක් stop කිරීම සඳහා වන කේතයන් වන අතර, case 3 තුල media file එකක් pause කිරීමට අදාල කේතයන් ලිවිය යුතුය. මෙම නව කේතයන් එක් කල විට Media Player වැඩසටහන පහත පරිදි දිස්වෙනු ඇත. අලුතින් එක් කර ඇති කේතයන් තද කලු අකුරින් දැක්වේ:

```
============================================================

#include "MediaPlayer.h"

void Player::MediaFunc(int x){
  switch(x){
   case 1:

      // if paused or playing, no menu
      if(Pause!=true && Play!=true){
       cout<< "Enter the path: " << endl;
       cin>> pathSrc;
       wsprintf(mediaCommand, "play %s", pathSrc);
       cout<<"\nPlaying..."<<endl;
      }else{
         if(Pause){
           cout<<"\nResumed... "<<endl;
           wsprintf(mediaCommand, "play %s", pathSrc);
         }
      }
      Pause = false;
      Stop = false;
      Play = true;
      break;

    case 2:
      if(!Stop){
        wsprintf(mediaCommand, "stop %s", pathSrc);
      }
      cout<<"\nStoped... "<<endl;
      Pause = false;
      Stop = true;
      Play = false;
      break;

     case 3:
      if(!Pause){
        wsprintf(mediaCommand, "pause %s", pathSrc);
        cout<<"\nPaused... (a- resume/s>a- New file)\n"<<endl;
      }
      Pause = true;
      Stop = false;
      Play = false;
      break;
   }

  if(mciSendString(mediaCommand, NULL, 0, NULL)!=0){
   system("cls");
   MessageBox(NULL,"Sorry, can't play your file.","HALT!",0);
   Pause = false;
   Stop = false;
   Play = false;
  }
 }
```

==========

පලමුව සිදුකර ඇත්තේ case 1 හි if වෙත else statement එකක් එකතු කිරීමය. මෙය සිදුකර ඇත්තේ Media Player එක file එකක් play කරමින් හෝ pause කර ඇතිනම් (Play හෝ Pause හි අගය true නම්) එම අවස්ථාව හඳුනාගැනීම සඳහාය. එම else තුල මීලගට, if statement එකක් මගින් file එකක් pause වී ඇත්දැයි (Pause හි අගය true දැයි) පරීක්ෂා කෙරේ. මෙය සත්‍ය නම් if තුල ඇති cout මගින් "Resumed…" යන්න print කරන්නේ අප මීලගට සිදුකරන්නේ pause වී ඇති file එක යලි ධාවනය කිරීම ඇරඹීම වන නිසයි. Resume කිරීමට අවශ්‍ය command string එක මීලීගට ඇති wsprintf function එක මගින් නිපදවයි. File එකක pause කිරීමෙන් පසුව resume වීමට mciSendString function එකට pass කල යුතු command එකද "Play *filename*" යන format එකින් ම යුතුවේ. එම command එක මගින් audio / video file එක නැවතුනු තැන සිට ධාවනය වීම ඇරඹෙනු ඇත. නමුත් මෙහිදී file එක ඇත්තේ play වෙමින් නම් ඇතුල් if එක false වීම නිසා ඉහත සඳහන් කිසිවක් සිදුනොවනු ඇත. අවසානයේදී Play variable එකෙහි අගය පමණක් true කර ඇත්තේ දැනට file එකක් play වෙමින් ඇති බව වාර්ථා කර තැබීමටය.

case 2 තුලදී සිදුවන්නේ Media Player එක තුල කිසියම් Media file එකක් play වෙමින් හෝ pause වී ඇතිනම් පමණක්, එය stop කිරීමට අවශ්‍ය command string එක wsprintf මගින් නිර්මාණය වීමයි. එම string එක මෙම format එක ගනු ඇති බව ඔබට වැටහේ**:** "Stop *fileName*" (wsprintf මගින් මෙහිදී සිදුකරන්නේ "stop" යන string එක අගට pathSrc අමුණා එය mediaCommand තුල ගබඩා කිරීම වේ). ඊලගට එය "Stoped…" යන message එක print වෙනු ඇත. ඉන්පසු, කලයුතු පරිදිම, දැන් file එක stop වී ඇති බව වාර්ථා කිරීමට, Stop හි අගය පමණක් true කර ඇත.

case 3 statement එක තුලදී, file එකක් pause වී ඇතඳැයි බැලෙන අතර, එසේ නැති නම් (pause වී ඇති file එකක් නැවත pause කිරීමේ අර්ථයක් නැති නිසා) pause කිරීමට අවශ්‍ය command string එක පෙර පරිදිම නිපදවීම සිදුවෙයි. එහි format එක**:** "pause *fileName*" වනු ඇත.

හොඳයි. දැන් අප මෙම වැඩසටහන නිමවීමේ අවසන් අධියරයට පැමිණ ඇත්තෙමු. ඔබ විසින් දැන් කල යුතුව ඇත්තේ එක් දෙයකි. එනම්, මෙම වැඩසටහන ධාවනය කිරීමට අත්‍යවශ්‍ය වන main function එක ලිවීමය.

main function එකෙහිදී අප සිදු කරන්නේ, Media Player වැඩසටහන ක්‍රියාත්මක කරවන්නාගෙන් අදාල input එක ලබා ගැනීමය. මෙම input එක මගින් Media Player වැඩසටහන මගින් සිදුකල යුතු දෙය තීරණය වනු ඇත. මෙහිදී input එක ලෙස ක්‍රියාකරවන්නා විසින් වැඩසටහනට සැපයිය යුත්තේ character එකකි. එම characters අප පහත පරිදි භාවිතා කරමු**:**

- 'a' character එක වැඩසටහනට ඇතුල් කරනු ලැබුවහොත් Media Player වැඩසටහන විසින් media file එකක් play කිරීමට අදාල ක්‍රියාමාර්ගය ගත යුතුය. මෙම ක්‍රියාමාර්ගය නම් **MediaFunc** function එකට 1 යන integer අගය pass කිරීම වනු ඇත.
- 's' character එක වැඩසටහනට ඇතුල් කරනු ලැබුවහොත් Media Player වැඩසටහන විසින් media file එකක් stop කිරීමට අදාල ක්‍රියාමාර්ගය ගත යුතුය. මෙම ක්‍රියාමාර්ගය නම් **MediaFunc** function එකට 2 යන integer අගය pass කිරීම වනු ඇත.
- 'd' character එක වැඩසටහනට ඇතුල් කරනු ලැබුවහොත් Media Player වැඩසටහන විසින් media file එකක් pause කිරීමට අදාල ක්‍රියාමාර්ගය ගත යුතුය. මෙම ක්‍රියාමාර්ගය නම් **MediaFunc** function එකට 3 යන integer අගය pass කිරීම වනු ඇත.
- ' f ' character එක වැඩසටහනට ඇතුල් කරනු ලැබුවහොත් Media Player වැඩසටහන නිමවිය යුතුය.

දැන් අප මෙම main function එක ඇතුලත් කරන ලද සම්පූර්ණ, ක්‍රියාකාරී Media Player පරිගණක වැඩසටහන නරඹමු**:**

==========

```
#include "MediaPlayer.h"

void Player::MediaFunc(int x){
  switch(x){
   case 1:

    // if paused or playing, no menu
```

```
   if(Pause!=true && Play!=true){
     cout<< "Enter the path: " << endl;
     cin>> pathSrc;
     wsprintf(mediaCommand, "play %s", pathSrc);
     cout<<"\nPlaying..."<<endl;
     }else{
        if(Pause){
          cout<<"\nResumed... "<<endl;
          wsprintf(mediaCommand, "play %s", pathSrc);
        }
     }
     Pause = false;
     Stop = false;
     Play = true;
     break;

   case 2:
     if(!Stop){
       wsprintf(mediaCommand, "stop %s", pathSrc);
     }
     cout<<"\nStoped... "<<endl;
     Pause = false;
     Stop = true;
     Play = false;
     break;

    case 3:
     if(!Pause){
       wsprintf(mediaCommand, "pause %s", pathSrc);
       cout<<"\nPaused... (a- resume/s>a- New file)\n"<<endl;
     }
     Pause = true;
     Stop = false;
     Play = false;
     break;
  }

  if(mciSendString(mediaCommand, NULL, 0, NULL)!=0){
   system("cls");
   MessageBox(NULL,"Sorry, can't play your file.","HALT!",0);
   Pause = false;
   Stop = false;
   Play = false;
  }
}

void main(){
  char c;
  Player pl;
  while(c!='f'){
    cout<<"\nWhat you wanna do? \na - Play.\ns - Stop.\nd –
        Pause.\nf - quit.\n"<<endl;
    c=getch();
    if(c=='a'){ // play
      pl.MediaFunc(1);
    }else if(c=='s'){ // stop
       pl.MediaFunc(2);
    }else if(c=='d'){ // pause
       pl.MediaFunc(3);
    }
  }
  cout<<"\nBye!\n"<<endl;
  exit(0);

}
```

==========

Main හි සිදුවන්නේ කුමක්ද? පලමුවෙන් එහි **char** c variable එක declare කර ඇත. මෙම variable එක අප පාවිච්චි කරන්නේ වැඩසටහන් ක්‍රියාකරුගෙන් ලබාගන්නා input character එක ගබඩා කිරීමටය. ඉන්පසු සිදුවන්නේ **pl** නමින් Player class එකෙහි instance එකක් හෙවත් **Player** object එකක් සෑදීමයි (Objects, classes, instances ආදිය පිලිබඳව අප **"Object Oriented Programming"** පරිච්ඡේදයේදී සලකා බැලුවෙමු).

වැඩසටහන අවසන් කිරීමට නම් ' f ' character එක වැඩසටහනට ඇතුල් කල යුතු බව ඉහත සඳහන් විණි. එමෙන්ම **c** variable එකෙහි අන්තර්ගත වන්නේ වැඩසටහනට ඇතුල් කරන ලද character එක බවද ඉහත දැක්විණි. එසේනම් වැඩසටහන අවසන් කිරීමට, එය පාවිච්චි කරන්නාට අවශ්‍ය දැයි දැනගැනීමට කල යුතු වන්නේ නිරන්තරයෙන්ම මෙම c හි අගය පරීක්ෂාවට ලක් කිරීමයි. මෙය main තුල සිදුවන්නේ **while** loop එකක් මගිනි. ඔබට code එකෙහි දැකිය හැකි පරිදි while මගින් එහි සෑම loop එකකදීම c හි අගය ' f ' දැයි පරීක්ෂා කෙරේ. අගය ' f ' නොවේ නම් loop එක නොනැවතී ක්‍රියාත්මක වනු ඇති අතර අගය ' f ' වේ නම් loop එක බිඳි පහලින් ඇති cout command එක ක්‍රියාත්මක වී "Bye!" යන්න print කරමින් වැඩසටහන අවසන් වන බව ක්‍රියාකරුට දන්වනු ලැබේ. ඒ සමගම execute වන **exit()** function එක මගින් වැඩසටහන අවසන් කරනු ඇත. exit function එක මගින් ඉතා සරල ලෙස පරිගණක වැඩසටහනකින් ඉවත් වීමට ඔබට අවස්ථාව ලැබේ. මෙහිදී එයට pass කර ඇත්තේ 0 අගය බව ඔබට පෙනේ. මෙම අගයෙන් දැක්වෙන්නේ වැඩසටහන තුල exit function එක පාවිච්චියට ගෙන ඇති අවස්ථා ගණනයි. එනම්, ඔබ මෙම වැඩසටහනෙහි තවත් ස්ථානයකදී exit function එක වෙත call කරනවා නම් එහිදී එයට pass කල යුත්තේ 1 අගයයි. තවත් තැනකදී call කරනවා නම් pass කල යුතු අගය 2 යි. මෙලෙස call කරනු ලබන අවස්ථා ගණන සමානව pass කලයුතු අගයද ඉහල යයි.

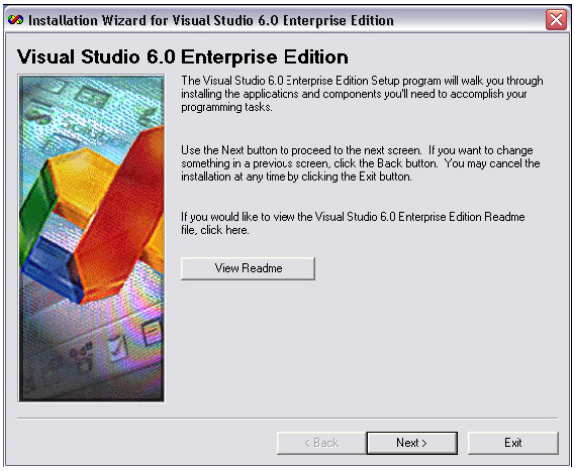
c හි අගය ' f ' නොවීමෙන් loop එක ක්‍රියාත්මක වන්නේ නම්, පලමුව සිදුවන්නේ වැඩසටහනෙහි ප්‍රධාන menu එක දිස්වීමය (රූපය10.1). මෙම menu එකින් වැඩසටහන පාවිච්චි කරන්නාට දක්වනු ලබන්නේ Media Player වැඩසටහන ක්‍රියාත්මක කරවිය යුතු ආකාරයයි. ඉන්පසුව හමුවන්නේ **getch()** function එකය. මෙම function එක මගින් සිදුකරන්නේ console window එක හරහා ඔබගෙන් කිසියම් input එකක් ලබා ගැනීමය. එලෙස ලබාගන්නා input data එක, getch function එක මගින් ආපසු return කරයි. Code එකෙහි සිදුකර ඇත්තේ මෙම return කරනු ලබන දත්තය c variable එක තුලට ලබා ගැනීමය. මීලගට සිදුවන්නේ if-else if statements කීපයක් මගින් එම ලබාගන්නා ලද දත්තය පරීක්ෂා කිරීමයි. පලමුවෙන්ම ඇති if මගින් c හි අගය ' a ' වේදැයි පරීක්ෂා කෙරෙන අතර එය සත්‍ය නම් **pl** object එක පාවිච්චි කිරීමෙන් **MediaFunc** function එකට 1 අගය pass කරමින් call කෙරෙනු ඇත. ඒ අනුව මෙහිදී MediaFunc function එක ක්‍රියාත්මක වීමෙන් ඉහත පැහැදිලි කරනු ලැබූ පරිදි, වැඩසටහන පාවිච්චි කරන්නාගෙන් media file එකක් ලබාගෙන එය play කරනු ඇත. ඉන්පසු ඇති else if statements යුගලින් පිලිවෙලින් 's' හා 'd' අගයන් සඳහා c පරීක්ෂාවට ලක් කෙරෙන අතර පිලිවෙලින් 2 හා 3 pass කරමින් MediaFunc වෙත call කෙරේ. ඒ අනුව පිලිවෙලින් media file එක stop හෝ pause වෙනු ඇත.

Appendix

Visual C++ 6.0 ස්ථාපනය (Install) කිරීම

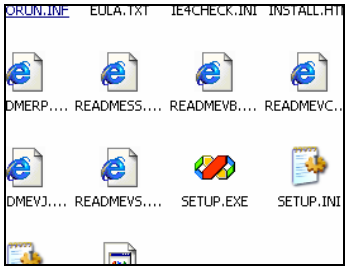
පහත දැක්වෙන ආකාරයට Visual C++ 6.0 install කරන්න:

- ඔබ මිලදී Visual C++ 6.0 ස්ථාපනය කිරීමේ සංයුක්ත තැටිය (CD) ඔබේ පරිගණකයේ සංයුක්ත තැටි ධාවකයට (CD driver) ඇතුල් කරන්න. එවිට මඳ වේලාවකින් 11.1 රූපයේ දැක්වෙන dialog box එක ස්වයංක්‍රියව විවෘත වනු ඇත.



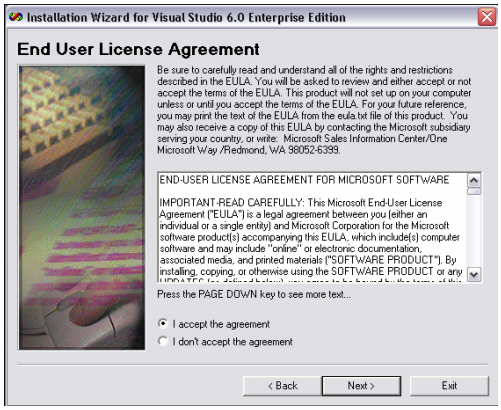
(රූපය 11.1)

- ඉහත Dialog box එක ස්වයංක්‍රියව විවෘත වූයේ නැතිනම් සංයුක්ත තැටියට පිවිසී එහි ඇති Setup.exe නැමැති file එක ක්‍රියාත්මක කරවන්න (11.2 රූපයේ දැක්වෙන්නේ එම file එකයි). එවිට ඉහත Dialog box එක විවෘත වේ.



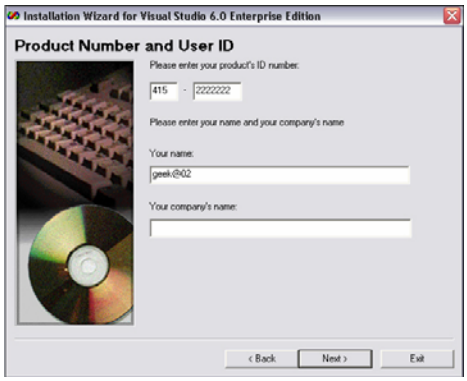
(රූපය 11.2)

- රූපය 1.1 dialog box එකෙහි “Next” button එක click කරන්න. එවිට රූපය 11.3 හි දැක්වෙන dialog box එක විවෘත වේ.



(රූපය 11.3)

- මෙහි අන්තර්ගත “I accept the agreement” නැමැති radio button එක select කොට Next button එක click කරන්න. එවිට මතුවන්නේ රූපය 11.4 හි දැක්වෙන dialog box එකයි.

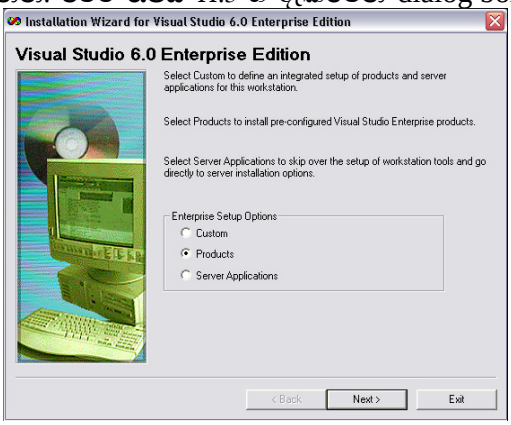


(රූපය 11.4)

- ඉහත dialog box එකෙහි ඉහලින්ම ඇති text box එක තුලට ඔබේ CD Key එක ඇතුල් කල යුතුය. මෙය බොහෝවිට ඔබ මිලදී ගත් Visual C++ සංයුක්ත තැටි කවරයේ සඳහන්ව තිබෙනු ඇත.

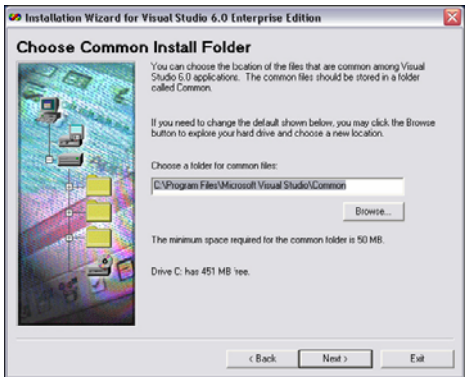
- ඊට පහතින් ඇති text box එකට ඔබේ නම ඇතුල් කරන්න.

- දැන් Next button එක click කරන්න. එවිට රූපය 11.5 හි දැක්වෙන dialog box එක විවෘත වනු ඇත.



(රූපය 11.5)

- මෙහි ඇති “Products” නැමැති option එක select කොට Next button එක select කරන්න. එවිට රූපය 11.6 හි දැක්වෙන dialog box එක විවෘත වේ.



(රූපය 11.6)

- මෙහි Next button එක click කරන්න. එවිට දිස්වෙනු ඇත්තේ රූපය 11.7 හි දැක්වෙන dialog box එකයි. එහි “Visual C++ Enterprise Edition” නැමැති check box එක select කර එය Next කරන්න.

(රූපය 11.7)

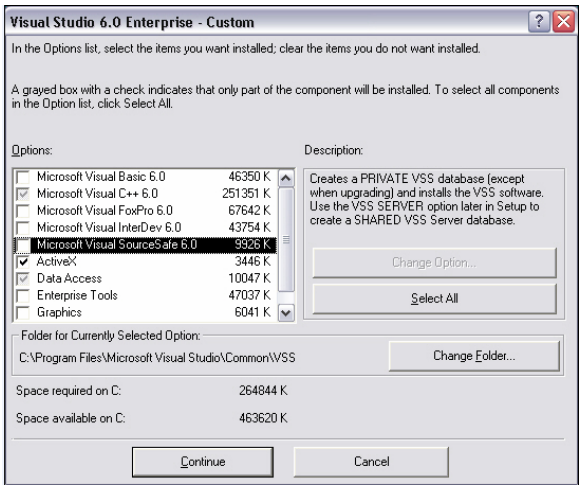
- එවිට 11.8 හි දැක්වෙන dialog box එක විවෘත වේ.

(රූපය 11.8)

- මෙහි Continue button එක click කරන්න. එවිට මතුවෙන, රූපය 11.9 හි දැක්වෙන Message box එක OK කරන්න.

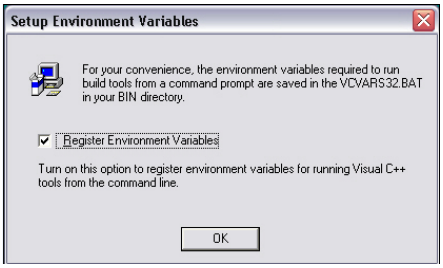
(රූපය 11.9)

- එවිට රූපය 11.10 හි දිස්වෙන dialog box එක විවෘත වනු ඇත.



(රූපය 11.10)

- ඔබට අවශ්‍ය නම් මෙහි options යටතේ ඇති check boxes වලින් Microsoft Visual C++ 6.0, ActiveX සහ Data Access යන ඒවා හැර අනෙක් check boxes සියල්ල clear කර දැමුවාට වරදක් වන්නේ නැත. අවසානයේදී Continue button එක click කරන්න. මෙවිට රූපය 11.11 දැක්වෙන dialog box එක දිස්වුවහොත් එය OK කරන්න.



(රූපය 11.11)

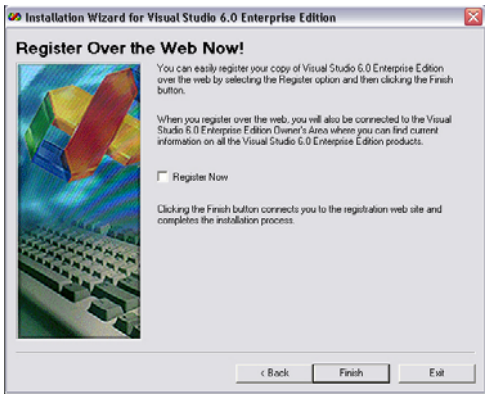
- එවිට රූපය 11.12 හි දැක්වෙන තිරය දිස්වී Visual C++ ස්ථාපනය වීම ඇරඹෙනු ඇත.

(රූපය 11.12)

- ඔබේ පරිගණකය සතු වේගය අනුව මෙම ස්ථාපනය වීමට ගන්නා කාලය වෙනස් වෙනු ඇත. ස්ථාපනය නිමවූ පසු මතුවෙන message boxes සියල්ල OK කල පසු රූපය 11.13 හි දැක්වෙන dialog box එක විවෘත වේ. එය Next කරන්න.

(රූපය 11.13)

- එවිට දිස්වන, රූපය 11.14 හි දැක්වෙන dialog box එකෙහි Register now නැමැති check box එක uncheck කර "Finish" button එක click කරන්න.



(රූපය 11.14)

ඔබ දැන් Visual C++ ස්ථාපනය කර අවසානයි. දැන් ඔබට එය Desktop එක මතින් හෝ Start menu එකෙහි "All Programs > Microsoft Visual Studio 6.0 > Microsoft Visual C++ 6.0" යන shortcut එක මගින් ක්‍රියාත්මක කල හැකි වනු ඇත.

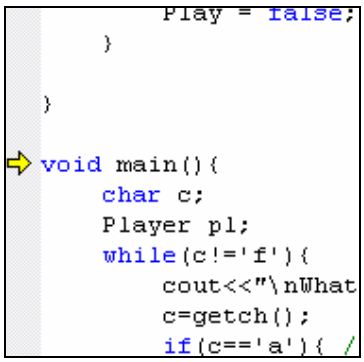
Debug කිරීම සහ Errors හඳුනාගැනීම

Program එකක් debug කිරීම යනුවෙන් හැඳින්වෙන්නේ කිසියම් පරිගණක වැඩසටහනක් ක්‍රියාත්මක වන ආකාරය නිරීක්ෂණය කිරීමට හා එහි අඩංගු දෝෂ හඳුනාගැනීමට භාවිතා කරන ක්‍රමවේදයයි. මෙම debugging පහසුකම ඔබට සපයනු ලබන්නේ debugger මෘදුකාංගය මගිනි. Visual C++ තුල debugger software එකක් අන්තර්ගතයි. Visual C++ හි ඇති මෙම පහසුකම ප්‍රයෝජනයට ගැනීමට නම් compile වීම සිදුවන, programming errors නොමැති පරිගණක වැඩසටහනක් අන්තර්ගත වන Visual C++ project එකක් තිබිය යුතුය.

වැඩසටහනක් debug කිරීමට නම් ප්‍රථමයෙන් සිදුකල යුත්තේ Visual C++ project එකක් load කර ගැනීමයි. ඉන්පසු Visual C++ තුල ඇති **Debug** window එක විවෘත කල යුතුය. එය විවෘත කිරීමට, Visual C++ හි menu bar එක මත right click කර, මතුවෙන pop-up menu එකින් "debug" නැමැති අයිතමය select කරන්න. රූපය 12.1 හි දැක්වෙන්නේ මෙම debug window එකයි.

(රූපය 12.1)

දැන් මෙම debug window එකෙහි ඇති **step into** සහ **step over** buttons පාවිච්චි කිරීමෙන් ඔබට පරිගණක වැඩසටහනක් ක්‍රියාත්මක වන ආකාරය නැරඹිය හැකිය. ඔබ step into හෝ step over button එක එබූ විට Visual C++ තුල **Program execution pointer** එක නමින් හැඳින්වෙන උපාංගය මතුවෙනු ඇත. 12.2 රූපයේ කහ පැහැති ඊතලයක් ලෙස දැක්වෙන්නේ program execution pointer එකයි.



(රූපය 12.2)

කිසියම් මොහොතකදී වැඩසටහනක ක්‍රියාකාරී ලක්ෂය දක්වන්නේ මෙම program execution pointer එක මගිනුයි. ඔබට step into button එක මගින්, function call එකක් ඇති ස්ථානයක සිට එම function එක අභ්‍යන්තරයට ගමන් කර, එම function එක තුල ඇති codes ක්‍රියාකාරී වන ආකාරය දැකගත හැකි අතර, step over button එක මගින් function එක තුලට ගමන් නොකර එයට උඩින් පැන, ක්‍රමලේඛණයක ඊලඟ කොටස ක්‍රියාකාරී වනු නැරඹිය හැකිය. ඉහත දැක්වෙන ඊතලය මගින් program එකක ඔබ සිටින ස්ථානය හැමවිටම දැනගත හැක. මෙය තෙරුම් ගැනීමට නම් ඔබ කලයුත්තේ වැඩසටහනක් debug කර බැලීමයි.

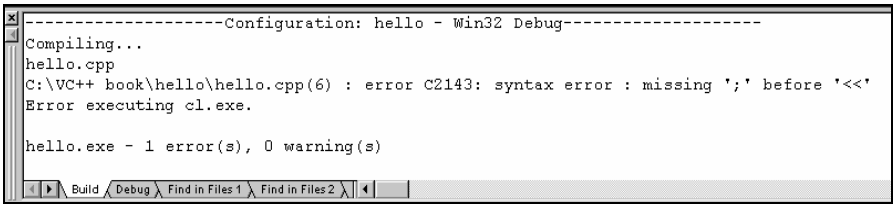
මේ සඳහා 10 පරිච්ඡේදයේදී අප සලකා බැලූ පරිගණක වැඩසටහන ඔබට යොදා ගත හැක. මේ සඳහා පලමුව, ඔබ සෑදූ Media Player project එක Visual C++ තුල විවෘත කර ගන්න. දැන්, debug window එකෙහි ඇති step over button එක හෝ එහි keyboard shortcut එක වන F10 යතුර ඔබන්න. එවිට වැඩසටහනෙහි ක්‍රියාකාරී ලක්ෂය පෙන්වන ඊතලය, වැඩසටහනෙහි main function එක තුල මතුවී පෙනෙනු ඇත. දැන්, ක්‍රමලේඛණයෙහි `pl.MediaFunc(1);` යන කේතය ඉදිරියට ඊතලය පැමිණෙන තෙක් F10 ඔබන්න. දැන්, `MediaFunc` function එකට ඇතුලු වීම පිණිස, step in button එක හෝ F11 යතුර හෝ ඔබන්න. එවිට ඊතලය main තුලින් ඉවත්ව, MediaFunc function එක තුලට ගමන් කරනු ඔබට දැකගත හැකි වනු ඇත. මීලඟට MediaFunc හි අවසානය වන තෙක්ම, F10 මගින් ඊතලය ගමන් කරවන්න. Function එක අවසානය තෙක් ගමන් කිරීමෙන් අනතුරුව ඊතලය නැවතත් main තුලට ඇතුල් වනු ඇත. මේ ආකාරයෙන්, ඔබට වැඩසටහනක් ක්‍රියාත්මක වන ආකාරය එය debug කිරීම මගින් දැකගත හැකිය.

පරිගණක වැඩසටහන් වල දෝශ - Programming errors

පහත දැක්වෙන්නේ මෙම පොතෙහි මුල් භාගයේ ඇති **"ඔබේ පලමු C++ පරිගණක වැඩසටහන"** නැමැති කොටසේ අඩංගු වන වැඩසටහනම දෝශ සහිතව ලියා ඇති ආකාරයයි:

```
================================================

#include<iostream.h>

void main()
{
      << "Hello beautiful world!" << endl;
}

================================================
```

ඉහත වැඩසටහන දෙස බැලුවහොත් එහි cout command එක නොමැති බව ඔබ දකිනු ඇති. මෙය error එකකි. මෙම වැඩසටහන VC++ තුල type කර compile කිරීමට උත්සාහ ගතහොත්, output window එක මගින් එම දෝශය පෙන්වා දෙනු ලබන බව ඔබ දකීවී. ඔබට output window එකතුල පහත දැක්වෙන ආකාරයේ output එකක් ලැබෙනු ඇත.



(රූපය 12.3)

Output window එකෙහි තෙවන පෙලින් (hello.cpp යන්නට පසුව ඇති පේළියෙන්) මෙම error එක දක්වනු ලැබේ. එම තෙවන පේළිය මත double click කර සිදුවන්නේ කුමක්දැයි බලන්න. එවිට VC++ හි editor window එකතුල එම error එක නිර්මාණය කල කේතය highlight වන බව ඔබ දකිනු ඇත. මේ අයුරින්, වැඩසටහනක කිසියම් දෝශයක් ඇතිනම් එය හඳුනාගෙන එය නිවැරදි කල හැකිය. ඔබ කලයුත්තේ,

(1) **Output window එකෙහි ඔබට අවශ්‍ය error එක දක්වන පෙලෙහි සදහන් වන්නේ කුමක්දැයි පරීක්ෂා කර බලන්න**.

දෝශ සහිත පරිගණක කේතයක් නිවැරදි කිරීමට නම් ඔබ පලමුව output window එකෙහි error එක මගින් කියවෙන්නේ කුමක්දැයි වටහාගත යුතුය. මක්නිසාදයත් ඔබ ගැටලුවකට පිලතුරක් සෙවීමට පලමුව ගැටලුව හරිහැටි වටහාගත යුතු හෙයිනි. ගැටලුව කුමක්දැයි හඳුනාගත් පසු ඔබට දෝශ සහිත කේතය නිවැරදි කල හැක.

Output window එකෙහි error එක සදහා හේතුවුණු කරුණ සදහන්වී තිබෙනු ඇත. ඇතැම්විට එම විස්තරයෙහි error එක කුමක්දැයි සෘජුවම සදහන් නොවේ. ඇතැම්විට සඳහන් වනු ඇත්තේ එම error එක නිසා මතුවුණු වෙනත් වරදක් පිලිබඳවය. උදාහරණයක් ලෙස ඉහත උදාහරණයේ output window එකෙහි ඇති error එක සදහන් වගන්තියෙන් කියවෙන්නේ වැඩසටහනෙහි “<<“ ලකුණට මුලින් “**;**” සලකුණ නොමැති බවය. සැබෑ වරද වන්නේ cout command එක නොමැති වීම බව එහි සඳහන් නොවේ. නමුත් එහි සඳහන් දෙය ඇතිවූයේ එහි cout විධානය නොමැති වීම නිසාය. එහිදී සෘජුවම error එක කුමක්දැයි සදහන් නොවුණි. සමාන්‍ය ලෙසම ඔබට තවම output window එක දෙස බලා කිසියම් error එකක් නිවැරදිව හඳුනාගැනීමට නොහැකිවනු ඇති. මෙම හැකියාව ඔබ වෙත පැමිණෙන්නේ පරිගණක වැඩසටහන් ලිවීමෙන් හා compile කිරීමෙන් ඔබ ලබන අද්දැකීම් මතය. මේනිසා දැනට compiler errors වටහා ගැනීම පිලිබඳව වැඩිපුර කරදර නොවන්න. හැකිතරම් පරිගණක වැඩසටහන් ලියමින් අද්දැකීම් එකතු කරගැනීමට උත්සාහ කරන්න. එවිට මෙම හැකියාවන් ඉබේම ඔබ කරා පැමිණෙනු ඇත.

(2) **Error එකමත double click කරමින් error එක නිපදවූ කේතය කුමක්දැයි හඳුනාගන්න**.

දෝශය කුමක් දැයි දැනගත් පසු ඔබට එම දෝශ සහිත ස්ථානය වෙත ගමන් කල හැකිය. එසේ කර cout විධානය නිසිතැන type කර වැඩසටහන නිවැරදි කලහැකිය.