

DATA STRUCTURE AND ALGORITHMS

FOR BEGINNERS



Dr. Myat Mon Aye

Ph.D. (IT)

myatmonaye@studyrighnow-mdy.com

Table of Contents

အခန်း (၀) – Computer System	1
အခန်း (၁) – Numbering System	7
အခန်း (၂) – သချို့ကြောင်းများ	23
အခန်း (၃) – Boolean algebra	36
အခန်း (၄) – Statistics ဆိုင်ရာ အခြေခံ	46
အခန်း (၅) – Data Structure မိတ်ဆက်	61
အခန်း (၆) – Algorithm မိတ်ဆက်	71
အခန်း (၇) – Control Flow and Flowchart	79
အခန်း (၈) – Efficiency and Basic Algorithms	107
အခန်း (၉) – Recursion	129
အခန်း (၁၀) – Data သိမ်းဆည်းပုံ	137
အခန်း (၁၁) – String	154
အခန်း (၁၂) – Arrays and Basic Algorithms	170
အခန်း (၁၃) – Stack, Queue and Hashing	201
အခန်း (၁၄) – Sorting	230
အခန်း (၁၅) – Linked List	269
အခန်း (၁၆) – Tree	295
အခန်း (၁၇) – Graph	324

ပညာတွေကို သင်ကြားဖို့ အခွင့်အရေးပေးခဲ့တဲ့ မိသားစု၊ ငယ်စဉ်မှစပြီး ယခုအချိန်ထိ
ပညာများသင်ကြားပေးခဲ့တဲ့ သင်ဆရာ၊ မြင်ဆရာ၊ ကြားဆရာများ၊ စာသင်ကြားခြင်းကို လုပ်ဖြစ်အောင်
အခွင့်အရေးပေးခဲ့ကြသူများ၊ ဤစာအုပ်နှင့်ပတ်သက်ပြီး ကျေးဇူးတင်ထိုက်သူအားလုံးကို ကျေးဇူးတင်
ကန်တော့ပါတယ်။

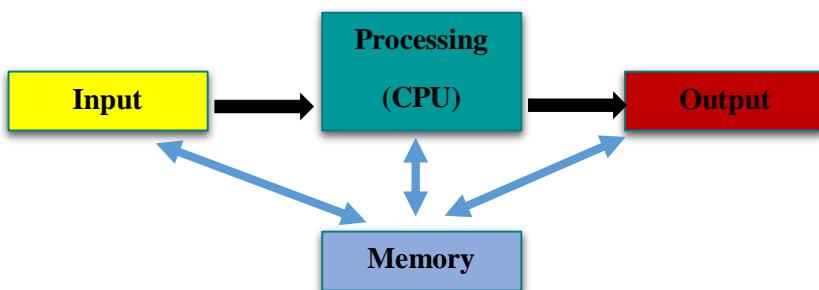
နှစ်ပေါင်း ဆယ်စုနှစ်ကျော်တိုင်အောင် စာသင်ကြားဖို့ အခွင့်အရေး ပေးခဲ့ကြသော၊ IT Field မှာ ကျင်လည်ဖို့
အခွင့်အရေးပေးခဲ့ကြသော တပည့်အပေါင်းကိုလည်း အထူးကျေးဇူးတင်ပါတယ်။

အခန်း (၀)

Computer System

Computer System ဆိတာဘာလဲ။

Computer System ဆိတာ ဘာလဲ မပြောခင် computer တစ်လုံး၏ အလုပ်လုပ်ပုံကို အရင်ပြောပါမယ်။



- Input ဆိတ္တဲ့ က ဝင်လာတာပါ
- Processing ဆိတာက အလုပ်လုပ်တာပါ၊ တွက်ချက်တာပါ
- Memory ဆိတာက အချက်အလက်တွေ သိမ့်းဆည်းထားတဲ့ နေရာပါ
- Output ဆိတာကတော့ ထုတ်ပြတာပါ

ဈေးဝယ်သွားရင်းနဲ့ လူတစ်ယောက်က ကိုယ့်ကို ရယ်ပြနေတာ မြင်လိုက်တယ်ဆိုပါစွဲ၊ အဲဒီလူကို ရင်းရင်းနှီးနှီး သိနေတယ်၊ ဒါပေမဲ့ ဘယ်သူလဲဆိတာ စဉ်းစားလို့မရဘူး၊ တစ်ဖက်က ရယ်ပြနေတော့ ခေါင်းတွေထူးနေအောင် စဉ်းစားတယ်၊ ဟောနောက်ဆုံးတော့ ဘယ်သူလဲ ဆိတာ စဉ်းစားလို့ရပြီး ဝမ်းသာအားရနဲ့ ရယ်ပြနေသူရဲ့ အမည်ကို ပြုးပြုးကြီး ရော်တိုက်တယ်။

- “လူတစ်ယောက်က ကိုယ့်ကို ရယ်ပြနေတာ မြင်လိုက်တယ်” ဆိုတာ ကိုယ့်ရဲ့ မျက်လုံးထဲကနေ ဝင်လာတာ ဖြစ်တဲ့အတွက် input ပါ။ မျက်လုံးဆိုတာကတော့ computer လို ပြောမယ် ဆိုရင် input device ပါ။
- ကျွန်ုမတို့ဟာ ကျွန်ုမတို့ မြင်သမျှ၊ ကြေားသမျှ၊ သိသမျှ အရာတွေ အားလုံးကို မှတ်ဉာဏ်ထဲမှာ သိမ်းဆည်းလေ့ရှိပါတယ်။ အဲဒီ မှတ်ဉာဏ်ကို memory လို့ ခေါ်ပါတယ်။
- “ခေါင်းတွေထူးဖော်အောင် ဘယ်သူလဲ စဉ်းစားတယ်” ဆိုတာ processing ပါ။ ဘယ်လို စဉ်းစားသလဲ ဆိုတော့ မျက်လုံးကနေ ဝင်လာတဲ့ ရယ်ပြနေတဲ့ လူရဲ့ ပုံကို မှတ်ဉာဏ်ထဲ သိမ်းထားတဲ့ data တွေနဲ့ တိုက်စစ်ပြီး လိုက်ရှာတာပါ။ ဒါကြောင့် တစ်ချို့ မှတ်ဉာဏ်ပျောက်တဲ့သူတွေဆိုရင် ဘာမှ မမှတ်မိဘူးဆိုတာ သိမ်းထားတာတွေ ယိုယွင်းကုန်လို့ ရှာမတွေ့တော့တဲ့သဘောပါ။ လူတွေမှာ စဉ်းစားတာ တွက်ချက်တာ တွေကို ဦးကျောက်က အလုပ် လုပ်သလို computer မှာတော့ CPU(Central Processing Unit) က အဲဒီအလုပ်တွေကို လုပ်ဆောင်ပါတယ်။ ဒါကြောင့် CPU ကို brain of computer လို့ ခေါ်ပါတယ်။ CPU ဟာ computer ရဲ့ အရေးကြီးဆုံး အစိတ်အပိုင်း ဖြစ်တဲ့အတွက် computer တစ်လုံးကို ပြောတဲ့အခါမှာ CPU နဲ့ i3, i5, i7 ဆိုပြီးပြောလေ့ ရှိပါတယ်။
- တိုက်စစ်လို့ တွေ့သွားပြီဆိုတော့ ပြီးပြီးကြီး အမည်ကို ရေးတိုက်တယ်ဆိုတာ output ထုတ်တာပါ။ output ကို နှစ်နေရာကနေထုတ်သွားတာပါ။ ပြီးပြီးကြီး ဆိုတာ output ကို မျက်နှာအမှုအရာကနေ ထုတ်ပြတာဖြစ်ပြီး၊ အမည်ကို ခေါ်လိုက်တယ် ဆိုတာ output ကို ပါးစပ်ကနေ ထုတ်လိုက်တာဖြစ်ပါတယ်။ Computer လိုပြောမယ် ဆိုရင်တော့ မျက်နှာရယ်၊ ပါးစပ်ရယ်ဆိုတဲ့ output device J ခုကို အသုံးပြုပြီး output ထုတ်သွားတာပါ။

ဒီလိုပါပဲ၊ computer ဆိုတာ လူနဲ့ ဆင်တူပါတယ်။ သူရဲ့ input devices တွေကတော့ mouse, keyboard, barcode reader (စတိုးဆိုင်တွေမှာ တွေ့ဖူးမှာပေါ့၊ ကောင်တာမှာ ပစ္စည်းတွေကို တီ ဆိုပြီး

အသံလေးမည်မည်ပြီး ဖတ်တဲ့စက်လေး)၊ scanner(တစ်ချို့ရုံးတွေမှာ လက်ဖွေတို့ မျက်နှာတို့နဲ့ရုံးရောက်ကြောင်း သတင်းပိုတဲ့ စက်လေးတွေ၊ တနည်းအားဖြင့် မျက်နှာ၊ လက်ဖွေ၊ ဓါတ်ပုံ စသည်ဖြင့် တစ်ခုခု ဖတ်တဲ့ စက်ကလေးတွေ) ဖြစ်ပါတယ်။ တခြား အများကြီး ရှိပါသေးတယ်။

Computer ရဲ့ output devices တွေကို ပြောရမယ် ဆိုရင်တော့ computer screen (monitor), စာရွက်ထုတ်တဲ့စက် (printer) တို့ စသည်ဖြင့် အများကြီး ရှိပါတယ်။

Memory မှာဆိုရင်တော့ အမိက အားဖြင့် ၃ မျိုးရှိပါတယ်။

1. **ROM (read only memory)** – read only ဆိုတဲ့ အတိုင်း ဖတ်လိုပဲ ရပါတယ်၊ သွားပြင်တာ ရေးတာ လုပ်လိုမရပါဘူး၊ အမိကအနေနဲ့ကတော့ စက်ကို ထုတ်လုပ်လိုက်တဲ့ ထုတ်လုပ်သူ(manufacturer) က စက်အလုပ်လုပ်ဖို့အတွက် လိုအပ်တဲ့အရာတွေကို ROM ထဲ ထည့်ပေးလိုက်တာပါ။ ROM ထဲ ထည့်ပေးလိုက်တာ ဖြစ်လို့ ကျွန်မတို့ ယူသုံးလိုပဲ ရပါတော့တယ်၊ သွားပြင်လို့ မရပါဘူး။
2. **Secondary Memory** - သူက power မရှိလဲ (စက်ပိတ်ထားလို့ လျှပ်စစ်မရှိလဲ) data(စာတွေ၊ သီချင်းတွေ၊ ပုံတွေ စသည်ဖြင့်) တွေကို ထိန်းသိမ်းထားနိုင်ပါတယ်၊ power ပိတ်လိုက်လို့ ပျောက်မသွားပါဘူး၊ ကျွန်မတို့ ဖျက်မှ ပျောက်ပါတယ်။ ဒါကြောင့် ကျွန်မတို့ စက်ထဲ သိမ်းသမျှဟာ အားလုံးဟာ secondary memory ပေါ်ရှိနေတာပါ။
3. **Main memory or primary memory or RAM** - ယခုလက်ရှိဖွင့်တာ အလုပ်လုပ်တာ မှန်သမျှကို main memory ပေါ်ဆဲတင်ပြီး အလုပ်လုပ်ပါတယ်။ သူကတော့ power ပိတ်လိုက်တာနဲ့ data အားလုံးပျောက်ကုန်ပါတယ်။ ဒါကြောင့် အငွေ့ပုံသလို ပျောက်တယ် ဆိုတဲ့ အဓိပါယ်နဲ့ **volatile memory**လို့ ခေါ်ပါတယ်။ ဆန္ဒကျင်ဘက် secondary memory ကိုတော့ **non-volatile memory**လို့ ခေါ်ပါတယ်။

စက်ဝယ်တုန်းက ဆိုင်ကန္မာပြီးတော့ စာရိုက်တဲ့ software (e.g. Microsoft word) ကို စက်ထဲ
ထည့်ပေးလိုက်တယ် ဆိုပါတော့။ အဲဒါဆိုရင် အဲဒီ software ဟာ စက်ထဲ အမြှေ့ရှိနေမှာပါ၊ ဘယ်အချိန်
ဖွင့်သုံးသုံးရပါတယ်။ ဒါဆို Microsoft Word ဆိုတဲ့ software ဟာ စက်ပိတ်တောင်
ပျောက်မသွားဘူးဆိုတော့ Secondary Memory ပေါ်မှာ ရှိနေတာပါ။

စာရိုက်ချင်လို့ J ချက်နိုပ်ပြီး Microsoft Word ကိုဖွင့်လိုက်ပြီ ဆိုရင်တော့ ယခုလက်ရှိ
အလုပ်လုပ်တာဖြစ်လို့ Main Memory ပေါ်ရောက်လာတာပါ။ စာရိုက်တယ် စာရိုက်တယ် Main
Memory ပေါ်မှာ အလုပ်လုပ်နေတာပါ။ အဲဒီအချိန် မီးယျက်သွားပြီး စက်ပိတ်သွားကြည့်ပါလား
ရိုက်လက်စစာတွေ ပျောက်သွားမှာဖြစ်ပါတယ်။ ဘာလို့လဲ ဆိုတော့ Main Memory ဆိုတာ power
မရှိရင် (စက်ပိတ်လိုက်တာနဲ့) data တွေ ပျောက်သွားလိုပါ။

စက်မပိတ်ခင် save လုပ်ခဲ့မယ် ဆိုရင် နောက်နဲ့ ပြန်ဖွင့်ရှိသလားဆိုတော့ ရှိပါတယ်။ ဒါဆို
save တာက ဘယ်မှာ သွားသိမ်းသလဲဆိုရင် Secondary Memory မှာ သွားသိမ်းလို့ဖြစ်ပါတယ်။

ဒီလောက်ဆိုရင် Computer System ကို သဘောပေါက်လောက်ပြီ ထင်ပါတယ်။ Computer
System ဆိုတာ input devices (input components), output devices (output components),
processing devices (processing components) နဲ့ memory စတဲ့ အစိတ်အပိုင်းတွေ
စုထားတာဖြစ်တယ်လို့ အကြမ်းဖျင်း ပြောလို့ရပါတယ်။

အရင်ဆုံးတော့ input device ကနေ input ဝင်လာမယ်၊ processing က တွက်ချက်မယ်၊
output က အဖြေဖြန်ထုတ်ပြမယ်။ အဲဒီမှာ input ၊ processing နဲ့ output devices တွေဟာ data
တွေ သိမ်းဆည်းထားတဲ့ memory နဲ့ ချိတ်ဆက်ပြီး အလုပ်လုပ်ပါတယ်။

Memory တွေဟာ မတူညီကြပါဘူး

လူတစ်ယောက်နဲ့ တစ်ယောက် မှတ်ဉာဏ်ကောင်းတာ၊ မကောင်းတာတွေ၊ မှတ်နိုင်၊ သိမ်းဆည်းနိုင်တဲ့ ပမာဏတွေ မတူညီကြပါဘူး။ အဲဒီလိုပါပဲ computer တစ်လုံးနှင့် တစ်လုံးမှာလဲ memory တွေဟာ အလုပ်လုပ်နိုင်စွမ်း (speed) နဲ့ သိမ်းဆည်းနိုင်တဲ့ ပမာဏတွေ (size) တွေ မတူညီကြပါဘူး။

Memory Measurement Units

အရပ်ဘယ်လောက်ရှည်သလဲ ဆိုရင် လက်မ(inches)၊ ပေ(feet)၊ မီတာ (meter) တို့နဲ့ တိုင်းတယ်၊ ခန္ဓာကိုယ် အလေးချိန်ကို kg, lb တို့နဲ့ တိုင်းပါတယ်။ အဲဒီလိုပါပဲ memory ပေါ်မှာ ဘယ်လောက်နေရာယူသလဲဆိုတာကို bit, byte, KB, MB, GB, TB တို့နဲ့ တိုင်းပါတယ်။

တစ်ပေမှာ ၁၂ လက်မ ဆိုတဲ့ ပေနဲ့ လက်မကြား ဆက်သွယ်ချက် ရှိသလိုပဲ bit, byte, KB, MB, GB, TB ကြားမှာလည်း ဆက်သွယ်ချက်တွေ ရှိပါတယ်။

- Bit – memory ကို တိုင်းတာတဲ့ အသေးဆုံး unit ပါ။ 0 သို့မဟုတ် 1 တစ်လုံးပဲ သိမ်းလို့ ရပါတယ်။
- Byte – 1-byte မှာ 8 bits ရှိပါတယ်။
- KB (kilobyte) – 1 KB မှာ 1024 Byte ရှိပါတယ်။
- MB (megabytes) – 1 MB မှာ 1024 KB ရှိပါတယ်။
- GB (gigabytes) – 1 GB မှာ 1024 MB ရှိပါတယ်။
- TB (terabytes) – 1 TB မှာ 1024 GB ရှိပါတယ်။

အလွယ်အနေနဲ့တော့ 1024 အစား 1000 လို့ပြောလေ့ရှိပါတယ်

“တွက်စာတွေ စပြီး လာတော့မှာ ဖြစ်လို့
မှတ်ဖို့ တွက်ဖို့ စာအုပ်တစ်အုပ် လိုလာပါပြီ။

အများအားဖြင့် ကျွန်ုင်မကတော့
မျဉ်းပါတဲ့ စာအုပ်အချောတွေထက်၊
အကြမ်းစာရွက်တွေ စုချပ်ထားတဲ့
စာအုပ်ကို ပိုပြီးသဘောကျပါတယ်။

စာအုပ်အကြမ်းလေးတွေက
ရေးလိုက်ရင်လည်း အိန္ဒတာပဲ၊
ပြီးတော့ အဲဒီတဲ့မှာ
မှတ်စုလေးတွေသေချာရေး၊ ပုဇွဲလေးတွေ သေချာတွက်လိုက်လို့
အကြမ်းစာအုပ်လေးကနေ အဖိုးတန်လေးဖြစ်သွားတဲ့
ခံစားချက်ကို သိပ်သဘောကျတာ။”

အခန်း (၁)

Numbering Systems

Numbering Systems ဆိတာ ဘာလဲ။

No	Numbering System	Base	Numbers
1	Binary	2	0,1
2	Octal	8	0,1,2,3,4,5,6,7
3	Decimal	10	0,1,2,3,4,5,6,7,8,9
4	Hexadecimal	16	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Numbering Systems 4 မျိုးမှာ ကျွန်မတိ နေ့စဉ် အသုံးပြုနေတာကတော့ **Decimal** စနစ်ပဲဖြစ်ပါတယ်။ သူဟာ base 10 ပါ။ ဒါကြောင့် ဂဏန်း 10 လုံးကိုပဲ ခွင့်ပြုပါတယ်။ 10 လုံးကို 0 ကနေ စတင်ရေတွက်လိုက်တဲ့ အခါမှာ 9 မှာ 10 လုံးပြည့်သွားပါတယ်။ 129 (တစ်ရာနှစ်ဆယ့်ကိုး) လို့ခေါ်တဲ့ ကိန်းဂဏန်းတစ်လုံးမှာ digit 3 လုံးပါနေပါတယ်။ 1 ရယ် 2 ရယ် 9 ရယ်ပါ။ ရာဂဏန်း ဖြစ်တဲ့အတွက် digit 3 လုံးပါတာပါ။ ထောင်ဂဏန်းဆိုရင် digit 4 လုံးပါပါမယ်။ digit ဘယ်နှစ်လုံးပဲပါပါ decimal ဖြစ်တဲ့အတွက် digits တွေဟာ 0 to 9 အတွင်းမှာပဲ ဖြစ်မှာ ဖြစ်ပါတယ်။

Binary ကတော့ base 2 ဖြစ်တဲ့အတွက် 2 လုံးခွင့်ပြုပါတယ်။ 0 ကနေ စတင်ရေတွက်လိုက်တော့ 1 မှာ 2 လုံးပြည့်ပါတယ်။ ဒါကြောင့် 0,1 ကိုသာ binary မှာ ခွင့်ပြုပါတယ်။

Octal ကတေသာ base 8 ဖြစ်တဲ့အတွက် 8 လုံးခွင့်ပြုပါတယ်။ 0 ကနေ စတင်ရေတွက်လိုက်တော့ 7 မှာ 8 လုံးပြည့်ပါတယ်။ ဒါကြောင့် 0,1,2,3,4,5,6,7 ကိုသာ octal မှာ ခွင့်ပြုပါတယ်။

Hexadecimal ကတေသာ base 16 ဖြစ်တဲ့အတွက် 16 လုံးခွင့်ပြုပါတယ်။ 0 ကနေ စတင်ရေတွက်လိုက်တော့ 15 မှာ 16 လုံးပြည့်ပါတယ်။ အဲဒီမှာ 0 to 9 ကိုတော့ 0 to 9 လို့ ရေးပြီး 10 ကိုတော့ A လို့ရေးပြီး၊ 11 က B, 12 က C, 13 က D, 14 က E, 15 က F လို့ ရေးပါတယ်။ ဒါကြောင့် 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F ကိုသာ hexadecimal မှာ ခွင့်ပြုပါတယ်။

Converting Between Numbering Systems

လက်မကို ပေ့ဖို့၊ ပေ ကို လက်မဖို့၊ နဲ့ လိုအပ်သလို ပြောင်းပြီး တွက်ကြရသလို Numbering System တွေကိုလည်း လိုအပ်ရင် တစ်ခုကနေ တစ်ခုကို ပြောင်းတွက်လေ့ရှိပါတယ်။ အလွယ်မှတ်မယ်ဆိုရင် နှစ်ချက်ပဲ မှတ်ဖို့လိုပါတယ်။

- Decimal ကိုလာရင် ပြောက်ပါမယ်။ binary ကနေလာရင် binary ရဲ့ base ဖြစ်တဲ့ 2 power တွေနဲ့ ပြောက်၊ Octal ကနေလာရင် 8 power တွေနဲ့ ပြောက်၊ Hexadecimal ကနေ လာရင် 16 power တွေနဲ့ ပြောက်ရုံပါပဲ။
- Decimal ကသွား စားပါမယ်။ ဘာနဲ့စားမလဲ ဆိုရင် binary ကိုသွားရင် binary ရဲ့ base ဖြစ်တဲ့ 2 နဲ့စားပါမယ်။ ဒါဆိုရင် Octal ကိုသွားရင် 8 နဲ့စား၊ Hexadecimal ကိုသွားရင် 16 နဲ့စား လိုက်ရုံပါပဲ။



Converting from Others to Decimal

Binary to Decimal



$$(1101.101)_2 = (?)_{10}$$

$$(1101.101)_2 = 1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 + 1 * 2^{-1} + 0 * 2^{-2} + 1 * 2^{-3}$$

$$= 1 * 8 + 1 * 4 + 0 * 2 + 1 * 1 \text{ (မည်သည်ကိန်းမဆို ထပ်ကိန်း 0 တင်ရင် 1 ပါ) } +$$

$$1/2 + 0/4 + 1/8$$

$$= 8 + 4 + 0 + 1 + 0.5 + 0 + 0.125$$

$$= (13.625)_{10}$$

ပုံစွဲက base 2, binary ကနေ base 10, decimal ကို ပြောင်းခိုင်းတာဖြစ်လို့ 2 power တွေနဲ့ မြှောက်ပါမယ်။ 2 power တွေနဲ့ မြှောက်တဲ့ အခါ ဒဿမ ရှေ့ ကပ်ရက်က ထပ်ကိန်း 0 နေ့စပ္ပါမယ်။ ရှေ့ကို သွားလေ ထပ်ကိန်း ၁ တိုးလေလေဖြစ်ပြီး ဒဿမကနေ နောက်ကိုသွားလေ ထပ်ကိန်း ၁ လျှော့လေလေ တွက်ရမှာ ဖြစ်ပါတယ်။

Octal to Decimal



$$(120.1)_8 = (?)_{10}$$

$$= 1 * 8^2 + 2 * 8^1 + 0 * 8^0 + 1 * 8^{-1}$$

$$= 1 * 64 + 2 * 8 + 0 * 1 + 1/8$$

$$= 64 + 16 + 0 + 0.125 = (80.125)_{10}$$

Hexadecimal to Decimal

$$(1A9)_{16} = (?)_{10}$$

$$= 1 * 16^2 + A * 16^1 + 9 * 16^0$$

$$= 1 * 256 + 10 * 16 + 9 * 1$$

$$= 256 + 160 + 9 = (425)_{10}$$

**Converting from Decimal to Others**

$$(123)_{10} = (?)_2$$

2 123

2 61(123 ကို 2 ဖြင့် စားလိုက်သော စားလဒ်), 1(123 ကို 2 ဖြင့် စားလိုက်သော အကြွင်း)

2 30(61 ကို 2 ဖြင့် စားလိုက်သော စားလဒ်), 1(61 ကို 2 ဖြင့် စားလိုက်သော အကြွင်း)

2 15, 0

2 7, 1

2 3, 1

2 1, 1

2 0, 1 (စားလဒ် 0 ဖြစ်သွားလျှင်ရပါ)



ပြီးလျှင် အကြွင်းတွေကို အောက်မှ အပေါ်သို့ ချရေးပြီး အဖြစ်ထုတ်ရမှာ ဖြစ်တဲ့အတွက်

$$(123)_{10} = (1111\ 011)_2$$

က အဖြေမှန်မမှန် အပေါ်မှာ ပြောခဲ့သလို ပြန်တွက်ကြည့်ရအောင်။ အဖြေ binary ကနေ decimal ပြန်ပြောင်းကြည့်မယ်။

$$\begin{aligned}
 (1111\ 011)_2 &= 1 * 2^6 + 1 * 2^5 + 1 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0 \\
 &= 1 * 64 + 1 * 32 + 1 * 16 + 1 * 8 + 0 * 4 + 1 * 2 + 1 * 1 \\
 &= 64 + 32 + 16 + 8 + 2 + 1 \\
 &= (123)_{10}
 \end{aligned}$$

က မူရင်း အဖြေ ပြန်ရတာ တွေ့ရမှာပါ။ အလွယ်နည်းနဲ့ ပြောပြမယ်။

ပေးထားတာ ချေရေး	\times	1	1	1	1	0	1	1
ဒေသမရှိကပ်ရက်က 2, 0 ထပ်ဆိုတော့ 1, ရှိသွား ၁ထပ်တိုးတော့ 2, 1 ထပ် 2, ပြီးရင် 2 ထပ် 4,.....		64	32	16	8	4	2	1
		2^6	2^5	2^4	2^3	2^2	2^1	2^0

$$= 1 * 64 + 1 * 32 + 1 * 16 + 1 * 8 + 0 * 4 + 1 * 2 + 1 * 1 \text{ (၁ ဖြစ်တဲ့ နေရာတွေ ယူပေါင်းလိုက်လည်းရပါတယ်)}$$

$$= 64 + 32 + 16 + 8 + 2 + 1 = (123)_{10}$$

Decimal to Octal



$$(123)_{10} = (?)_8$$



$$(123)_{10} = (173)_8$$

ကဲ အဖော်မှန်မှန် အပေါ်မှာ ပြောခဲ့သလို ပြန်တွက်ကြည့်ရအောင်။ အဖော် octal ကနေ decimal ပြန်ပြောင်းကြည့်မယ်။

$$(173)_8 = 1 * 8^2 + 7 * 8^1 + 3 * 8^0$$

$$= 1 * 64 + 7 * 8 + 3$$

$$= 64 + 56 + 3 = (123)_{10}$$

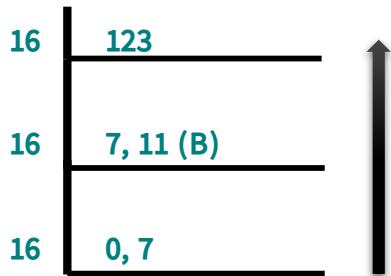
မူရင်းအဖော်ပြန်ရတာ တွေ့ရမှာပါ။ အလွယ်နည်း နဲ့ တွက်ကြည့်ရအောင်။

1	7	3
$64(8^2)$	$8(8^1)$	$1(8^0)$
$1 * 64 = 64$	$7 * 8 = 56$	$3 * 1 = 3$

$$= 64 + 56 + 3 = (123)_{10}$$

Decimal to Hexadecimal

$$(123)_{10} = (?)_{16}$$



$$(123)_{10} = (7B)_{16}$$

အလွယ်နည်းနဲ့ ပြန်ပြောင်းပြီး အဖြေ စစ်ကြည့်ကြရအောင်။

7	B
$16 (16^1)$	$1 (16^0)$
$7 * 16 = 112$	$11 * 1 = 11$

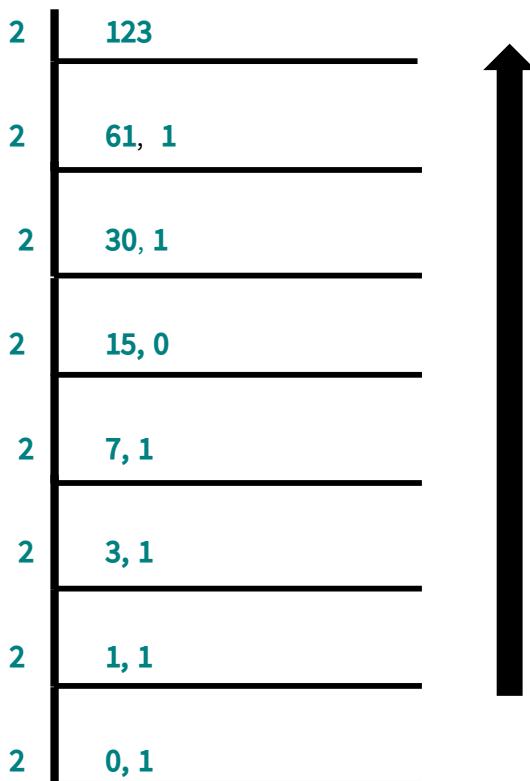
$$112 + 11 = (123)_{10}$$

**Decimal ဒသေမကိန်း ကို ပြောင်းနည်း**

Decimal ဒသေမကိန်း ကို ပြောင်းမယ် ဆိုရင်တော့ အပိုင်းနှစ်ပိုင်းခဲ့ပြီး ပြောင်းရမှာ ဖြစ်ပါတယ်။ ဒသေမ ရှုံးအပိုင်းကိုတော့ ထဲးစံအတိုင်း စားပြီး၊ ဒသေမ အနောက်ပိုင်းကိုတော့ ထပ်ခါထပ်ခါ ငြောက်ပေးရပါတယ်။

❓ $(123.125)_{10} = (?)_2$

ဒသုမ ရှေ့ အပိုင်းကိုတော့ အရင်အတိုင်း စားလဒ် 0 ရသည့်အထိ ထပ်ခါ ထပ်ခါ စားပါမယ်။



ဒီတော့ ဒသုမ ရှေ့ပိုင်း အဖြောက် 1111 011 ပါ။

ဒသုမ နောက်ပိုင်းကိုတော့ ထပ်ခါ ထပ်ခါ မြောက်ပါမယ်။ Binary ကိုသွားမှာ ဖြစ်လို 2နဲ့ မြောက်ပါမယ်။ Octal ဆုံး 8, hexadecimal ဆုံး 16 ပေါ့။

$$\begin{aligned}
 0.125 * 2 &= 0.250 \quad (\text{ဒသုမရှေ့ ဂဏန်း } = > 0) \\
 .25 * 2 &= 0.50 \quad (\text{ဒသုမရှေ့ ဂဏန်း } = > 0) \\
 .50 * 2 &= 1.0 \quad (\text{ဒသုမရှေ့ ဂဏန်း } = > 1)
 \end{aligned}$$

တစ်ကြိမ်ချင်းရတဲ့ ဒသုမရှေ့
ဂဏန်းတွေကို အပေါ်အောက်ယူပြီး
အဖြေထုတ်ပါ

မြောက်လို ရလာတဲ့ အဖြေရဲ့ ဒသုမနောက် 0 ပဲ မထွက်သေးလို ထပ်မြောက်မယ်ဆုံးရင် ပြန်မြောက်ဖို့ ဒသုမနောက်က ဂဏန်းကိုပဲ ယူတာ တွေ့ပါလိမ့်မယ်။ အခုတွက်တာမှာက ဒသုမရှေ့ 0 ချဉ်းတွက်လို သိပ်မသိသာပါဘူး။ ဥပမာ 1.125 ဆုံးရင် အောက်မှာ ပြန်မြောက်ဖို့ 0.125 ပဲယူရပါမယ်။

မြောက်လိုဂုဏ်တဲ့အဖြစ် ဒသဗုမ နောက်မှာ 0 ပဲရှိတော့ရင် ရပ်ပါမယ်။ တစ်ခုံးဟာတွေမှာ ဘယ်လောက်မြောက်မြောက် ဒသဗုမနောက် 0 ဖြစ်မသွားတာတွေ ရှိပါတယ်၊ အဲဒီအခြေအနေဆိုရင် ထိုက်သင့်သလောက် အကြိမ် အရေအတွက်(အနည်းဆုံး 8 ကြိမ်လောက်) မြောက်ပြီးရင် ရပ်လိုက်ပါ။ တစ်ကြိမ်ခြင်းရတဲ့ ဒသဗုမရှိ ဂဏုန်းတွေကို အပေါ်အောက်ယူလိုက်ရင် **001** ဆိုပြီး ရပါတယ်။

အဖြေ နှစ်ခုပြန်ပေါင်းလိုက်မယ်ဆိုရင် **1111 011.001** ဆိုပြီးရပါတယ်။

$$(123.125)_{10} = (1111\ 011.001)_2$$



ဒီလိုဆိုရင် ဒသဗုမကိန်းတွေကိုလည်း တစ်ခြား Numbering System တွေကို ပြောင်းတာ တွက်တတ်လောက်ပါပြီ။ အောက်က ပုဇွာလေးတွေ တွက်ကြည့်ကြည့်ပါ။

$$(2345.140625)_{10} = (?)_2$$

$$(2345.140625)_{10} = (?)_8$$

$$(2345.140625)_{10} = (?)_{16}$$

$$(246.8)_{10} = (?)_2$$

$$(246.8)_{10} = (?)_8$$

$$(246.8)_{10} = (?)_{16}$$



Binary, Octal and Hexadecimal

- Binary က base 2 ဖြစ်ပါတယ်။
- Octal က base 8၊ 8 ဆုံးတာ 2, 3 ထပ်ဖြစ်တဲ့အတွက် Octal တစ်လုံးမှာ Binary 3 လုံးရှိပါမယ်။

- Hexadecimal က base 16၊ 16 ဆိုတာ 2, 4 ထပ်ဖြစ်တဲ့အတွက် Hexadecimal တစ်လုံးမှာ Binary 4 လုံးရှိပါမယ်။

Octal and Binary

Octal တစ်လုံးမှာ binary 3 လုံးရှိတာဖြစ်လို့ အောက်က ပေါ်လေးကို သိရင်ရပါပြီ။

$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
-----------	-----------	-----------

Octal to Binary

$$(127)_8 = (?)_2$$

1 = 001 (အပေါ်က ပေါ်လေးကိုကြည့်ပါ 1 ဆိုတော့ 4 နဲ့ 2 မယူပဲ 1 ပဲ ယူပါမယ်)

2 = 010 (အပေါ်က ပေါ်လေးကိုကြည့်ပါ ၂ ဆိုတော့ 4 နဲ့ 1 မယူပဲ 2 ပဲ ယူပါမယ်)

7 = 111 (7 ဆိုတော့ 4+2+1 အကုန်ပေါင်းမှ ရတာမို့ အကုန်ယူမယ်)

$$(127)_8 = (001\ 010\ 111)_2$$

Binary to Octal

$$(1111\ 011)_2 = (?)_8$$

1 111 011 (နောက်နောက်ပြီး ၃ လုံးတစ်ဖြတ် ဖြတ်ပျော်ပါမယ်။ လိုအပ်ရင် အရှေ့မှာ ၀ ထည့်ပါ)

001 = 1

111 = 4+2+1 = 7

011 = 2+1 = 3

$$(1111\ 011)_2 = (173)_8$$

Hexadecimal and Binary

Hexadecimal တစ်လုံးမှာ binary 4 လုံးရှိတာဖြစ်လို့ အောက်က ပေါ်လေးကို သိရင်ရပါပြီ။

$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
-----------	-----------	-----------	-----------

Hexadecimal to Binary

$$(A7)_{16} = (?)_2$$

A (10) = $8+2 = 1010$ (အပေါ်ကပေါ်လေးကို ကြည့်ပါ 10 ဆိုတော့ 8+2 ယူပါမယ်၊ 1,4 နေရာ မယူပါ)

7 = 0111 (7 ဆိုတော့ 4+2+1 မဲ့ 8 နေရာက 0 ကျန်တာ 1)

$$(A7)_{16} = (1010\ 0111)_2$$

Binary to Hexadecimal

$$(1111011)_2 = (?)_{16}$$

0111 1011 (နောက်ကနေ စပြီး 4 လုံးတစ်ဖျတ် ဖြတ်ပါမယ်။ 4 လုံးမပြည့်ရင် ရှေ့မှာ 0 ထည့်)

$$0111 = 0+4+2+1 = 7$$

$$1011 = 8+0+2+1 = 11 (\text{B})$$

$$(1111\ 011)_2 = (7B)_{16}$$

Hexadecimal and Octal

Hexadecimal နဲ့ Octal အချင်းချင်း အပြန်အလှန်ပြောင်းချင်ကြရင် Binary ကို ကြားခံပြီး
ပြောင်းလိုက်ပါတယ်။

Octal to Hexadecimal

$$(173)_8 = (?)_{16}$$

$$1 = 001, 7 = 111, 3 = 011$$

$$001 \ 111 \ 011 \Rightarrow 0000 \ 0111 \ 1011 \text{ (ရလာတဲ့ 3 လုံးတစ်တွဲကို hexadecimal)}$$

ပြောင်းတော့မှာမို့ အနောက်ကနေ 4 လုံး တစ်ဖြတ်ဖြတ်၊ အလုံးအရေတွက်
မကိုက်ရင် ရှုံးကနေ 0 တွေ ထည့်လိုက်ပါတယ်)

$$0000 = 0, 0111 = 7, 1011 = 11(B)$$

$$(173)_8 = (7B)_{16} \text{ (ရှုံးက 0 ပဲရလို့ တန်ဖိုးမရှိတော့ ဖြုတ်လိုက်ပါတယ်။)}$$

Hexadecimal to Octal

$$(7B)_{16} = (?)_8$$

$$7 = 0111, B = 11 = 8+2+1 = 1011$$

$0111 \ 1011 \Rightarrow 001 \ 111 \ 011 \text{ (ရလာတဲ့ 4 လုံးတစ်တွဲကို Octal ပြောင်းတော့မှာမို့
အနောက်ကနေ 3 လုံး တစ်ဖြတ်ဖြတ်၊ အလုံးအရေတွက် မကိုက်ရင် ရှုံးကနေ 0 တွေ
ထည့်လိုက်ပါတယ်)}$

$$001 = 1, 111 = 7, 011 = 3$$

$$(7B)_{16} = (173)_8$$

Numbering System ကို ဘယ်လိုတွေ အသုံးပြုကြသလဲ။

Character

ကျွန်ုမတို့ နေစဉ် ရောင်းတယ် ဝယ်တဲ့ စတဲ့ တွက်ချက်ကြတဲ့နေရာတွေမှာ အသုံးပြု နေကြတာက Decimal System ဖြစ်ပါတယ်။ Computer ကတော့ on/off ပဲရှိတာမို့ on ဆိုရင် 1, off ဆိုရင် 0 စသည်ဖြင့် သတ်မှတ်တာမို့ 0 နဲ့ 1 ခွင့်ပြုတဲ့ binary system ကို အသုံးပြုပါတယ်။

Computer မှာ အသုံးပြုတဲ့ လူတွေက လူနားလည်သလို အလုပ်လုပ်တယ်၊ နိုင်းတယ်၊ အဲဒါကို စက်က computer နားလည်တဲ့ စက်နားလည်တဲ့ binary 0,1 ပြောင်းပြီး အလုပ်လုပ်ပါတယ်။

နှမူနာအနေနဲ့ ပြောရရင် ကျွန်ုမတို့ 'A' ဆိုပြီးနှင့်လိုက်တယ်။ စက်ထဲမှာ A လို အလုပ်မလုပ်ပါဘူး။ A (A အကြီး)ရဲ့ သတ်မှတ်ထားတဲ့ တန်ဖိုး 65 ကို binary ပြောင်းပြီး 0100 0001 စက်ထဲမှာ အလုပ်လုပ်ပါတယ်။

$$A \rightarrow (65)_{10} \rightarrow (0100\ 0001)_2$$

B ဆိုရင် 66 ကို binary ပြောင်းပါမယ်။ စက်မှာရှိတဲ့ character တစ်ခုချင်းမှာ သတ်မှတ်ထားတဲ့ တန်ဖိုးတွေရှိပါတယ်။

Character	Decimal
A	65
B	66
...	
a	97
b	98

ကျန်တဲ့ characterတွေရဲ့ တန်ဖိုးတွေ သိချင်ရင် အောက်က link ကနေ ဝင်ကြည့်လိုပါတယ်။ အဲဒီထဲမှာ Decimal တန်ဖိုး၊ Octal တန်ဖိုးတွေပါပါတယ်။ Binary တို့ Hexadecimal တို့ ပြောင်းကြည့်ကြည့်ပါ။

https://en.wikipedia.org/wiki/List_of_Unicode_characters

Color

Web design ပိုင်းကို လေ့လာလို့ web site တွေ ရေးသားဖန်တီးတဲ့ အခါမှာ color ကိုဖော်ပြတဲ့ နေရာမှာ red, green, blue ဆိုတဲ့ အခြေခံ အရောင် ၃ ရောင်ပေါ်မှာ အလုပ်လုပ်ပါတယ်။ r (red), g(green), b(blue) တန်ဖိုးတွေဟာ 0 ကနေ 255 အထိ decimal တန်ဖိုး ခွင့်ပြုပါတယ်။

အနီရောင် လိုချင်ရင် - rgb (255, 0 , 0) လို့ ရေးလိုရသလို၊ အကြီးဆုံးခွင့်ပြုတဲ့ 255 ကို Hexadecimal ပြောင်းကြည့်ရင် FF ရပါတယ်၊ ဒါကြောင့် decimal တစ်လုံးမှာ hexadecimal 2 လုံးနဲ့ ရေးလိုရပါတယ်။

255 → FF, 0→00, 0 → 00

ပြီးရင် hexadecimal နဲ့ ရေးမယ်ဆိုရင် ရှေ့က # ထည့်ရေးရပါတယ်၊ ဒါကြောင့် web site မှာ color အနီလို ရေးချင်ရင် - rgb (255, 0 , 0) or #FF0000 လို့ ရေးလိုရပါတယ်။

အစိမ်းရောင် လိုချင်ရင် - rgb (0, 255, 0) or #00FF00

အပြာရောင် လိုချင်ရင် - rgb (0, 0 , 255) or #0000FF

အနီ နဲ့ အစိမ်းစပ်ရင် လိမ္မာ်ရပါတယ် အနုအရင်ကတော့ 0 ကနေ 255 အတွင်း အတိုးလျော့လုပ်ကြည့်ရင် အနု အရင့်ရပါတယ်။ ဥပမာ

rgb (255, 165, 0) => #FFA500 ,

rgb (238, 154, 0) => #EE9A00

အဲဒီနှစ်ခုရဲ့ အရောင် ကွာခြားချက်ကို ကြည့်ချင်ရင် internet မှာ rgb တန်ဖိုးဖြစ်ဖြစ် hexadecimal တန်ဖိုးဖြစ်ဖြစ် ရိုက်ရှာလိုပါသလို အောက်က link မှာ ဝင်ကြည့်လိုလည်း ရပါတယ်။ အရောင်စပ်တာက စိတ်ဝင်စားဖို့ အင်မတန် ကောင်းပါတယ်။ အရောင်စပ်တဲ့ အခြေခံလည်း သဘောတရားတွေလည်း နားလည်း hexadecimal ပြောင်းတာ မှန်မှန်လဲ စစ်ရအောင် အဲဒီ link ကနေ အရောင်တွေ RGB တန်ဖိုးတွေ hexadecimal တန်ဖိုးတွေ လေ့လာ ကြည့်ကြည့်ပါ။

<https://cloford.com/resources/colours/500col.htm>



Chapter အနှစ်ချုပ်

- IT ကို လေ့လာရင် Numbering System ဆိုတာ ရှောင်လွှဲလို့ မရပါဘူး။ နေရာအစုံမှာ အသုံးပြုပါတယ်။ အဲဒီထဲကမှ character code နဲ့ color နှစ်ခုပဲ နမူနာပြ သွားတာဖြစ်ပါတယ်။ ရအောင်လုပ်ဖို့ လိုပါတယ်၊ စာအုပ်ထဲ ချွေက်ပါနော်။

“Numbering System အတွက်

နမူနာတွေရော၊ လေ့ကျင့်ခန်းတွေပါ

အကုန်လုပ်ခဲ့ရဲ့လားတော့မသိဘူး။

ပြီးမှ လုပ်တော့မယ်ဆိုရင် မလုပ်ဖြစ်တာများပါတယ်။

သိပြီးသားတွေပါကွာလို့ ထားခဲ့ရင်လဲ

မကြောခင်မေ့မှာဖြစ်ပါတယ်။

ဒီတော့ တစ်ခန်းတစ်ခန်းချင်း တစ်ခါတည်း

အပြန်ဖြတ်ခဲ့ပေးပါ”

အခန်း (J)

သချို့ဆိုင်ရာ အခြေခံများ

Integer, Double

ကိန်းပြည့်ကို integer လို့ ခေါ်ပြီးတော့၊ ဒသုမကိန်းကို float or double လို့ ခေါ်ပါတယ်။ INT() ကတော့ ဒသုမကိန်းတွေ ဒသုမနောက်ပိုင်းဖြတ်ချုပြီး ကိန်းပြည့်ပြောင်းတာဖြစ်ပါတယ်။

$$\text{INT}(3.534) = 3, \text{INT}(-45.67) = -45$$

Absolute Values

Absolute ကတော့ အပေါင်းကိန်းဖြစ်ဖြစ်၊ အနှုတ်ကိန်းဖြစ်ဖြစ် ပကတိတန်ဖိုး(အပေါင်း)ကို ပြောင်းချင်ရင် သုံးပါတယ်။

$$|34| = 34, |-4.5| = 4.5, |-67| = 67$$

Floor, Ceiling and Round Functions

$$\lfloor x \rfloor = \text{floor of } x, \lceil x \rceil = \text{ceiling of } x$$

ဒသုမကိန်းဆိုတာ အမြဲ ကိန်းပြည့်နှစ်ခု အကြေားမှာ ရှိပါတယ်။ Floor ဆိုရင် အဲဒီကိန်းပြည့်နှစ်ခုထဲက ငယ်တာကိုယူပါတယ်။ ceiling ဆိုရင် အဲဒီကိန်းပြည့်နှစ်ခုထဲက ကြိုးတာကိုယူပါတယ်။ round ကတော့ ကျွန်မတို့ လုပ်နေကြအတိုင်းပဲ .5 နဲ့ အထက်ဆိုရင် တိုးယူပြီး၊ .5 အောက်ဆို လျော့ယူတာပါ။ အလွယ်ပြောရရင်တော့ နီးတဲ့ဘက်ကို ယူတာပါ။

$\lfloor 3.2 \rfloor = 3$, $\lceil 3.2 \rceil = 4$, round (3.2) = 3

$\lfloor 3.7 \rfloor = 3$, $\lceil 3.7 \rceil = 4$, round (3.7) = 4

$\lfloor -3.5 \rfloor = -4$, $\lceil -3.5 \rceil = -3$, round (-3.5) = -4

$\lfloor -3.1 \rfloor = -4$, $\lceil -3.1 \rceil = -3$, round (-3.1) = -3

(-3 နှင့် -4 မှာ -4 ကင်ယ်တာဖြစ်လို့ floor မှာ -4 ရပြီး, ceiling မှာ -3 ရတာဖြစ်ပါတယ်၊ round က ထိုးစံအတိုင်း.5 အောက်က ဒီဘက် .5 နဲ့ အထက်က ဟိုဘက် ဆိုပြီး နီးတာကိုယူတာပါ။)

ဒီလောက်ဆိုရင်တော့ Floor, Ceiling နှင့် Round ဟာလည်း အသေမကိန်းတွေကို
ကိန်းပြည့်ပြောင်းတဲ့ နေရာမှာ သုံးတာ သိလောက်ပါပြီ။

Arithmetic Operators

+	$2 + 3 = 5$
-	$5 - 2 = 3$
*	$2 * 5 = 10$
/	$7/2 = 3$ (စားလဒ်)
Mod	$7 \bmod 2 = 1$ (အကွင်း)

$25 \bmod 5 = 0$, $9 \bmod 5 = 4$

ဒီနေရာမှာ / operator လေးက programming language ပေါ်မှတည်ပြီး အလုပ်လုပ်ပုံ
ကွဲပြားတတ်တာလေး သတိထားစေချင်ပါတယ်။ Java မှာ ဆိုရင် ကိန်းပြည့်ကို ကိန်းပြည့်နဲ့ စားရင်
ကိန်းပြည့်ရပြီး (e.g. $5/2 = 2$)၊ အနည်းဆုံး အသေမ တစ်ခုပါတာနဲ့ အသေမ ထွက်ပါတယ် (e.g. $5.0/2 =$

2.5, $5/2.0 = 2.5$, $5.0/2.0 = 2.5$) || ဒါပေမဲ့ PHP မှာဆိုရင်တော့ / ဟာ အမြဲ အသာမ ထွက်ပါတယ်။ (e.g. $5/2 = 2.5$, $5.0/2 = 2.5$)

Summation Symbol

ပေါင်းတာကို သချိုာမှာ Σ (summation) နဲ့ ကိုယ်စားပြုပါတယ်။

n

$$\sum_{i=1}^n a_i b_i = a_1 b_1 + a_2 b_2 + a_3 b_3 + \dots + a_n b_n$$

summation symbol Σ အောက်မှာ i သည် 1 စမယ်ပြောတယ်၊ Σ အပေါ်ကို $\overline{\text{ကြည့်}}$ လိုက်တော့ n အထိဘွားမယ်၊ ဒါကြောင့် ဖြန့်လိုက်တော့ i တွေနေရာမှာ 1 က စတည့်လိုက်တော့ $a_1 b_1$, ပြီးရင် 2 ထည့်လိုက်တော့ $a_2 b_2$, အဲဒီလိုနဲ့ n ထိ လိုပြောတဲ့အတွက် $a_n b_n$ ထိသွင်းပါတယ်။ Σ ဖြစ်တဲ့အတွက် တစ်ခုနဲ့ တစ်ခု $\overline{\text{ကြား}}$ အပေါင်း လက္ခဏာ ထည့်လိုက်တော့ ညာဘက်ခြမ်းမှာ ဖြန့်ပြထားတဲ့ အတိုင်း $a_1 b_1 + a_2 b_2 + a_3 b_3 + \dots + a_n b_n$ ဆိုပြီး ထွက်လာတာဖြစ်ပါတယ်။

$$\sum_{i=1}^n c = c + c + c + \dots + c$$

ဒါ Σ မှာက ထူးဆန်းပါတယ်။ i တန်ဖိုးသည် 1 ကနေစမယ် n အထိလိုပြောထားပေမဲ့ Σ နောက်မှာက c ပဲပါပါတယ်၊ i အစားထိုးစရာ မပါပါဘူး။ ဆို $\overline{\text{ကြပါစို့}}$ n = 3 လို့။ c + c + c ဖြစ်ပါမယ်။ c ကို 3 ခါ ပေါင်းတာဟာ c^3 နဲ့ အတူပါပဲ။ တကယ်လို့ n=5 ဆိုခဲ့ရင် c+c+c+c+c ဖြစ်ပါမယ်။ အဲဒါကလည်း c^5 နဲ့ အတူပါပဲ။ ထပ်ခါ ထပ်ခါ ပေါင်းတာက မြောက်တာနဲ့တူပြီး၊ ထပ်ခါ ထပ်ခါ နှုတ်တာက စားတာနဲ့ အတူတူပါပဲ။

5

$$\sum a^2 = 1^2 + 2^2 + 3^2 + 4^2 + 5^2 = 1 + 4 + 9 + 16 + 25 = 55$$

a=1

summation symbol Σ အောက်မှာ a သည် 1 စမယ်ပြောတယ်၊ Σ ပေါ်ကို $\overline{\text{ကြည့်}}$ လိုက်တော့ 5 အထိဘွားမယ်၊ ဒါကြောင့် ဖြန့်လိုက်တော့ a နေရာ 1 အစားထိုးတော့ 1^2 , 2 အစားထိုးတော့ 2^2 အဲဒီလိုနဲ့

သူပြောတဲ့ 5 အထိ အစားထိုးသွားပါတယ်။ \sum ဆိုတဲ့ အတွက် ကြားထဲမှာ အပေါင်းလက္ခဏာထည့်သွားပါတယ်။ ဒီလောက်ဆို \sum (summation symbol) အကြောင်းနားလည်လောက်ပါပြီ။



ဒါဆိုရင် အောက်က ပြထားတာလေး နှစ်ပုဒ်ကို \sum သုံးပြီး ရေးကြည့်ကြည့်ပေးပါ။

$$-1 + -2 + -3 + -4 + -5 + -6 + -7$$

$$1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{100}$$

ပေါင်းခြင်း Formula များ



$$\sum_{i=0}^n i = \sum_{i=1}^n i = \frac{n(n+1)}{2}$$

၁,၂,၃ စသည်ဖြင့် အစဉ်လိုက်ဖြစ်တဲ့ ကိန်းတွေ ပေါင်းတဲ့အခါမှာ အလွယ်သုံးလိုဂုဏ်တဲ့ formula ဖြစ်ပါတယ်။ မှန်သလား မမှန်ဘူးလား တွက်ကြည့်ကြရအောင်လို့ 1 ကနေ 5 ထိ ပေါင်းကြည့်ပါမယ်။

$$1 + 2 + 3 + 4 + 5 = 15$$

ပုံသေနည်းနဲ့ ထည့်တွက်ကြည့်ပါမယ်။ n ဆိုတာ နောက်ဆုံးကဏ္ဍး (အကြီးဆုံးကဏ္ဍး) တန်ဖိုးဖြစ်ပါတယ်။ ၁ ကနေ ၅ အထိပေါင်းတဲ့ နေရာမှာ ၅ သည် အကြီးဆုံး နောက်ဆုံးကိန်းဖြစ်လို့ n တန်ဖိုးသည် 5 ဖြစ်ပါမယ်။ ဒါကြောင့်

$$n(n+1)/2 = 5(5+1)/2 = (5*6)/2 = 30/2 = 15 \text{ ရပါတယ်။}$$



$$\sum_{i=m}^n i = \frac{n(n+1)}{2} - \frac{m(m-1)}{2}$$

1 ကစတာမျိုးမဟုတ်ပဲ စခင်တဲ့ ဂဏန်းကနေစပြီး အစဉ်လိုက်ဖြစ်တဲ့ ကိန်းဂဏန်းတွေ ပေါင်းတဲ့အခါမှာ အလွယ်သုံးလိုရတဲ့ formula ဖြစ်ပါတယ်။ Formula မှာတော့ စတဲ့ ဂဏန်းကို m လိုပေါ်ပြီး၊ နောက်ဆုံးဂဏန်းကို n လို သတ်မှတ်ထားပါတယ်။ မှန်သလား မမှန်ဘူးလား တွက်ကြည့်ကြရအောင်လို့ 5 ကနေ 10 ထိ ပေါင်းကြည့်ပါမယ်။ 5 သည် စတဲ့ဂဏန်းဖြစ်လို့ m ဖြစ်ပြီး၊ 10 ကတော့ နောက်ဆုံးကိန်းဖြစ်လို့ n ဖြစ်ပါမယ်။ ပုံသေနည်းထဲ အရင်ထည့်တွက်ကြည့်ပါမယ်။

$$10(10+1)/2 - 5(5-1)/2$$

$$= (10*11)/2 - (5*4)/2 = 55 - 10 = 45$$

ပုံမှန်အတိုင်းတွက်ကြည့်ပါမယ်။

$$5 + 6 + 7 + 8 + 9 + 10 = 45$$



$$\sum_{i=0}^n i^2 = \frac{n(n+1)(2n+1)}{6}$$

နှစ်ထပ်ကိန်းတွေကို အစဉ်လိုက် ပေါင်းတဲ့ ပုံသေနည်းဖြစ်ပါတယ်။ ပုံသေနည်းမှာတော့ 0 စတယ် ပြောထားပေမည့် အမှန်ကတော့ 1 ကနေ စလည်း အတူတူပါပဲ။ 0 က တန်ဖိုးမရှိလိုပါ။ ဒီတော့ 1 နှစ်ထပ် ကနေ 5 နှစ်ထပ် အထိ ပေါင်းကြည့်ကြရအောင်။ 5 နှစ်ထပ်အထိလို့ ပြောတဲ့အတွက် n တန်ဖိုးသည် 5 ဖြစ်ပါမယ်။ ပုံသေနည်းထဲအရင်ထည့်တွက်ကြည့်ရအောင်ပါ။

$$5(5+1)(2*5+1)/6 = (5 * 6 * 11)/6 = 330/6 = 55$$

ပုံမှန်တွက်ကြည့်လည်း 55 ပါပဲ။ $1^2 + 2^2 + 3^2 + 4^2 + 5^2 = 1 + 4 + 9 + 16 + 25 = 55$



$$\sum_{i=0}^n i^3 = \left(\frac{n(n+1)}{2}\right)^2$$

ဒါကတော့ ၃ ထပ်ကိန်း အစဉ်လိုက် ပေါင်းတဲ့ Formula ဖြစ်ပါတယ်။ ကဲ ၁ ကနေ ၅ အထိ ၃ ထပ်ကိန်းတွေ ပေါင်းကြည့်ကြရအောင်ပါ။

$n = 5$

$$[(5*6)/2]^2 = 15^2 = 225$$

ပုံမှန်တွက်ကြည့်ကြရအောင်။ $1 + 8 + 27 + 64 + 125 = 225$



$$\sum_{i=m}^n 1 = n + 1 - m$$

ဒီ Formula လေးကတော့ အရေအတွက် တွက်နဲ့နေရာမှာ အလွန် အသုံးဝင်ပါတယ်။ နှမှနာပြောရရင် ၁ ကနေ ၁၀ အထိ ဘယ်နှစ်လုံးရှိသလဲဆို လူတိုင်းအလွယ်သိပါတယ် ၁၀ လုံးပေါ့။ ဒါဆို ၂ ကနေ ၅ ဆိုရင်ကော့ ဘယ်နှစ်လုံး ရှိသလဲဆို နည်းနည်း ကြောင်သွားတတ်ပါတယ်။ ဒီပုံသေနည်းနဲ့ တွက်လို့ရပါတယ်။ စတာကာ n , ဆုံးတာကာ n ဒီတော့

$$5+1 - 2 = 6 - 2 = 4 \text{ ဆိုတဲ့ အတွက် 4 လုံးပါ။}$$



ဒီမှာ ကိန်းစဉ်တန်းနည်းနည်းလေး လေးငါး ဆယ်လုံးနဲ့ စမ်းပြနေလို့တာ ပုံမှန်တွက်တဲ့နည်းက တွက်လို့ ရနေသေးတာပါ။ တကယ်လို့ ၁ ကနေ ၁၀၀၀ တို့၊ ၁ ကနေ ၁၀၀၀၀ တို့ဆို ပုံမှန်ချတွက်ဖို့ မလွယ်ပါဘူး။ Calculator နဲ့ တွက်မယ်ဆိုရင်တောင် ၁ ကနေ ၁၀၀၀ ထိဆိုရင် ရှိက်နေရတာကို မလွယ်ကူလှပါဘူး။ အချိန်အတော်ယူပါလိမ့်မယ်။ ဒီအတွက် ယခုပြောပြတဲ့ အခြေခံ Formula တွေက သိထားဖို့ လိုပါတယ်။ လူတွေမှာ တစ်ခုချင်း ရှိက်ပေါင်းနေရင် ကြာပြီး Formula သုံးရင် မြန်သလို computer မှာလဲ အတူတူပါပဲ။ ဒါကြောင့် တစ်ခုမှတ်ထားစေချင်တာက computer program

တွေရေးတဲ့ အခါမှာ ထပ်ခါ ထပ်ခါ တစ်ခုချင်းတွက်တာထက် Formula သုံးတွက်တာ မြန်တဲ့အတွက် တတ်နိုင်သမျှ Formula သုံးပြီး program တွေ ရေးသင့်ကြောင်းပါ။



ကဲ အောက်က ပုစ္စာလေးတွေကိုတော့ လေ့ကျင့်တဲ့အနေနဲ့ တွက်ချက်ကြည့်ကြည့်ပါ။



$$1 + 2 + 3 + 4 + \dots + 10000$$



$$45 + \dots + 999$$



49 ကနေ 512 အထိဆိုရင် ကိန်းကဏ္ဍး ဘယ်နှစ်လုံးရှိလဲ တွက်ချက်ပါ။



1 မှ 100 ထိ နှစ်ထပ်ကိန်းများပေါင်းလဒ်နှင့် သုံးထပ်ကိန်းများပေါင်းလဒ်ကို ရှာပေးပါ။



$$\text{Find } \sum_1^{20} 5 \text{ and } \sum_{45}^{70} 10.$$

Factorial

Factorial ကို ! နဲ့ ကိုယ်စားပြုပြီး $n!$ (n factorial) ဆိုတာ n ကနေ 1 အထိ ငြောက်သွားတာပါ။

$0! = 1$ တန်ဖိုးက 1 ဖြစ်ပါတယ်။

$$n! = n * (n-1) * \dots * 3 * 2 * 1$$

$$0! = 1, 1! = 1$$

$$2! = 2 * \boxed{1} = 2 * 1! = 2$$

$$3! = 3 * \boxed{2 * 1} = 3 * 2!$$

$$4! = 4 * \boxed{3 * 2 * 1} = 4 * 3!$$

ဒီကနေ မှတ်လိုရတာက $n! = n(n-1)!$ ဖြစ်ပါတယ်။ ပြောင်းပြန်ပြန်တွက်လိုလည်း ရပါသေးတယ်။

$$\begin{aligned} 4! &= 4 * 3 * 2 * 1 \\ 3! &= 3 * 2 * 1 = 4! / 4 \\ 2! &= 2 * 1 = 3! / 3 \end{aligned}$$

$$n! = \frac{(n+1)!}{(n+1)}$$

Permutation and Combination

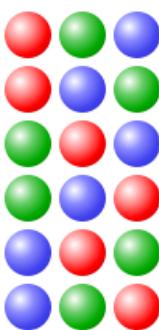
Permutation ဆိုတာ n ခုရှိတဲ့ elements လေးတွေကို ကိုယ်လိုသလောက်ယူပြီးတဲ့တာဖြစ်တပါတယ်။ Permutation က order ကို ထည့်ပြီး စဉ်းစားပါတယ်။ Combination ကတော့ order ကို ထည့်မစဉ်းစားပါဘူး။

Permutation

$${}_nP_r = \frac{n!}{(n-r)!}$$

$$n = 3, r = 3$$

$$3! / (3-3)! = 3! / 0! = 6/1 = 6$$



Combination

$$C(n, k) = \binom{n}{k} = \frac{n!}{k!(n-k)!}$$

$$N = 3, k = 3$$

$$C(3,3) = 3! / [3! (3-3)!] = 3! / 3! = 1$$

	${}_nP_r = \frac{n!}{(n-r)!}$	$C(n, k) = \binom{n}{k} = \frac{n!}{k!(n-k)!}$
3 ခုရှိတာ	ABC, ACB	ABC
3 ခုစီ ထဲ	BAC, BCA CAB, CBA	$C(3,3) = 3! / (3! * 0!) = 3! / 3! = 1$
	${}_3P_3 = 3! / 0! = 3 * 2 * 1 = 6$	

3ခုရှိတာ	AB, AC,	AB, AC, BC
2 ခုစီတဲ့	BA, BC CA, CB	$C(3,2) = 3! / [2! (3-2)!]$ $= (3 * 2!) / (2! * 1)$ $= 3$
	Order ကို ထည့်သွင်းစဉ်းစားတာ ဖြစ်တဲ့အတွက် AB နဲ့ BA က မတူပါဘူး။	Orderကို ထည့်သွင်းမစဉ်းစား တဲ့အတွက် AB နဲ့ BA က တူတယ်လို့ ယူဆပါတယ်။

Fibonacci Series



Fibonacci Series ဆိုတာ ရှေ့ နှစ်လုံးပေါင်းရင် နောက်တစ်လုံးရတာပါ။

$$F_3 = F_2 + F_1$$

$$F_4 = F_3 + F_2$$

$$F_n = F_{n-1} + F_{n-2}$$

$F_0=0$, $F_1=1$ လို့ ထားကြည့်မယ်ဆိုရင် Fibonacci Series ဟာ အောက်ပါအတိုင်း ရပါမယ်။ $F_2 = F_1 + F_0$, $F_2 = 1 + 0 = 1$

$F_0, F_1, F_2, F_3, \dots$

0, 1, 1, 2, 3, 5, 8, 13, 21, 34,

Exponents and Logarithms

$$2^3 = 8, \quad 2^{-3} = 1/2^3 = 1/8 = 0.125$$

$$27^{2/3} = (3^3)^{2/3} = 3^2 = 9$$

ဒီနေရာမှာ သတိထားစေခဲင်တာက - (အနှစ်) တန်ဖိုးကို $1/3$ ထပ်ရှာလို့ရပါတယ်။ ဘာကြောင့်လဲဆိုရင် အနှစ်ကို မကိန်းထပ်တင်ရင် အနှစ်ရလို့ အနှစ်ကိန်းကို $1/3, 1/5$ စတဲ့ $\frac{1}{\text{မကိန်း}}$ ထပ်တွေ တင်ပြီး တွက်လို့ရပါတယ်။

$$\text{အနှစ်} * \text{အနှစ်} = \text{အပေါင်း}$$

$$\text{အနှစ်} * \text{အပေါင်း} = \text{အနှစ်}$$

$$\text{အနှစ်} * \text{အနှစ်} * \text{အနှစ်} = \text{အပေါင်း} * \text{အနှစ်} = \text{အနှစ်}$$

$$-27^{1/3} = (-3^3)^{1/3} = -3$$

သို့သော် $-4^{1/2}$ ဆိုရင်တော့ အဖြေရှာလို့ရမှာ မဟုတ်ပါဘူး။ အနှစ်ကို စုံကိန်းထပ်တင်ရင် + ပဲ ရပါတယ်။

ဒါကြောင့် အနှစ်ကို $\frac{1}{\sqrt[n]{\cdot}}$ ထပ်တင်ပြီး တွက်လို့မရပါဘူး။

$y = \log_b x \rightarrow b^y = x$

$$\log_2 8 = 3 \rightarrow 2^3 = 8$$

$$\log_{10} 100 = 2 \rightarrow 10^2 = 100$$

$$\log_{10} 0.001 = -3 \rightarrow 10^{-3} = 0.001$$

$$\log_b 1 = 0 \rightarrow b^0 = 1$$

$$\log_b b = 1 \rightarrow b^1 = b$$



အောက်က ပုစ္စာလေးတွေ လေ့ကျင့်ကြည့်ပါ။

1. Find floor, ceiling and round of the following numbers.

-7.8, 7.4, $\sqrt[3]{30}$, π

2. $29 \bmod 8$, $-372 \bmod 8$, $-36 \bmod 3$

3. 2^{-5} , $8^{2/3}$, $16^{-4/3}$, $\log_2 16$, $\log_{10} 10000$, $\log_2 (1/8)$, $\log_2 0.125$

4. အောက်ပါတို့ကို \log ပုံစံဖြင့် ပြန်ရေးပြပါ။

a) 10^4

b) 2^5

c) $1/1000$

d) $1/27$

5. Find $6!$, $0!$, $1!$, $-1!$ and $8!$.

6. Find ${}_4P_2$, $C(4,2)$, ${}_4P_3$, $C(4,3)$.

7. Decimal number မှာ 0 to 9 ဆိုပြီး ခွင့်ပြုတာ 10 လုံးရှိပါတယ်။ 3 digit lottery မှာ ဂဏန်းအတွဲပေါင်း ဘယ်နှစ်လုံးရမလဲဆိုတာ တွက်ပေးပါ။

8. မီးခံသော် မှာ 0 to 9 ဆိုပြီး ခလုတ်လေး 10 ခုရှိပါတယ်။ key ဟာ 7 လုံးထားရတယ်ဆိုပါတော့။ ဒါဆို key ဖြစ်နိုင်တဲ့ 7 လုံး အတွဲပေါင်း မည်မျှရှိပါသလဲ။ တကယ်လို့ key ဟာ 8 လုံးထားရတယ်ဆိုရင်ရော ဘယ်နှစ်တဲ့ ရနိုင်ပါသလဲ။

9. F, R, I, E, N, D ဆိုတဲ့ character 6 လုံးကို ပေးပြီး စကားလုံးဆက်တာမှာ FRIEND လို့ မဟုတ်ပဲမှားဆက်တာ ဘယ်နှစ်ခုရှိနိုင်ပါသလဲ။

10. လုပ်ငန်းခွင်မှာ လူ 15 ယောက်ရှိပါတယ်။ သူတို့ထဲက 5 ယောက်ကိုအဖွဲ့ဖွံ့ဖြိုး အလုပ်တစ်ခုလုပ်ဖို့ လွတ်ပေးရပါမယ်။ ရွေးချယ်စရာ ဘယ်နှစ်ဖွဲ့ ရှိပါမည်နည်း။

11. $F_0=1$, $F_1 = 2$ and then Find F_0 to F_{15} .



Chapter အနှစ်ချုပ်

အခန်းအကျယ်ကြီး တစ်ခန်းထဲ ခုံတွေ တန်းစီပြီး ထည့်မယ်။ ဘယ်နှစ်ခု ဆုံးမယ်မသိဘူး။ သချို့မရတဲ့သူက ဘယ်နှစ်ခုဆုံးလဲ သိရအောင် ခုံတွေအများကြီးကို အချိန်ကုန်ခံ၊ လူပင်ပန်းခံပြီး မ ရွှေ့ပြီး နေရာချကြည့်တယ်။ သချို့ရတဲ့သူကတော့ အခန်းအကျယ်ကိုတိုင်းတယ်၊ ခုံတစ်ခုရဲ့ဆိုဒ်ကိုတိုင်းတယ်၊ တစ်ခုနဲ့ တစ်ခုကြား ဘယ်လောက်နေရာ ခြားမလဲ တိုင်းတယ်။ ပြီးတော့ စာရွက်ပေါ်ချွောက်လိုက်တယ်။ အဖြေတန်းထွက်လာတယ်။

ပြီးတော့ လူတွေ တွက်ချက်တဲ့အခါမှာလဲ ထပ်ခါ ထပ်ခါ တွက်ချက်တာက ကြာတတ်၊ မှားတတ်သလို Program တစ်ခုကို ရေးသားတဲ့အခါမှာ ထပ်ခါ ထပ်ခါ လုပ်တာမျိုးတွေက တွက်ချက်တဲ့ အချိန် ကြာဖော်တယ်။ Formula နဲ့ တွက်တာက မြန်ဖော်တယ်။ ဒါကြောင့် ထပ်ခါ ထပ်ခါ တွက်ချက်တာတွေ အစား အလျင်းသင့်သလို့ Formula တွေ ကို အသုံးပြုသင့်ပါတယ်။ ဒါကြောင့် Formula လေးတွေ သိထားသင့်ပါတယ်။ ပြောပြီးသား ထပ်ပြောရရင် စာအုပ်ထဲ ချွောက်ကြည့်ပါနော်။ ပြီးတော့ ထပ်ခါ ပေါင်းတာက မြောက်တာနဲ့ တူပြီး၊ ထပ်ခါ ထပ်ခါ နှုတ်တာက စားတာနဲ့ တူတာလေးလဲ သတိချပ်ပါ။

“စာတွက်နေတဲ့အချိန် စားပွဲမှာ ထိုင်၊
ဘေးမှာ Coffee လေးတစ်ခွက် နဲ့
သီချင်းလေးက ဖွင့်ထား
သိပ်မိုက်တာပဲ”

အခန်း (၃)

Boolean algebra

Logical Operators

Operators	အလုပ်လုပ်ပုံ	နမူနာ
And (\wedge)	အားလုံးမှန်မှ အလုပ်လုပ်တယ်။ တစ်ခုမှားတာနဲ့ အလုပ်မလုပ်တော့ဘူး။	ဘာသာစုံအားလုံးအောင်မှ စာမေးပွဲအောင်
Or (\vee)	အနည်းဆုံး တစ်ခုမှန်တာ နဲ့ အလုပ်လုပ်တယ်။	တစ်ဘာသာကျတာနဲ့ စာမေးပွဲကျ
Not (!)	ဆန့်ကျင်ဘက် ပြောင်းချင်တဲ့ အခါမှာသုံးတယ်။	True ကို not လုပ်ရင် false False ကို not လုပ်ရင် true And ကို not လုပ်ရင် or Or ကို not လုပ်ရင် and
Exclusive Or (xor)	တူရင် 0(False)၊ မတူရင် 1(True)	

Truth Table

Logical Operator တွေဟာ boolean တွေပေါ်မှာ အလုပ်လုပ်လေ့ ရှိပါတယ်။ Boolean ဆိုတာ Numbering System မှာ ပြောခဲ့တဲ့ binary လိုပဲ ဖြစ်နိုင်ချေ နှစ်ခုပဲ ရှိပါတယ်။ 1 (True) နှင့် 0(False) တို့ပဲဖြစ်ပါတယ်။ 1 ဟာ true နှင့် တူပြီး၊ 0 ဟာ false နဲ့ တူပါတယ်။ 01 နဲ့ ရေးလည်း ရသလို့ True False နဲ့ ရေးလည်းရပါတယ်။

- A လိုပြောလိုက်ရင် A တစ်ခုထဲမဲ့ 2^1 ဖြစ်နိုင်ချေ နှစ်ခုပဲ ရှိပါတယ်။
- A, B လိုပြောလိုက်ရင် A, B နှစ်ခုဖြစ်တာကြောင့် 2^2 ဖြစ်နိုင်ချေ 4 ရှိပါတယ်။

- A, B, C လိုပြောလိုက်ရင် A, B, C သုံးခုဖြစ်တာကြောင့် 2^3 ဖြစ်နိုင်ချေ 8 ရှိပါတယ်။

Not A

A	Not A (!A)
0	1
1	0

ဒီလို table လေးဆဲတာကို truth table လို ခေါပါတယ်။ A ဆိတာလေး တစ်ခုပဲ ပါလို ဖြစ်နိုင်ချေ စုစုပေါင်း 2 ခု ရှိပါတယ်။ 2 ခုကို တစ်ဝက်ဝက်လိုက်တော့ 1 ခုရပါတယ်။ ဒါကြောင့် A ကို 0 တစ်လုံး၊ 1 တစ်လုံး ရေးမှာ ဖြစ်ပါတယ်။ A က 0(False) ဆိုရင် Not A က 1(True), A က 1(True) ဆိုရင် Not A က 0(False) ရေးမှာ ဖြစ်ပါတယ်။

A and B, A or B, A xor B

A, B လိုပြောလိုက်ရင် A, B နှစ်ခုဖြစ်တာကြောင့် 2^2 ဖြစ်နိုင်ချေ 4 ရှိပါတယ်။ 4 ခုကို တစ်ဝက်ဝက်တော့ 2 ခုရပါတယ်။ ဒါကြောင့် A ကို 0(False) နှစ်လုံး၊ 1(True) နှစ်လုံး ရေးမှာ ဖြစ်ပါတယ်။ 2 ခုကို ထပ်ပြီး တစ်ဝက်ဝက်တော့ 1 ဒါကြောင့် B ကို 0(False) တစ်လုံး၊ 1(True) တစ်လုံး ထပ်ထပ် ရေးရမှာ ဖြစ်ပါတယ်။

A	B	A and B	A or B	A xor B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

- And ကတေသ့ အားလုံးမှန်မှ True(1)
- Or ကတေသ့ အနည်းဆုံး တစ်ခုမှန်တာနဲ့ True(1)
- Xor ကတေသ့ တူရင် 0(False)၊ မတူရင် 1(True) ဖြစ်ပါတယ်။

A, B, C သုံးခုအတွက် Truth Table ဆွဲပြပါမယ်။ A, B, C လို့ပြောလိုက်ရင် A, B, C သုံးခုဖြစ်တာကြောင့် 2³ ဖြစ်နိုင်ချေ 8 ရှိပါတယ်။ A အတွက် 8 ခုကို တစ်ဝက်ဝက်တော့ 4 ခုရမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် A အတွက် 0(False) 4 ခု၊ 1(True) 4 ခုရေးမှာ ဖြစ်ပါတယ်။ B အတွက် 4 ခုကို တစ်ဝက်ထပ်ဝက်တော့ 2 ရပါတယ်။ ဒါကြောင့် B အတွက် 0(False) 2 ခု၊ 1(True) 2 ခုဆီ ထပ်ထပ် ရေးမှာ ဖြစ်ပါတယ်။ C အတွက် 2 ခုကို တစ်ဝက်ထပ်ဝက်တော့ 1 ရပါတယ်။ ဒါကြောင့် C အတွက် 0(False) 1 ခု၊ 1(True) 1 ခု ထပ်ထပ် ရေးမှာ ဖြစ်ပါတယ်။

A	B	C	A and B and C	A or B or C	A xor B xor C
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	1
0	1	1	0	1	1
1	0	0	0	1	1
1	0	1	0	1	1
1	1	0	0	1	1
1	1	1	1	1	0

ဒီလောက်ဆိုရင် Truth Table ကို နားလည်လောက်ပြီ ထင်ပါတယ်။

Rules in Boolean algebra

De Morgan's Theoremlaw

စာမေးပွဲတစ်ခုမှာ ဘာသာရပ် ၃ ခု ဖြေရတယ်၊ ၃ ဘာသာစလုံးအောင်မှ စာမေးပွဲအောင်ပါမယ်၊ တစ်ဘာသာကျတာနဲ့ စာမေးပွဲ ကျမယ် ဆိုပါတော့။ ဒါဆို အောင်တယ်ဆိုတာ ၃ ဘာသာလုံးအောင်မှ အောင်တာ၊ အားလုံးအောင်မှ အောင်တာဖြစ်တဲ့အတွက် and နဲ့ ရေးရပါမယ်၊ ကျတာက တစ်ဘာသာ ကျရင် ကျတာဖြစ်တဲ့အတွက်၊ တစ်ခုဖြစ်တာနဲ့ လုပ်တာဖြစ်လို့ ကျတာကိုရေးရင် or နဲ့ရေးရပါမယ်။

Not(Inversion) ဆိုတာကတော့ ဆန့်ကျင်ဘက်ပြောင်းတာပါ။ true ကို not လုပ်ရင် false ဖြစ်ပြီး၊ false ကို not လုပ်ရင် true ရပါတယ်။ ထပ်မံတ်ရမှာက and ကို not လုပ်ရင် or ရပြီး၊ or ကို not လုပ်ရင် and ရပါတယ်။

အပေါ်ကပြောခဲ့တဲ့ နမူနာကို ကြည့်ရင်သိနိုင်ပါတယ်။ အောင်တာကို ရေးရင် and, သူ့ရဲ့ဆန့်ကျင်ဘက်(not လုပ်လိုက်ရင်) စာမေးပွဲကျတာ၊ ကျတာက က or နဲ့ရေးရတယ်။ ထို့ အတူပဲ စာမေးပွဲကျတာက or, သူ့ရဲ့ ဆန့်ကျင်ဘက်(not လုပ်လိုက်ရင်) အောင်တာ၊ အောင်တာက and နဲ့ရေးရတယ်။ ဒီလိုပုံစံပါ။ သဘောကတော့ not နဲ့ ဝင်မြောက်လိုက်တဲ့ သဘောပါပဲ။

$$! (A \wedge B) = !A \vee !B$$

$$! (A \vee B) = !A \wedge !B$$

ပထမ ပြထားတဲ့ $! (A \wedge B) = !A \vee !B$ ကို ညီကြောင်း Truth table ကိုသုံးပြီး သက်သေပြပါမယ်။

A	B	$A \wedge B$	$!(A \wedge B)$	$!A$	$!B$	$!A \vee !B$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

အဖြောက်ခုတူနေတာ တွေ့ရမှာ ဖြစ်ပါတယ်။ $! (A \vee B) = !A \wedge !B$ ကိုလည်း Truth table ဆဲပြီး
ညီကြာင်းသက်သေးပြေားပါ။

INVERSION law

$$! (!A) = A$$

A	$!A$	$!(!A)$
0	1	0
1	0	1

Commutative law

$$A \wedge B = B \wedge A$$

$$A \vee B = B \vee A$$

အထူးအဆန်းတော့ မဟုတ်ပါဘူး။ $2 * 3 = 3 * 2$ ၊ $2 + 3 = 3 + 2$ ဆိုတာနဲ့ အတူပါပဲ။ And (\wedge) ဟာ
ငြောက်တာနဲ့ တူပြီး၊ Or (\vee) ဟာ ပေါင်းတာနဲ့ သဘောချင်းတူပါတယ်။

Associative law

$$(A \wedge B) \wedge C = A \wedge (B \wedge C)$$

$$A \vee (B \vee C) = A \vee (B \vee C)$$

ဒါကလည်း $(2 * 3) * 4 = 2 * (3 * 4)$ ၊ $(2 + 3) + 4 = 2 + (3 + 4)$ ဆိုတာနဲ့ အတူပါပဲ။

Distributed law

$$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$$

$$A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$$

ဒါကလည်း $2(3 + 5) = 2 * 3 + 2 * 5$ ဆိုတာနဲ့ အတူပါပဲ။ ဖြန့်လိုက်တဲ့ သဘောဖြစ်ပါတယ်။

$$2(3 + 5) = 2 * 8 = 16$$

$$2 * 3 + 2 * 5 = 6 + 10 = 16$$

And law

$$A \wedge 0 = 0$$

And က အားလုံးမှန်မှ မှန်တာဖြစ်တဲ့ အတွက် 0 နဲ့ and လုပ်မှတော့ တစ်ခုက 0 ဖြစ်နေတဲ့ အတွက် 0 ပဲ ထွက်မှာ ဖြစ်ပါတယ်။

$$A \wedge 1 = A$$

And က အားလုံးမှန်မှ မှန်တာဖြစ်တဲ့ အတွက် တစ်ခုက 1 ဖြစ်နေတော့ A က 0 ဆို 0 ထွက်မှာ ဖြစ်ပြီး၊ A က 1 ဆို 1 ထွက်မှာ ဖြစ်ပါတယ်။

$$A \wedge A = A$$

$A \neq A$ and လုပ်မှတော့ တစ်ခုက 0 ဆို ကျွန်တစ်ခုကလည်း 0 အဖြေကလည်း 0၊ တစ်ခုက 1 ဆို ကျွန်တစ်ခုကလည်း 1 အဖြေကလည်း 1 ဆိုပြီးဖြစ်ပါတယ်။

$$A \wedge !A = 0$$

$A \neq !A$ and $\text{လုပ်မှတော့} \rightarrow \text{တစ်ခုက } 0 \text{ ဆို } \text{ကျန်တစ်ခုက } 1 \text{ ဒီတော့ } \text{နှစ်ခုလုံး } 1 \text{ မဟုတ်လို့ } \text{အဖြေက } 0 \text{ တစ်ခုက } 1 \text{ ဆို } \text{ကျန်တစ်ခုက } 0 \text{ ဆိုတော့ } \text{အဖြေက } 0 \text{ ဆိုပြီးဖြစ်ပါတယ်။}$

မြင်သာအောင် ပြရရင်

A	0	$A \wedge 0$	1	$A \wedge 1$	$!A$	$A \wedge A$	$A \wedge !A$
0	0	0	1	0	1	0	0
1	0	0	1	1	0	1	0

Or law

$$A \vee 0 = A$$

Or က အနည်းဆုံး တစ်ခုမှန်တာနဲ့ မှန်တာဖြစ်တဲ့အတွက် တစ်ခုက 0 ဖြစ်နေတော့ A က 0 ဆို 0 ထွက်မှာဖြစ်ပြီး၊ A က 1 ဆို 1 ထွက်မှာ ဖြစ်ပါတယ်။

$$A \vee 1 = 1$$

Or က အနည်းဆုံး တစ်ခုမှန်တာနဲ့ မှန်တာဖြစ်တဲ့အတွက် 1 နဲ့ Or လုပ်မှတော့ တစ်ခုက 1 ဖြစ်နေတဲ့အတွက် 1 ပဲ ထွက်မှာ ဖြစ်ပါတယ်။

$$A \vee A = A$$

$A \neq A$ or လုပ်မှတော့ တစ်ခုက 0 ဆို $\text{ကျန်တစ်ခုကလည်း } 0$ အဖြေကလည်း 0 ၊ တစ်ခုက 1 ဆို $\text{ကျန်တစ်ခုကလည်း } 1$ အဖြေကလည်း 1 ဆိုပြီးဖြစ်ပါတယ်။

$$A \wedge !A = 1$$

$A \neq !A$ or လုပ်မှတော့ တစ်ခုက 0 ဆို $\text{ကျန်တစ်ခုက } 1$ ဒီတော့ တစ်ခု 1 ဖြစ်လို့ အဖြေက 1 ၊ တစ်ခုက 1 ဆို $\text{ကျန်တစ်ခုက } 0$ ဆိုတော့ အဖြေက 1 ဆိုပြီးဖြစ်ပါတယ်။

Operator Precedence

အလုပ်နှစ်ခု ယျဉ်ရင် ဘယ်အလုပ်ကို ဦးစားပေးလုပ်မလဲ ဆိုတာ ရှိသလို လက္ခဏာ နှစ်ခု ယျဉ်ရင် ဘယ်လက္ခဏာ(operator)ကို အရင်ဦးစားပေးလုပ်မလဲ ဆိုတဲ့ သတ်မှတ်ချက်ရှိပါတယ်။ အဲဒီလို ဦးစားပေး သတ်မှတ်တာကို *Operator Precedence* လို့ ခေါ်ပါတယ်။

Priority	Operators
1	()
2	!
3	And (\wedge)
4	Or(\vee)

အပေါ်က Table မှာပြထားတဲ့ အစဉ်လိုက်အတိုင်း အလုပ်လုပ်ပါတယ်။ ဆိုလိုတာက ()-လက်သည်းကွင်း ဟာ ဦးစားပေး နံပါတ်တစ် ဖြစ်ပါတယ်။ သူ့ပြီး ရင် not က ဒုတိယ ဦးစားပေး၊ ပြီးမှ and နဲ့ or ဖြစ်ပါတယ်။ programming language တွေ ပေါ်မှုတည်ပြီး Operator Precedence သတ်မှတ်ချက်တွေဟာ အနည်းငယ် ကွဲပြား တတ်ပါတယ်။

A \wedge B \vee C

A \wedge B က အရင် လုပ်ပြီး ရလာတဲ့ အဖြောက်မှ C နဲ့ \vee လုပ်မှာဖြစ်ပါတယ်။ and က or ထက် priority ပိုမြင့်တဲ့ အတွက်ကြောင့်ဖြစ်ပါတယ်။

A \wedge (B \vee C)

B \vee C ကို အရင်လုပ်ပြီး ရလာတဲ့ အဖြောက်မှ A နဲ့ \wedge လုပ်မှာပါ။ လက်သည်းကွင်းက priority ပိုမြင့်လို ဦးစားပေးနံပါတ်တစ် ဖြစ်တာကြောင့် ဖြစ်ပါတယ်။

!A \wedge (B \vee C)

B \vee C နဲ့ !A ကို အရင်လုပ်၊ ပြီးမှ အဲဒီကနေ ရလာတဲ့ အဖြောက်ခုကို ပြန်ပြီး and(\wedge) လုပ်မှာ ဖြစ်ပါတယ်။ priority အတိုင်း လုပ်တာ ဖြစ်ပါတယ်။

Solving Boolean algebra

And ဆိုတာ မြောက်တာနဲ့ တူတယ်၊ or ဆိုတာ ပေါင်းတာနဲ့တယ်လို့ ပြောခဲ့ပါတယ်။ not ကို တော့ bar လေးနဲ့ရေးပါတယ်။

$$\text{Q1} \quad !(A \wedge B \wedge C) \vee !(A \vee B)$$

$$= \overline{(ABC)} + \overline{(A+B)}$$

$$= \overline{(A+B+C)} + A\overline{B}$$

$$= \overline{A} + \overline{B} + \overline{AB} + \overline{C}$$

$$= \overline{A} + \overline{B} (1+A) + \overline{C}$$

$$= \overline{A} + \overline{B} \cdot 1 + \overline{C}$$

$$= \overline{A} + \overline{B} + \overline{C}$$

$$= (!A \vee !B \vee !C)$$

$$\text{Q2} \quad !(A \wedge B \vee A \wedge C) \vee !(A \wedge !B \wedge C)$$

$$= \overline{(AB+AC)} + \overline{(AB C)}$$

$$= (\overline{A} + \overline{B})(\overline{A} + \overline{C}) + (\overline{A}\overline{B}C)$$

$$= \overline{AA} + \overline{AC} + \overline{A}\overline{B} + \overline{B}\overline{C} + \overline{A}\overline{B}\overline{C}$$

$$= \overline{A} + \overline{A}\overline{C} + \overline{A}\overline{B} + \overline{B}\overline{C} + \overline{A}\overline{B}\overline{C}$$

$$= A(1 + \overline{C} + \overline{B} + \overline{B}\overline{C}) + \overline{B}\overline{C}$$

$$= \overline{A} \cdot 1 + \overline{B}\overline{C}$$

$$= \overline{A} + \overline{B}\overline{C}$$

$$= !A \vee (!B \wedge !C)$$

အခုတွက်ပြတဲ့ ပုစ္စာနှစ်ပုဒ်ကို တစ်ပုဒ်ချင်းရဲ့ မေးခွန်းနဲ့ ရှင်းထားတဲ့ အဖြန့်ခုတူညီကြောင်း Truth Table ဆဲပြီး သက်သေပြပါ။



အောက်က ပုစ္စာလေးတွေ လေ့ကျင့်ကြည့်ပါ။

1. အောက်ပါတို့အတွက် Truth Table ဆွဲပါ။
 - a. $A \wedge B \wedge C \wedge D$,
 - b. $A \vee B \vee C \vee D$,
 - c. $A \text{ xor } B \text{ xor } C \text{ xor } D$
 - d. Or Law ကို သက်သေပြုပါ။
 - e. De Morgan's Theorem ကို သက်သေပြုပါ။
 - f. Distributed Law ကို သက်သေပြုပါ။
2. အောက်ပါတို့ကို Laws တွေသုံးပြီးရှင်းပေးပါ။ ပြီးရင် ရှင်းလို့ရတဲ့ အဖြေနဲ့ မူရင်း ကိုက်ညီကြောင်း Truth Table ဆွဲပြီး သက်သေပြုပါ။
 - a. $(A \wedge B \wedge C) \vee (A \wedge B \wedge C \wedge D) \vee (A \wedge B)$
 - b. $!(A \vee B) \vee !A \vee !D \wedge (C \wedge D)$
 - c. $!(\neg A \wedge \neg B) \wedge (A \vee B \vee C)$



Chapter အနှစ်ချုပ်

- အားလုံးမှန်မှ အလုပ်လုပ်ချင်တယ်ဆိုရင် and, အနည်းဆုံး တစ်ခုမှန်တာနဲ့ အလုပ်လုပ်ချင်ရင် or, တူရင် 0 မတူရင် 1 ထွက်ချင်ရင် xor ကို သုံးရပါတယ်။
- True(1) ကို not လုပ်ရင် False(0), False(0) ကို not လုပ်ရင် True(1) ရတယ်။
- And ကို not လုပ်ရင် ori or ကို Not လုပ်ရင် and ရတယ်။

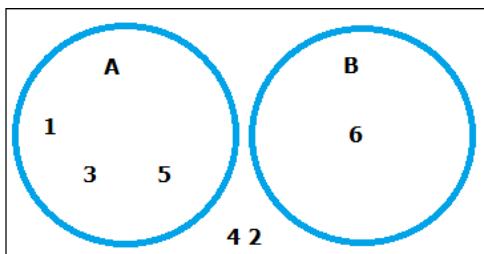
အခန်း (၄)

Statistics ဆိုင်ရာ အခြေခံများ

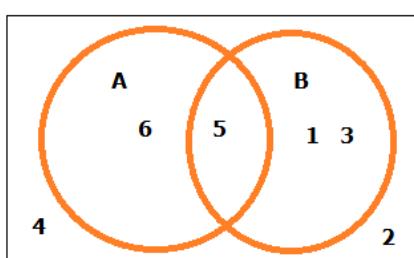
Set Theory

Disjoint and Overlapping

Disjoint ဆိုတာက တစ်ခု နှင့် တစ်ခု ဘုတူတာမပါပဲ သီးခြားစီ ဖြစ်နေတာကိုပြောတာပါ။ အောက်ကပိုတွင် Set A မှာ ပါတာက 1,3,5၊ Set B မှာ ပါတာ 6 ဒီတော့ တူတာ မရှိပါဘူး။ ဒါကြောင့် A နှင့် B ဟာ disjoint ဖြစ်ပါတယ်။

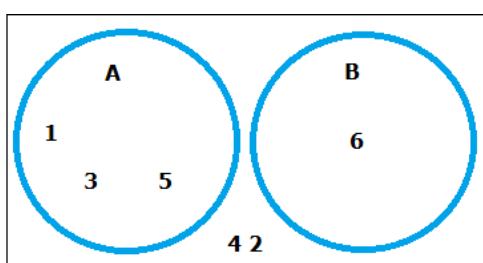


Overlapping ဆိုတာ ထပ်နေတာပါ။ ဘုတူနေတာ ပါတဲ့ အဓိပ္ပာယ်ဖြစ်ပါတယ်။ အောက်မှာ ပြထားတဲ့ပုံမှာ Set A နဲ့ Set B မှာ 5 ဆိုတာလေး ဘုပါနေတာ ဖြစ်တဲ့အတွက် Overlapping ဖြစ်နေတာပါ။



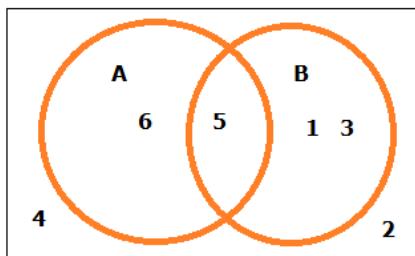
Union (U) and Interest (\cap) and Difference (-)

- Union(U) ဆိုတာ or နဲ့ တူပါတယ်။ ဘယ်ဟာမှာ ပါပါ အနည်းဆုံး တစ်ခုမှာ ပါရင်ပြီးတာပဲ ဆိုတဲ့ အဓိပ္ပာယ်ပါ။
- Interest (\cap) ဆိုတာ and နဲ့ တူပါတယ်။ and ဆိုတာ ကအားလုံးမှာ ပါရမယ် ဘုံတူတဲ့ဟာ ဆိုတဲ့ အဓိပ္ပာယ်ပါ။
- Different ဆိုတာကတော့ ခြားနားခြင်း (နှုတ်ခြင်း)၊ တူတာ ဖယ်ပစ်တယ် ဆိုတဲ့ အဓိပ္ပာယ်ပါ။



$$A = \{1, 3, 5\}, \quad B = \{6\}$$

$$A \cup B = \{1, 3, 5, 6\}, A \cap B = \{\}, A - B = \{1, 3, 5\}, B - A = \{6\}$$



$$A = \{6, 5, 4\}, B = \{1, 3, 2\}$$

$$A \cup B = \{1, 3, 4, 5, 6\}, A \cap B = \{5\}, A - B = \{6, 4\}, B - A = \{1, 3, 2\}$$

Probability

Probability Example:

	Ace	2	3	4	5	6	7	8	9	10	Jack	Queen	King
Clubs													
Diamonds													
Hearts													
Spades													

အချပ် Cards တွေဟာ probability အတွက် အကောင်းဆုံး နမူနာဖြစ်လို့ ဒီမှာလဲ အဲဒါနဲ့ပဲ နမူနာ ပြသွားပါမယ်။ မြင်သာအောင် ပုံလေးနဲ့ ပြထားပေးပါတယ်။

ပုံမှာ row လိုက်ကြည့်ကြည့်လိုက်ရင် ဖဲထူတ် တစ်ထူတ်မှာ Club 13 ချပ်၊ Diamond 13 ချပ်၊ Heart 13 ချပ် နဲ့ Spade 13 ချပ် ဆိုတော့ စုစုပေါင်း အချပ်ပေါင်း 52 ချပ်ရှိပါတယ်။

ပုံမှာ row နဲ့ column တွဲကြည့်လိုက်ရင် Club 13 ချပ်မှာမှ Ace ပါတာ 1 ချပ်၊ 2,3,4,5,6,7,8,9,10 ပါတာ 1 ချပ်စီ၊ Jack က 1 ချပ်၊ Queen က 1 ချပ် နဲ့ King က 1 ချပ် တို့ ဖြစ်ပါတယ်။ Diamond, Heart, Spade တို့မှာလဲ ထိုအတူပဲဖြစ်ပါတယ်။

ပြီးတော့ ပုံမှာ ကြည့်လိုက်ရင် သိသာတာက Club နှင့် spades ဟာ အနက်ရောင်တွေဖြစ်ပြီး၊ Diamonds နဲ့ Hearts ကတော့ အနီရောင်တွေ ဖြစ်ပါတယ်။

Probability Formulas

Probability ဆိုတာ ဘယ်လောက် ဖြစ်နိုင်ခြရိတယ်ဆိုတာကို တွက်ချက် တာဖြစ်ပါတယ်။ ကိုယ်လိုချင်တဲ့ အရာရဲ့ ဖြစ်နိုင်တဲ့ အကြိမ်အရေအတွက်ကို တည်ပြီးတော့ စုစုပေါင်း ဖြစ်နိုင်ချေနဲ့ စားရတာပါ။

$$P(\text{event}) = \frac{\text{Number of occurrences of the event}}{\text{Total Number of trials or outcomes}}$$

Probability ဆိုတဲ့ ဖြစ်နိုင်ခြေဟာ 100% ထက်ပိုပြီး မဖြစ်နိုင်ပါဘူး။ 100% ကို ကိန်းကဏ္ဍးလိုပြောရင်တော့ 1 ဖြစ်ပါတယ်။ ဒါကြောင့်

$$P(A) + P(\bar{A}) = 1$$

ဆိုလိုတာက A ရဲ့ probability နဲ့ A မဟုတ်တဲ့ probability ပေါင်းရင် ၁ ရပါတယ်။

$$P(\bar{A}) = 1 - P(A)$$

ဒါကြောင့် $P(\text{not } A)$ ကို (not A ဆိုတာ အလွယ်ပြောတာပါ၊ complement of A လိုခံဗျာပါတယ်)

လိုချင်ရင် 1 ထဲက $P(A)$ နှတ်တာနဲ့ အတူတူပါပဲ။



1, 2, 3, 4, 5, 6 ဆိုပြီး 6 မျက်နှာရှိတဲ့ အန်စာတုံးတစ်တုံးကို ပစ်လိုက်လို့ 2 ဂဏ်းကျနိုင်တဲ့ probability နှင့် J ဂဏ်းမဟုတ်သော အခြားဂဏ်းများကျတဲ့ probability ကို တွက်ချက် ပေးပါ။

Number of occurrences of the 2 = 1 (2 ဂဏ်းသည် တစ်မျက်နှာမှာပဲ ပါလို့)

Total number of trials or outcomes = 6 (စုစုပေါင်း 6 မျက်နာ)

$$P(2) = 1/6, \quad P(\text{not } 2) \text{ or } P(1 \text{ or } 3 \text{ or } 4 \text{ or } 5 \text{ or } 6) = 1 - 1/6 = 5/6$$

ရာခိုင်နှစ်းနဲ့ ပြောက်လိုက်တော့ 100 နဲ့ မြောက်ပေးရပါတယ်။

အံစာတူနှစ်းကို မြောက်လိုက်လို့ ၂ ကျိုင်တဲ့ ရာခိုင်နှစ်း $1/6 * 100 = 16.67\%$ ဖြစ်ပါတယ်။

အံစာတူနှစ်းကို မြောက်လိုက်လို့ ၂ မဟုတ်သော ၁ သို့မဟုတ် ၃ သို့မဟုတ် ၄ သို့မဟုတ် ၆ ကျိုင်တဲ့ ရာခိုင်နှစ်း $5/6 * 100 = 83.33\%$ ဖြစ်ပါတယ်။

$16.67\% + 83.33\% = 100\%$ ဖြစ်ပါတယ်။ ရာခိုင်နှစ်းအားလုံးပေါင်းရင် 100 ပါပဲ။ 100 ထက်မကျော်ပါဘူး။



P ('2' of spades) = ?

No of '2' of spades = 1 (အပေါ်ကပံကိုသွားကြည့်ပါ spades row ထဲမှာ 2 ဖြစ်တာ 1 ခုပဲရှိပါတယ်)

Total Number of cards = 52 (စုစုပေါင်း 52 ကော်)

P ('2' of spades) = 1/52



P (a king of red color) = ?

No of king of red color = 2 (စုစုပေါင်း King 4 ချပ်မှာ အနီရောင်ကတော့ 2 ချပ်ပဲ ရှိပါတယ်။)

P(a king of red color) = 2/52

တကယ်လို့ P(not king of red color) တွက်ပေးပါဆိုရင်

P(not king of red color) = 1 - 2/52 = 50/52



$P(\text{a non-face card}) = ?$

Total number of face card out of 52 cards = 12 (Jack 4 + Queen 4+ King 4)

$P(\text{face card}) = 12/52$

$$P(\text{a non-face card}) = 1 - \frac{12}{52} = \frac{52 - 12}{52} = \frac{40}{52} = \frac{10}{13}$$



Or logic in Probability

For Mutually Exclusive Events (disjoint)

$P(\text{A or B}) \text{ or } P(A \cup B) = P(A) + P(B)$

For Non-Mutually Exclusive Events (Overlap)

$P(\text{A or B}) \text{ or } P(A \cup B) = P(A) + P(B) - P(\text{A and B})$



$P(\text{King or Queen})$ ကို တွက်ပေးပါ။

(မမှတ်မိရင် အပေါ်မှာ ပြောခဲ့တဲ့ ပုံကိုပြန်ကြည့်ပါ။ King ထဲမှာ Queen ပါနေတာ၊ Queen ထဲမှာ King ပါနေတာမရှိတဲ့အတွက် disjoint ဖြစ်တဲ့ Mutually Exclusive Events ပုံသေနည်းကို သုံးပြီး တွက်ရပါတယ်။)

Let Event (A) = King, Event (B) = Queen

$$P(A \cup B) = P(A) + P(B) = 4/52 + 4/52 = 8/52 = 2/13$$



P(spade or jack) ကို တွက်ပါမယ်။

(spades 13 ချုပ်ထဲမှာ jack 1 ချုပ်ပါနေလို့ Non-Mutually Exclusive Events(Overlap)
ပုံသေနည်းနဲ့ တွက်ပါမယ်။)

$$P(\text{spades}) = 13/52$$

$$P(\text{jack}) = 4/52$$

$$P(\text{spade and jack}) = 1/52$$

$$P(\text{spade or jack}) = P(A) + P(B) - P(A \text{ and } B)$$

$$= 13/52 + 4/52 - 1/52 = 16/52 = 4/13$$

$$P(\text{neither a spade nor a jack}) = 1 - 4/13 = 9/13$$



And Logic in Probability

For Independent Events

$$P(A \text{ and } B) = P(A \cap B) = P(A) \times P(B)$$

For Dependent Events (Conditional Probability)

$$P(A|B) = \frac{P(A \cap B)}{P(B)}, P(A \cap B) = P(A|B) * P(B)$$

$$P(B|A) = \frac{P(A \cap B)}{P(A)}, P(A \cap B) = P(B|A) * P(A)$$

-  52 ချပ်ထဲ က တစ်ချပ်ပြီးတစ်ချပ် စုစုပေါင်းနှစ်ချပ်ကို ဆွဲယူပါမယ်။ ပထမတစ်ကားကို ယူပြီး
ပြန်မထည့်ပါဘူး။ ပထမတစ်ကားက Club ဖြစ်ပြီး၊ ဒုတိယ တစ်ကားက spade ဖြစ်တဲ့ probability ကို
တွက်ပါ။

Let Event (A) = the first card is Club,

Event (B) = the second card is Spade

$$P(A) = 13/52$$

$P(B/A) = 13/51$ (မမှတ်မရင် အပေါ်မှာ ပြောခဲ့တဲ့ ပုံကိုပြန်ကြည့်ပါ။ ပြန်မထည့်ဘူး
လိုပြောတဲ့အတွက် dependent formula နဲ့ တွက်ပါမယ်။ တက်သူ့ပြီးသားဖြစ်လို့ 51 ကုန်ပဲ
ကျန်ပါမယ်။ အဲဒါ 51 ကုန်ထဲမှာ spade က 13 ချပ် ရှိပါတယ်။)

$$P(A \cap B) = P(B/A) * P(B)$$

$$= \frac{13}{52} * \frac{13}{51} = \frac{169}{2652}$$

-  52 ချပ်ထဲ က တစ်ချပ်ပြီးတစ်ချပ် စုစုပေါင်းနှစ်ချပ်ကို ဆွဲယူပါမယ်။ ပထမတစ်ကားကို ယူပြီး
ပြန်ထည့်ပါမယ်။ ပထမတစ်ကားက Club ဖြစ်ပြီး၊ ဒုတိယ တစ်ကားက spade ဖြစ်တဲ့ probability ကို
တွက်ပါ။

Let Event (A) = the first card is Club, Event (B) = the second card is spade

$$P(A) = 13/52$$

$P(B) = 13/52$ (ပြန်ထည့်မယ်ပြောတဲ့အတွက် ပထမဆွဲယူလိုက်လို့ ဒုတိယတစ်ခုမှာ
လျော့တာမျိုး သက်ရောက်မှုမရှိလို့ independent formula နဲ့ တွက်ပါမယ်။)

$$P(A \cap B) = P(A) * P(B)$$

$$= \frac{13}{52} * \frac{13}{52} = \frac{1}{4} * \frac{1}{4} = \frac{1}{16}$$

Mean

Mean ဆိတာ အလွယ်ပြောမယ် ဆိုရင်တော့ average ပါပဲ။ စုစု ပေါင်းကို တည်ပြီး အရေအတွက်နဲ့ စားတာပေါ့။

$$\mu = \Sigma X / N$$

ဤ ဗုံမှာ 10, 20, 30 တို့ရဲ့ mean ကိုတွက်မယ်ဆိုရင်

$$\Sigma X = 10 + 20 + 30 = 60, N = 3$$

$$\mu = 60/3 = 20$$

Medium

Medium ဆိတာ ငယ်စဉ်ကြီးလိုက် ကိန်းစဉ်တန်းထဲက အလယ်ကိန်းကို ရှာတာ ဖြစ်ပါတယ်။ ကိန်းအလုံး အရေအတွက်က မ ဖြစ်တယ်ဆိုရင် အလည်တည့်တည့်က ကိန်းသည် medium ဖြစ်ပြီး၊ ကိန်းအလုံး အရေအတွက်က စုံ ဖြစ်တယ်ဆိုရင်တော့ အလည်တည့်တည့်က နှစ်လုံးကိုပေါင်းပြီး 2 ဖြင့် စားပေးရမှာ ဖြစ်ပါတယ်။

100, 100, **130**, 140, 150

$$\text{Medium} = 130$$

100, 100, **120, 130**, 140, 150

Medium = $(120+130)/2 = 125$

Standard deviation (σ)

Standard deviation (σ) ဆိတာ ပျမ်းမျသွဖယ်မှာ ကွာခြားမှုကို တွက်ချက်တာပါ။ standard deviation (σ) တန်ဖိုးများလေလေ ကိန်းစဉ်တန်းဟာ range ကျယ်ပြန်လေလေ ကွာခြားချက်များလေလေ၊ standard deviation (σ) တန်ဖိုးနည်းလေလေ range ကျဉ်းလေလေ၊ ကွာခြားချက်နည်းလေလေ လိုပြောလို ရပါတယ်။

$$\sigma = \sqrt{\sigma^2}$$

$$variance = \sigma^2 = \Sigma \frac{(xi - \mu)^2}{N}$$

ဒါကြောင့် standard deviation ကို တွက်ချင်ရင် variance ကို အရင်တွက်ရမှာ ဖြစ်ပါတယ်။



1, 3, 5, 7

$$\Sigma X = 1 + 3 + 5 + 7 = 16, N = 4$$

$$\mu = 16/4 = 4$$

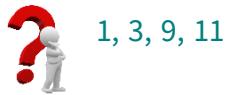
$$\sigma^2 = \Sigma \frac{(xi - \mu)^2}{N}$$

$$\sigma^2 = \frac{(1-4)^2 + (3-4)^2 + (5-4)^2 + (7-4)^2}{4}$$

$$\sigma^2 = \frac{(-3)^2 + (-1)^2 + (1)^2 + (3)^2}{4}$$

$$\sigma^2 = \frac{9 + 1 + 1 + 9}{4} = \frac{20}{4} = 5$$

$$\sigma = \sqrt{5} = 2.2$$



1, 3, 9, 11

$$\Sigma X = 1 + 3 + 9 + 11 = 24, N = 4$$

$$\mu = 24/4 = 6$$

$$\sigma^2 = \Sigma \frac{(xi - \mu)^2}{N}$$

$$\sigma^2 = \frac{(1-6)^2 + (3-6)^2 + (9-6)^2 + (11-6)^2}{4}$$

$$\sigma^2 = \frac{(-5)^2 + (-3)^2 + (3)^2 + (5)^2}{4}$$

$$\sigma^2 = \frac{25 + 9 + 9 + 25}{4} = \frac{68}{4} = 17$$

$$\sigma = \sqrt{17} = 4.1$$

Range

Range ဆုတေသန ကိန်းစည်တန်းထဲက အကြီးဆုံးကိန်းနှင့် အငယ်ဆုံးတို့၏ ခြားနားခြင်းဖြစ်ပါတယ်။

1, 3, 4, 5, 5, 6, 7, 11

$$\text{Range} = 11 - 1 = 10$$

The Interquartile Range (IQR)

The interquartile range (IQR) is ဆိုတာ ကိန်းစဉ်တန်းကို အညီအမျှ အပိုင်း င့် ပိုင်းချင်တဲ့ အခါမှာ သုံးပါတယ်။ ကိန်းစဉ်တန်းကို နှစ်ပိုင်းပိုင်းပါ၊ ပထမတစ်ပိုင်းက Q1 တွက်ရင် သုံးမှာ ဖြစ်ပြီး၊ ဒုတိယ တစ်ပိုင်းက Q3 တွက်ရင် သုံးမှာ ဖြစ်ပါတယ်။

- Q1 စီထားတဲ့ ကိန်းစဉ်တန်းရဲ့ ပထမတစ်ပိုင်းရဲ့ အလယ်ကိန်းဖြစ်ပါတယ်။
- Q2 ကတော့ medium ဖြစ်ပါတယ်။
- Q3 စီထားတဲ့ ကိန်းစဉ်တန်းရဲ့ ဒုတိယတစ်ပိုင်းရဲ့ အလယ်ကိန်းဖြစ်ပါတယ်။

1, 3, 5, **7, 9**, 11, 13, 15

ပထမတစ်ဝက် = 1, 3, 5, 7 (စုံအလုံးအရေအတွက်)

$$Q1 = (3+5)/2 = 4$$

$$Q2 = (7+9)/2 = 8$$

ဒုတိယတစ်ဝက် = 9, 11, 13, 15 (စုံအလုံးအရေအတွက်)

$$Q3 = (11+13)/2 = 12$$

Q1 က 4 ရလို့ ပထမအပိုင်း => 1,3

Q1 က 4 , Q2 က 8 ရလို့ ဒုတိယအပိုင်း => 5,7

Q2 က 8, Q3 က 12 ရလို့ တတိယအပိုင်း => 9,11 နှင့် စတုထွေအပိုင်း => 13,15 ဆိုပြီး အညီအမျှ 4 ပိုင်း ပိုင်းပေးသွားတာ ဖြစ်ပါတယ်။



အောက်က ပုံစံလေးတွေ လေ့ကျင့်ကြည့်ပါ။

1. $A = \{2, 4, 6, 8, 10, 12, 14, 15, 16, 18, 20\}$, $B = \{0, 5, 10, 15, 20\}$. Find $A \cup B$, $A \cap B$, $A - B$ and $B - A$.
2. P (a black face card) ကို တွက်ပါ။
3. $P(\text{Clubs or Hearts or Spade})$ ကို $P(A) + P(\bar{A}) = 1$ ဆိုတဲ့ ပုံစံနည်း သုံးပြီးတွက်ပေးပါ။
4. အကြွောင်း တစ်ခွေမှာ ခေါင်းနဲ့ ပန်းဆိုပြီး နှစ်ဖက်ရှိပါတယ်။ ခေါင်းကျနိုင်တဲ့ probability နဲ့ ပန်းကျနိုင်တဲ့ probability ကိုရှာပါ။
5. အတန်းထဲမှာ မိန်းကလေး 10 ယောက် ယောက်ဦးလေး 5 ယောက်ရှိပါတယ်။
အတန်းခေါင်းဆောင်အဖြစ် မိန်းကလေးထဲက တစ်ယောက်နဲ့ ယောက်ဦးလေး တစ်ယောက်ကို ရွှေးချယ်မှာဖြစ်ပါတယ်။
 - a. အေးအေး အတန်းခေါင်းဆောင် ဖြစ်နိုင်တဲ့ probability
 - b. မောင်မောင် အတန်းခေါင်းဆောင် ဖြစ်နိုင်တဲ့ probability
 - c. အေးအေး သို့မဟုတ် မောင်မောင် အတန်းခေါင်းဆောင်ဖြစ်နိုင်တဲ့ probability
 - d. အေးအေးရော၊ မောင်မောင်ပါ အတန်းခေါင်းဆောင် ဖြစ်နိုင်တဲ့ probability တို့ကို ရှာပေးပါ။
6. 52 ချပ်ထဲ က တစ်ချပ်ပြီးတစ်ချပ် စုစုပေါင်းနှစ်ချပ်ကို ဆွဲယူပါမယ်။ ပထမတစ်ကားကို ယူပြီး ပြန်မထည့်ပါဘူး။ ပထမတစ်ကားက Red Face ဖြစ်ပြီး၊ ဒုတိယ တစ်ကားက Black Face ဖြစ်တဲ့ probability ကို တွက်ပါ။
7. 52 ချပ်ထဲ က တစ်ချပ်ပြီးတစ်ချပ် စုစုပေါင်းနှစ်ချပ်ကို ဆွဲယူပါမယ်။ ပထမတစ်ကားကို ယူပြီး ပြန်ထည့်ပါမယ်။ ပထမတစ်ကားက Club ဖြစ်ပြီး၊ ဒုတိယ တစ်ကားက Queen ဖြစ်တဲ့ probability ကို တွက်ပါ။

8. 37 ကနေ 123 ထိ ကိန်းစဉ်တန်းရဲ့ mean ကို ရှာပေးပါ။ စုစုပေါင်းကိုတည်ပြီးတော့ အရေအတွက်နဲ့ စားမယ်ဆိုပြီး စုစုပေါင်းမနေပါနဲ့၊ ထိုင်ပြီး ဘယ်နှစ်လုံးလဲ ရေတွက် မနေပါနဲ့နော်။ ရှုံးသင်ခန်းစာက summation Formula တွေကို အသုံးချပါ။
9. 1, 3, 5, 7, 9, 11, 13, 15 ရဲ့ Medium ကိုရှာပေးပါ။
10. 2, 4, 6, 8, 10, 12, 14, 16, 18, 10 ရဲ့ Medium နှင့် Standard deviation ကို ရှာပေးပါ။
11. 1, 5, 30, 45, 60, 770, 888, 990 ကိန်းစဉ်တန်းရဲ့ Range ကိုရှာပါ။ ပြီးလျှင် IQR ကိုသုံးပြီး 4 ပိုင်းပိုင်းပေးပါ။

Chapter အနှစ်ချုပ်

ဆရာဝန်တစ်ယောက် ဟာ လူနာတွေကို ဆေးကုတ္တုမှတ်တမ်းတွေကို သိမ်းဆည်းထားလေ့ရှိတယ်။ လူနာတွေ အရမ်းများတဲ့အတွက် အချက်အလက်တွေကလည်း များမှာ အမှန်ပါပဲ။ အဲဒါတွေကို လာပြရင် ကျွန်ုတ်မတို့လို သာမာန်လူ(ဆေးဝန်ထမ်း မဟုတ်တဲ့သူတွေ) က နားမလည်ပါဘူး။ ဆေးဝန်ထမ်းတွေတောင် တစ်ခုချင်းနားလည်မှာဖြစ်သော်လည်း data များရင် ကြည့်ရတာ မူးပြီး၊ ကောက်ချက်ဆွဲဖို့ ခက်ခဲမှာ အသေအချာပါပဲ၊ အဲဒီလို များပြားလှတဲ့ နားလည်ဖို့ ခက်ခဲတဲ့ အကြမ်းအချက်အလက်တွေကို data လို့ခေါ်ပါတယ်။ အဲဒီ အချက်အလက်တွေပေါ်မှာ တွက်ချက်မှုတစ်ခုခု လုပ်ပြီး ဘာရောဂါလက္ခဏာတွေ ရှိရင် ဘာရောဂါလို ထုတ်ပြရင်တော့ ကျွန်ုတ်နားလည်ပါတယ်။ အဲဒီလို အများနားလည်လွယ်တဲ့ အသုံးဝင်တဲ့ အချက်အလက်တွေကို information လို့ခေါ်ပါတယ်။

ပြန်ပြောရရင် နားလည်ဖို့ရာ မလွယ်ကူတဲ့ များပြားလှတဲ့ အကြမ်း အချက်အလက်တွေကို data ခေါ်ပြီး၊ အဲဒီ data ပေါ်မှာ တွက်ချက်မှုလုပ်ပြီး အများနားလည်တဲ့ အသုံးဝင်တဲ့ ပုံစံနဲ့ ထုတ်ပြပေးတာကို information လို့ခေါ်ပါတယ်။

ကျောင်းတွေမှာ ကလေးတွေကို နောက်အတန်းကိုတက်ဖို့ ယခုလက်ရှိအတန်းအတွက် စာမေးပွဲ ဖြေရပါတယ်။ အဲဒီစာမေးပွဲမှာ လုံလောက်သော အမှတ်ကိုရမှုသာ နောက်အတန်းကို တက်ခွင့်ရမှာ ဖြစ်ပါတယ်။ လုံလောက်သော အမှတ်ဆိတာ၊ လိုလောက်သော မှန်ကန်မှု(accuracy) ပဲဖြစ်ပါတယ်။ ထိုအတူပဲ ထွက်လာတဲ့ information ကို တန်းပြီး ပေးသုံးလို့မရပါဘူး၊ အဲဒီ information ဟာ ဘယ်လောက်မှန်ကန်မှုရှိသလဲဆိုတဲ့ accuracy ကို တွက်ချက်ပြီး accuracy ကောင်းမှ information ဟာ စိတ်ချွော အသုံးပြုလို့ ရမှာ ဖြစ်ပါတယ်။

Statistics ဟာ data တွေကနေ information ဆဲထုတ်ဖို့ တွက်ချက်တဲ့ နေရာတွေ နှင့် ဆဲထုတ်လိုက်တဲ့ information ဟာ ဘယ်လောက် မှန်ကန်မှုရှိသလဲဆိုတဲ့ accuracy တွက်ချက်တဲ့ နေရာတွေမှာ အလွန်အသုံးပြုပါတယ်။

နောက်ပြီး programmer တစ်ယောက်အနေနဲ့ သချိုာတို့ statistic တို့ဆိတာ မသိမဖြစ်လိုအပ်ချက်တွေ ဖြစ်ပါတယ်။ ဒါကြောင့် note စာအုပ်လေး တစ်အုပ်ထားပြီး လိုက်မှတ်ပါ။ ချွောက်ပါ။ ဖတ်ရုံနဲ့တော့ အော်အင်း ဆိုပြီး ပြီးသွားမှာပါ။ ဘာမှ မှတ်မိမှာ မဟုတ်ပါဘူး။

အခန်း (၅)

Data Structure မိတ်ဆက်

Data Structure ဆိတာဘာလဲ။

လူတွေ တစ်ယောက်နဲ့တစ်ယောက် ပြောဆိုဆက်ဆံတဲ့အခါမှာ ပဲဖြစ်ဖြစ် စာရေးသားပြီး ဆက်သွယ်တဲ့အခါမှာပဲဖြစ်ဖြစ် မြန်မာလူမျိုးအချင်းချင်းက မြန်မာလို ရေးသားပြောဆိုဆက်ဆံမယ်။ ထိုင်းလူမျိုးအချင်းချင်းက ထိုင်းလို ရေးသားပြောဆို ဆက်ဆံမယ်။ အဲဒီလို အချင်းချင်း ပြောဆိုဆက်ဆံတဲ့ မြန်မာလို ထိုင်းလို စတာတွေကို language(ဘာသာစကား) လိုခေါ်ပါတယ်။ ဒီလိုပါပဲ program(software/code) တွေကိုရေးသားတဲ့ programmer တွေ အသုံးပြုတဲ့ language ကို ကြတော့ programming language လိုခေါ်ပါတယ်။ လူတွေပြောဆိုဆက်ဆံတဲ့အခါမှာသုံးတဲ့ language မှာ မြန်မာ၊ ထိုင်း စသည်ဖြင့် များစွာရှိသလို programming language တွေလဲများစွာရှိပါတယ်။ ဥပမာအနေနဲ့ ပြောရရင် Python, Java, PHP, C#.net တို့ပေါ့။

Data Structure ဆိတာကတော့ programming language မဟုတ်ပါ။ Programming Language တွေကိုသုံးပြီး program တွေရေးတဲ့အခါမှာ

1. Dataတွေကို ဘယ်လိုပုံစံတွေ စုဆည်း သိမ်းဆည်း လိုရသလဲ၊ အဲဒီထဲကမှ ဘယ်လိုအခြေအနေမှာ ဘယ်လိုပုံစံနဲ့ သိမ်းဆည်းရင်ကောင်းမလဲ၊
2. အဲဒီ data တွေကနေ ကိုယ်လိုချင်တဲ့အဖြေကိုရဖို့ ဘယ်လို algorithm တွေနဲ့ အလုပ်လုပ်လို (တွက်ချက်လို) ရမလဲ

စတဲ့ အချက် J ချက်ကို အမိက သိအောင် လေ့လာတာကို data structure လို ခေါ်ပါတယ်။

၁. Data သိမ်ဆည်းပုံ ကို လွှေလာခြင်း

Data တွေကို သိမ်းတဲ့အခါ လိုအပ်ချက်အရ တန်ဖိုးတစ်ခုထဲသိမ်းချင်တာမျိုးရှိသလို၊ တန်ဖိုးတွေအတွဲလိုက်သိမ်းချင်တာမျိုးလည်းရှိပါတယ်။

ဥပမာ လူတစ်ယောက်ရဲ့ လက်ရှိအသက်အရွယ်ကိုသိမ်းချင်တယ်ဆိုရင် လူတစ်ယောက်မှာ လက်ရှိ အသက်ရွယ်ဆိုတာ တစ်ခုတည်းပဲရှိပါတယ်။ ဒီလို တစ်ခုထဲရှိတဲ့ တန်ဖိုးသိမ်းချင်တယ်ဆိုရင်တော့ primitive data types တွေထဲကတစ်ခု ကို အသုံးပြုပါတယ်။ Primitive data type ဆိုတဲ့ ခေါင်းစဉ်အောက်မှာလဲ int, float စသည်ဖြင့် အခွဲလေးတွေ အများကြီးရှိပါသေးတယ်။

မွေးချင်းမောင်နှစ်မတွေကိုသိမ်းချင်တယ် ဆိုရင်တော့ တစ်ချို့က မောင်နှစ်မ မရှိဘူး၊ တစ်ချို့က အများကြီးရှိတယ်၊ ဒီတော့ အားလုံးအဆင်ပြအောင် မွေးချင်းမောင်နှစ်မဆိုရင် တန်ဖိုးအတွဲလိုက် အများကြီး သိမ်းတဲ့ ပုံစံမျိုးနဲ့ သိမ်းရပါလိမ့်မယ်။ ဒီလို အတွဲလိုက်အများကြီး သိမ်းချင်ရင်တော့ array, stack, queue, lists စသည်တို့ထဲက တစ်ခုကို ရွေးချယ် အသုံးပြုလိုရပါတယ်။ array, stack, queue, lists တို့အပြင် အတွဲလိုက်သိမ်းလို့ရတဲ့ဟာတွေ နောက်ထပ် အများကြီးရှိပါသေးတယ်။

ဒါဆိုရင် primitive data types ဆိုတာ တန်ဖိုးတစ်ခုတည်းသိမ်းတဲ့နေရာမှာသုံးပြီး array, stack, queue, lists တို့ကတော့ တန်ဖိုးအများကြီးကို အတွဲလိုက်သိမ်းချင်တဲ့အခါမှာ သုံးတယ်ဆိုတာကို သဘောပေါက်လောက်ပါပြီ။

Primitive data type ပဲဖြစ်ဖြစ် array, stack, queue, lists တို့ပဲဖြစ်ဖြစ် တစ်ခုချင်းစီမှာ အားနည်းချက်၊ အားသာချက်တွေရှိပါတယ်။ အားနည်းချက်၊ အားသာချက်ပေါ်မူတည်ပြီး ဘယ်လို အခြေနော၊ ဘယ်လိုလိုအပ်ချက်မျိုးမှာ ဘယ်ဟာကို အသုံးပြုသင့်တယ် စသည်ဖြင့် လွှေလာတာဖြစ်ပါတယ်။ တစ်ခုချင်းစီရဲ့ အားနည်းချက် အားသာချက်တွေကို သိထားခြင်းအားဖြင့်

ကိုယ်ရေးနေတဲ့ code ရဲ့ လိုအပ်ချက် အခြေနေပါ်မူတည်ပြီး သုံးစွဲတတ်သွားရင် code ရေးတဲ့အခါမှာ အရမ်းသက်သာ လွယ်ကူမှာဖြစ်ပါတယ်။

J. Algorithm ဆိတာဘာလဲ။

အလုပ်တစ်ခုပြီးမြောက်ဖို့ တစ်ဆင့်ပြီး တစ်ဆင့်အလုပ်လုပ်ပုံ တနည်းအားဖြင့် ဘာပြီးရင် ဘာလုပ်ရမလဲဆိုတဲ့ လုပ်ရမည့်အဆင့်ဆင့် ကို Algorithm လိုခေါ်ပါတယ်။

ဥပမာ ထမင်းချက်မယ် ဆိုတဲ့ algorithm ဆိုရင်



၁ -စားသုံးမည့်သူအရောင်က် နှင့် တစ်ဦးချင်းစားသုံးနိုင်တဲ့ ပမာဏပါ်မူတည်ပြီး ချက်ရမည့် ဆန်ပမာဏ တွက်ချက်ပါ။

J - တွက်ချက်ထားသော ပမာဏအတိုင်း ဆန်ကိုယူပါ

၃ - ယူလာသောဆန်ကို ရေဖြင့် နှစ်ထပ်သုံးထပ် ဆေးပါ

၄-မပျော့၊ မမာ အနေတော် ထမင်းရအောင် ဆေးပြီးသားဆန်နှင့် သင့်တော်တဲ့ရေပမာဏကို ထမင်းပေါင်းအိုးထဲထည့်ပါ။ (ရေများလွန်းလျှင်ပျော့မည်၊ ရေးနည်းသွားလျှင်မာမည်။)

၅ - ထမင်းပေါင်းအိုးအဖုံးကိုပိတ်ပြီး မီးကြိုးကို ပလပ်ပေါက်မှာတပ်ပါ။

၆ - မီးခလုတ်ဖွင့်ပါ။

၇- ကျက်လို့ ထမင်းပေါင်းအိုးက အချက်ပြရင် ပလပ်ပေါက်ဖြုတ်ပါ၊ မီးခလုတ်ပိတ်ပါ။

၈- ထမင်း သုံးဆောင်လို့ရပါပြီ

ဒါဆိုရင် ပေါင်းအိုးနဲ့ထမင်းချက်နည်း အဆင့်ဆင့်ပြထားတာမို့ ပေါင်းအိုးနဲ့ ထမင်းချက်တဲ့ algorithm လို့ ပြောလို့ရပါတယ်။ (ထမင်းချက်တာ မမြင်ဖူးလို့ သိချင်ရင် youtube မှာ video ရှာဖွေလို့ဖြစ်ပါတယ်)

ဒီမှာ တစ်ခုထပ်ပြောချင်တာက algorithm ဆိုတာက step by step procedure (အလုပ်လုပ်ပုံ တစ်ဆင့်ချင်း ကိုဖော်ပြခြင်း) လို့ပြောပါတယ်။ တစ်ဆင့်ချင်း အစီစဉ်လိုက် လုပ်ရပါတယ်။ ကျော်ချလို့ မရပါဘူး။

ကျောင်းတက်တူနဲ့က သူ့ငယ်ချင်းနှစ်ယောက်နဲ့ အဆောင်မှာ အတူနေပါတယ်။ တစ်ရက်မှာ သိပ်ပြီးနေမကောင်းတာနဲ့ စောစောအိပ်မယ်ဆိုပြီး သုံးယောက်စာ ဆန်ဆေး၊ ပေါင်းအိုးထဲကိုအဆင်သင့် ထည့်ပေးခဲ့တယ်။ ပြီးတော့ သူ့ငယ်ချင်းကို

“ဆန်ဆေးပြီး ပေါင်းအိုးထဲ အဆင်သင့်ထည့်ခဲ့တယ်။ နောက်ကျမှ ချက်လေ မနက် ပူဗူစားရလေမို့ နှင်အိမ်တော့မယ်ဆိုမှ ပလပ်ပေါက်ထိုးလိုက်နော်”လို့မှာခဲ့ပါတယ်။

မနက်ထမင်းစားမယ်လုပ်တော့မှ ပေါင်းအိုးထဲမှာ ထမင်းအစား မချက်ရသေးတဲ့ ဆန်အစွဲလိုက်တွေ တွေ့ရပါတယ်။ အဲဒီတော့မှ သူ့ငယ်ချင်းက ပလပ်ပေါက်တော့ထိုးတယ်

မီးခလုပ်ဖွင့်ဖို့ မေ့နေတာဟာ တဲ့လေ။ ကဲ မီးခလုပ်ဖွင့်ဆိုတဲ့ အဆင်တစ်ဆင့် မေ့ကျွန်းတာနဲ့ စားလို့ရတဲ့ထေမင်းဆိုတဲ့ လို့ချင်တဲ့ result မရတော့ပါဘူး။

ဒါကြောင့် မှတ်ထားရမှာက algorithm ဆိုတာက step by step procedure လို့ပြောပါတယ်။
တစ်ဆင့်ချင်း အစီစဉ်လိုက် လုပ်ရပါတယ်။ ကျော်ချလို့မရပါဘူး။

ဒီမှာ input, output ဆိုတာကို နည်းနည်း ရှင်းချင်ပါသေးတယ်။

Input - ဆိုတာ လိုအပ်ချက်လို့ပြောလို့ရမယ်ထင်ပါတယ်။

Output - ဆိုတာ လို့ချင်တဲ့ result ပါ။

ဒီ algorithm မှာ ဆိုရင် စာသုံးမည့်လူအရေတွက်၊ တစ်ယောက်ချင်းစားနှင့်တဲ့ ပမာဏ ဒီနှစ်ခုဟာ သိဖို့လိုတဲ့ လိုအပ်ချက်ဖြစ်ပြီး၊ ဆန်၊ ရော၊ ထမင်းပေါင်းအိုး၊ လျှပ်စစ်မီး ကတော့ ရှိဖို့လိုတဲ့ လိုအပ်ချက်ဖြစ်ပါတယ်။ ရှိဖို့လိုတဲ့ လိုအပ်ချက်ရှိနေမှ အဲဒီ algorithm ကို သုံးသင့်ပါတယ်။ ဥပမာ လျှပ်စစ်မီး မရရှိသေးတဲ့ နေရာမျိုးမှာ ပေါင်းအိုးနဲ့ ထမင်းချက်ဖို့ဆိုတာ မဖြစ်နိုင်ပါဘူး။ သိဖို့လိုတဲ့ လိုအပ်ချက်ကိုတော့ input လို့ခေါ်ပါတယ်။ စားလို့ရသော ထမင်း ဆိုတာကတော့ လို့ချင်တဲ့ result နဲ့ output လို့ခေါ်ပါတယ်။

မီးသွေးနဲ့ ထမင်းချက်နည်း algorithm ရေးပါဆိုရင်

၁ - မီးမွေးထားပါ

၂ - စားမည့်လူအရေတွက်နှင့် တစ်ယောက်ချင်းစားသုံးနှင့်တဲ့ ပမာဏပေါ်မှုတည်ပြီး ချက်ရမည့် ပမာဏကို တွက်ချက်ပါ။

၃ - တွက်ချက်ထားသော ပမာဏအတိုင်း ဆန်ကိုယူပါ

၄ - ယူလာသောဆန်ကို ရေဖြင့် နှစ်ထပ်သုံးထပ် ဆေးပါ

၅ - ချက်မည့်အိုးထဲသို့ ဆေးပြီးသားဆန်ထည့်ပါ၊ မီးဖိုပေါ်အိုးကိုတင်ထားလိုက်ပါ။

(ရေးဆွဲးချက်မှာ မဟုတ်ပဲ ဆန်စွဲတွေ မပျော်မမာ အနေအထားရောက်ရင် ထမင်းရောတွေငဲ့ပစ်မှာမူး ရေကိုအရမ်းကြီး တွက်ချက်ထည့်နေစရာမလိုပါဘူး၊ ကြိုက်သလောက် ထည့်လို့ရပါတယ်)

၆ - ရေများပွဲက်ပွဲက်ဆူလာပြီ ဆိုရင် အထဲက ဆန်အစွဲလေးကို ယူပြီး လက်နဲ့ဖို့ကြည့်ပါ၊ မာသေးတယ်ထင်ပါကဆက်တည်ထားပါ၊ မပျော်မမာ မိမိ ကြိုက်တဲ့ အနေထား ရောက်ပါက ရေများ ကုန်အောင်ငွဲပါ(သွန်ပါ)။

၇ - ငွဲတာရောမကျန်တော့ရင် မီးဖိုပေါ်တွင် ထမင်းအိုးကို တစောင်းလေး တင်ပြီး တစ်ပတ် ပြည့်အောင် လှည့်လှည့်ပြီး အပူပေးပါ။

၈- ထမင်း သုံးဆောင်လို့ရပါပြီ



ဒီ algorithm မှာ ဆိုရင် စာသုံးမည့်လူအရောတ်ကိုတစ်ယောက်ချင်းစားနိုင်တဲ့ ပမာဏ ဒီနှစ်ခုဟာ သိဖို့လိုတဲ့ လိုအပ်ချက်ဖြစ်ပြီး ထမင်းချက်ဖို့အိုး၊ မီးမွေးဖို့ ထင်း သို့မဟုတ် မီးသွေး ကတော့ ရှိဖို့လိုတဲ့ လိုအပ်ချက်ဖြစ်ပါတယ်။ ဒါကြောင့် အဲဒါရိုဖို့လိုတဲ့ လိုအပ်ချက်တွေမရှိရင် ဒီ algorithm ကို သုံးဖို့ မစဉ်းစားသင့်ပါဘူး။ သိဖို့လိုတဲ့ လိုအပ်ချက်ဖြစ်တဲ့ စာသုံးမည့်လူအရောတ်၊ တစ်ယောက်ချင်း

စားနိုင်တဲ့ ပမာဏတွေကတော့ input ဖြစ်ပြီး စားလိုရသော ထမင်း ဆိုတာကတော့ လိုချင်တဲ့ result မဲ့ output ဖြစ်ပါတယ်။

ဒီ algorithm နှစ်ခုမှာ ပထမတစ်ခုက

- လူသက်သာတယ်။
- ဒါပေမဲ့ ရေပမာဏကိုကောင်းကောင်းထည့်တတ်မှ အဆင်ပြေမယ်၊ မဟုတ်ရင် ပျောတာ မာတာ ဖြစ်နိုင်တယ်။
- ထမင်းရေမင့်တဲ့အတွက် ရေခန်းချက်တာဖြစ်တဲ့အတွက် စာသုံးမည့်ထမင်းထဲမှာ အဟာရ ပြည့်ဝ စွာရတယ်။
- ဆန်နဲ့ သိပ်မတည့်သော ဆီးချို့သမားတွေ အတွက်တော့ ပေါင်းအိုးနဲ့ချက်တာဟာ ရေခန်းချက်တာမို့ ဆန်ရဲ့ အဟာရ ပြည့်ဝစွာပါနေလို့ မသင့်တော်ပါဘူး။
- လျှပ်စစ်မီးနဲ့ ပေါင်းအိုးရှိမှ ဒီနည်းနဲ့ချက်လို့ရပါမယ်။

ဒုတိယ တစ်ခုက

- လူပင်ပန်းတယ်၊ ဆန်က ငွဲလို့ရတဲ့ အနေအထားရောက်လား မာသေးသလား ဆိုတာ ကြည့်ဖို့ အချိန်ပေးဖို့လိုပါတယ်။
- ငွဲပစ်မှာမို့ ရေများလည်း သိပ်ပြဿနာမရှိဘူး၊ ဒါကြောင့် ရေပမာဏကို အဆင်ပြေသလို ထည့်လို့ရပါတယ်။
- လျှပ်စစ်မီးရှိဖို့မလိုဘူး။
- ထမင်းရေဉဲပစ်တာကြောင့် အဟာရလျော့သွားလို့ ဆန်နဲ့သိပ်မတည့်တဲ့ ဆီးချို့သမားတွေ အတွက် သင့်တော် ပါတယ်။
- ငွဲထားသောရေ(ထမင်းရေ) ကို ဆားလေး သကြားလေးထည့်ပြီး သီးသန့် သောက်လို့ ရ တယ်။

- နှစ်ထားတဲ့ ထမင်းရောကို ဟင်းချို့ချက်တဲ့ နေရာမှာလဲ သုံးလို့ရတယ်။



အထူးပြောချင်တာတစ်ခုကတော့ ဒုတိယ algorithm မှာ မီးမွေးတာကို ဆန်ဆေးပြီး ချက်မည့်အိုးထဲထည့်ပြီးမှ လုပ်လို့ရပေမဲ့ ဆန်ပမာဏ မတွက်ရသေးခင် ကတည်းက ကြိုတင်ပြီး မီးမွေးထားမယ်ဆိုတာဟာ algorithm ရေးသားသူရဲ့ ကျွမ်းကျင်မူဖြစ်တယ်။



မီးဆိုတာ မွေးမွေးချင်း မတက်ဘူး(မီးအားမကောင်းဘူး)၊ မီးတက်လာအောင် မီးအားကောင်းလာအောင် အချိန်ယူဖို့လို့တယ်။ မြန်မြန်တက်ချင်ရင်တော့ ယက်ခပ်ပေးဖို့လိုပါတယ်။ ဒီတော့ မီးကိုအရင်မွေးထားပြီးတော့မှ ချက်ရမည့် ဆန် ပမာဏတွက်မယ်၊ ဆန်ယူမယ်၊ ဆန်ဆေးမယ်၊ ဆန်ဆေးပြီး ချက်မည့်အိုးထဲထည့်မယ် စတဲ့အလုပ်တွေက ယူတဲ့ကြာချိန်တွေဟာ မီးအတက် အကိုက်ပဲဖြစ်တာမို့ မီးအတက် စောင့်စရာမလိုလို အချိန်ကုန်သက်သာတယ်၊ မီးအမြန်တက်ဖို့ ယက်ထိုင်ခပ်နေစရာမ လိုပါဘူး။ မီးအတက်ကို ထိုင်ပြီး စောင့်နေစရာလည်း မလိုတော့ပါဘူး။

ထမင်းချက်နည်းအမျိုးမျိုးရှိသလို၊ သချို့ပုစ္စာတစ်ပုဒ်မှာ တွက်နည်းအမျိုးမျိုးရှိသလို၊ ကိစ္စ တစ်ခု အတွက်လည်း ဖြေရှင်းနည်း algorithm တွေအများကြီးရှိပါတယ်။ အဲဒီလိုမျိုး ဖြေရှင်းနည်း တွက်ချက်နည်း algorithm တွေလေ့လာတဲ့အခါမှာ အားနည်းချက်၊ အားသာချက်၊ ကြာချိန် လိုအပ်ချက် စတာတွေအားလုံးကို သိအောင်လေ့လာရပါမယ်။ ကြာချိန်သိမှ မြန်တဲ့ algorithm

ကိုရွေးချယ်လိုရမှာပေါ့။ လိုအပ်ချက်သိမှ ကိုယ်နဲ့ကိုက်မကိုက်ဆုံးဖြတ်လိုရမှာပေါ့။ ဥပမာ ပေါင်းအိုးနဲ့
ထမင်းချက်ဖို့ဆို လျှပ်စစ်မီးလိုအပ်တယ်။ ဒီတော့ လျှပ်စစ်မီးမရရှိသေးတဲ့ အေသာမှာ အဲဒီ ချက်နည်းကို
သုံးလိုမရဘူး။ ဒါကြောင့် အားသာချက်၊ ကြာချိန်၊ လိုအပ်ချက် စတာတွေအားလုံးကို သိမှသာလျှင်
ကိုယ့်အခြေနဲ့ ဘယ် algorithm ကိုသုံးရမလဲ ဆိုတာ ကောင်းမွန်စွာ ဆုံးဖြတ်နိုင်မှာဖြစ်ပါတယ်။



Chapter အနေစုစုပ်

- Data Structure ဆိုတာ programming language မဟုတ်ပါ။
- programming language တစ်ခုခုကိုသုံးပြီး program ရေးသားတဲ့အခါ data တွေကို
ဘယ်လိုပုံစံနဲ့သိမ်းရင်ကောင်းမလဲ၊ အဲဒီ data တွေကို တွေက်ချက်တဲ့အခါမှာ ဘယ် algorithm
နဲ့ တွေက်သင်လဲဆိုတာ သိဖို့ data သိမ်းဆည်းနည်းတွေနဲ့ data တွေကို
တွေက်ချက်နည်းအဆင့်ဆင့် step by step procedure လိုခေါ်တဲ့ algorithm ကို
လေ့လာတာကို Data structure လို ခေါ်ပါတယ်။ ဒီလောက်ဆိုရင် data structure ဆိုတာ
ဘာလဲ သိလောက်ပြီလို ထင်ပါတယ်။
- နောက်ပြီး algorithm ကိုလုပ်ဖို့ ရှိဖို့လိုအပ်ချက်တွေကိုယ့်မှာရှိနေမှ သုံးလိုရမယ်။
သိဖို့လိုအပ်ချက် အားလုံးကို input လိုခေါ်ပြီး၊ လိုချင်တဲ့ အဖြေ result ကိုတော့ output ပါ။
လုပ်နည်းတွေက်ချက်နည်း ကိုတော့ process လိုခေါ်ပါတယ်။ ဒါကြောင့် algorithm တစ်ခုမှာ
- 1. input |
- 2. Process
- 3. output ဆိုပြီး အပိုင်း ၃ ပိုင်းပါ ပါမယ်။

“ကျောင်းတက်တုန်းက ဆရာမက စာသင်သွား

စာကတော့ ရင်မဲ့၊

ပေးသွားတော့ ဥပမာတွေတော့ အကုန်မှတ်မိတယ်။

အခဲလဲ Data structure တော့ မသိလိုက်ဘူး၊

ထမင်းတော့ မျိုးစံချက်တတ်သွားပြီ လိုများ ဖြစ်နေပြီလားမသိဘူး”

အခန်း (၆)

Algorithm မိတ်ဆက်

Variables

Algorithm အကြောင်းကို မပြောသေးင် Variable ကို အရင် ပြောပါမယ်။ **Variable** ဆိတာ memory ပေါ်မှာ သိမ်းဖို့ နေရာလေး တစ်နေရာယူတာကို variable လို့ ခေါ်ပါတယ်။

ဥပမာ ၂ နဲ့ ၅ နဲ့ ပေါင်းပါလို့ ပြောရင် ကျွန်မတို့ လူကြီးတွေကတော့ ဂဲ ဆိုတဲ့ အဖြေတန်းပေးမှာပါ။ ဒါပေမဲ့ အဲဒီလောက်မြန်မြန် မတွက်တတ်သေးတဲ့ ကလေးလေးတွေ သချာတွက်တာကို စဉ်းစာကြည့်ရအောင်ပါ။

“၂ ကို စိတ်ထဲမှာမှတ်”၊

“လက် ၅ ချောင်းထောင်”၊

“၂ ပြီးတော့ ၃၊ ၃ ပြီးတော့ ၄၊ ၄ ပြီးတော့ ၅၊ ၅ ပြီးတော့ ၆၊ ၆ ပြီးတော့ ၇” ဆိုပြီး လက်ကလေးတွေ ချိုးချိုး ပြီး တွက်သွားတာ သိကြမှာပါ။

အဲဒီမှာ ၂ ကို စိတ်ထဲမှာ မှတ် ဆိုတာ ၂ ကို စိတ်ထဲ သိမ်းလိုက်တာ၊ လက် ၅ ချောင်းထောင် ဆိုတာ ၅ ကို လက်မှာ သိမ်းလိုက်တာ၊ အဲဒီတော့ သိမ်းဖို့ နေရာ ၂ နေရာလိုတာပေါ့။ computer အနေနဲ့ ပြောမယ် ဆိုရင် variable 2 ခုပေါ့။ ဒါကို တွက်လိုရလာတဲ့ အဖြေ ၇ ကို ထပ်ပြီး သိမ်းရင် စုစုပေါင်း သိမ်းဖို့နေရာ နေရာ ၃နေရာ လိုသွားတာ ဆိုတော့ variable ၃ ခု လိုမှာပေါ့။ ဒါဆို variable ဆိုတာ ဘာလဲ သဘောပေါက်လောက်ပါပြီ။

variable တွေကို သုံးတော့မယ် ဆိုရင် အမည်ပေးရပါတယ်။ အမည်ကတော့ ကိုယ်ကြိုက်တာပေးလိုပါတယ်။ ဒါပေမဲ့ programming language ကသတ်မှတ်ထားတဲ့ စဉ်းကမ်း

ဘောင်အတွင်းမှာပဲ ဖြစ်ရပါမယ်။ Language တစ်ခုနဲ့ တစ်ခု variable name မှာ ဘာတွေပါလို့ရသလဲ ဆိုတာ သတ်မှတ်ထားတာလေးတွေ နည်းနည်းကွဲတာမျိုး ရှိပါတယ်။ အများအားဖြင့် တူညီကြတာကတော့ variable name မှာ

- Letter (a to z, A to Z), digit (0-9), underscore
- Don't start with digit

ဆိုလိုတာက variable name မှာ a to z အသေး၊ အကြီးရယ်၊ 0 to 9 ရယ်၊ underscore ရယ်ပဲ ပါလို့ရပါတယ်။ အဲဒီ သူခွင့်ပြုထားတဲ့ ၃ ခုကလွှဲပြီး ကျန်တဲ့ space ခြားတာတို့၊ ပေါင်းနှုတ်မြောက်စား၊ #, ! စတဲ့ special symbol တို့ ဘာမှ ပါလိုမရပါဘူး။ ပြီးတော့ ပါလိုရတဲ့ ၃ ခုတဲ့ digit နဲ့ စလို မရပါဘူး။ letter သို့မဟုတ် underscore (_) နဲ့ စပြီးရင်တော့ နောက်က digit, letter, underscore လိုက်လိုပါတယ်။

1num (မှား - 1 ဆိုတဲ့ digit နဲ့စတို့)

num1 (မှန် n ဆိုတဲ့ letter နဲ့စပြီး၊ နောက်ကလည်း letter နဲ့ digit ဲ ပါလို)

Birth year (မှား space ခြားထားလို့)

Birth_year (မှန် - letter နှင့် underscore ဲ ပါ)

_year1 (မှန် - underscore နှင့် ပြီး၊ နောက်က letter နဲ့ digit လိုက်)

Algorithm ဆိုတာ

Algorithm ဆိုတာ step by step procedure ဖြစ်တယ်။ ပြီးတော့ သချိုာတစ်ပုဒ်မှာ တွက်နည်းပေါင်းများစွာရှိသလို တစ်စုံတစ်ခုကို ဖြေရှင်းတွက်ချက်ဖို့ အတွက်လည်း algorithm တွေ များစွာရှိတယ်။ အဲဒီ algorithm တွေထဲကမှာ အရင်ဆုံးတော့ ကိုယ့်မှာ ရှိတာတွေနဲ့ ကိုက်ညီတဲ့

အသုံးပြုလိုရတဲ့ algorithm တွက် အရင်စစ်ထုတ်ရပါမယ်။ အဲဒီစစ်ထုတ်ထားတဲ့ အထဲကမှ အားသာချက်၊ အားနည်းချက်ကို လေ့လာပြီး မိမိ အဆင်ပြေတာကို ရွေးချယ်ရပါမယ်။ ဒါကြောင့် algorithm တစ်ခုကို လေ့လာတဲ့ အခါမှာ ဘယ်လို အခြေအနေမှာ သုံးနိုင်လဲ၊ အားသာချက်က ဘလဲ အားနည်းချက်က ဘာလဲဆိုတာ လေ့လာဖို့လိုပါတယ်။ စတဲ့ အကြောင်းအရာတွက်ကို ရှုံးမှာ ပြောခဲ့ပြီးသွားပါပြီ။

နောက်ပြီး algorithm တစ်ခုမှာ input, process, output ဆိုပြီး ၃ ပိုင်းပါမည့် အကြောင်းလဲ ပြောခဲ့ပြီးပါပြီ။

- **Input** - ဆိုတာက သိဖို့လိုတဲ့ လိုအပ်ချက်၊ တနည်းအားဖြင့် အသုံးပြုသူက ထည့်သွင်းပေးရမည့် ဝင်လာမည့် အချက်အလက်။
- **Process** - ဆိုတာက တွက်ချက်တာ၊
- **Output** - ဆိုတာကတော့ အဖြေ ထုတ်ပြေတာဖြစ်ပါတယ်။



လူတစ်ယောက်က ကျွန်တော့အသက် ဘယ်နှန်စုံပြီလဲ တွက်ပေးပါလို့ ပြောတယ် ဆိုပါတော့။ ဒါဆို ကျွန်မတို့ သူကို ဘာပြန်မေးကြမလဲ။ အကြမ်းယျင်းတွက်မယ်ဆိုရင် မွေးတဲ့ ခုနှစ်လောက်တော့ မေးရမှာပေါ့။ ဒီတော့ မွေးတဲ့ ခုနှစ်ဆိုတာ အသုံးပြုမည့်သူက ပြောရမှာ ဖြစ်တဲ့အတွက် input ဖြစ်ပါတယ်။ ပြီးတော့ ဘယ်လိုတွက်မလဲ။ ယခု လက်ရှိ ခုနှစ်ထဲကနေ မွေးတဲ့ ခုနှစ်ကို နှုတ်ပြီး တွက်မှာပေါ့။ ဒါဟာ တွက်ချက်တာဖြစ်လို့ process ပါ။ နှုတ်လို့ရလာတဲ့ (တွက်လို့ရလာတဲ့) အဖြေကို user ဆီ ပြန်ပြောတာက output ပါ။ ဒါဆို input, output, process ကို သဘောပေါက်လောက်ပါပြီ။

ပြီးတော့ ဒီမှာ input လက်ခံမှာက တစ်ခု - မွေးတဲ့ခန့်စွဲ၊ output ထုတ်ပေးရမှာက တစ်ခု - အသက် ဆိုတော့ input တစ်ခု၊ output တစ်ခု၊ computer အနေနဲ့ ပြောရရင် စုစုပေါင်း variable နှစ်ခုလိုပါမယ်။

အဲဒီ variable နှစ်ခုရဲ့ အမည်တွေကို စဉ်းကမ်းအတွင်းက ကိုယ်ကြိုက်တာ ပေးလို့ရပါတယ်။ ဒီမှာတော့ **birth_year** နှင့် **age** လို့ ပေးလိုက်ပါတယ်။

Algorithm : Age Calculation

1. Write ‘Enter your birth year’
2. Read birth_year.
3. Set age = 2021 – birth_year.
4. Write ‘Your age is’, age
5. Exit.

1. အရင်ဆုံး user ကို အသက်ရှိက်ထည့်ပါဆိုပြီး Enter your age လို့ user ကို ပြောချင်တာ၊ User မြင်အောင် ထုတ်ပြုချင်တာဖြစ်လို့ write ဆိုပြီး output ထုတ်ပြလိုက်ပါတယ်။
2. အဲဒီမှာ user က သူ့မွေးတဲ့ခန့်ကို ရှိက်ထည့်လိုက်တာနဲ့ Read လုပ်ပြီး ရှိက်ထည့်တဲ့ တန်ဖိုးကို birth_year ဆိုတဲ့ variable ထဲ သိမ်းလိုက်ပါတယ်။ (input)
3. ယခု ခန့်က 2021 ဖြစ်လို့ 2021 ထဲက user ရှိက်ထည့်လိုက်တဲ့ တန်ဖိုး birth_year ကို နှုတ်ပြီး တွက်ချက်လိုက်ပါတယ်။ တွက်ချက်တာဟာ process ဖြစ်ပြီး ရလာတဲ့ အဖြောက်ကို age ဆိုတဲ့ variable ထဲ သိမ်းလိုက်ပါတယ်။
4. တွက်လို့ရလာတဲ့ အဖြောက်ကို ကို ပုံစံတစ်ခုနဲ့ ထုတ်ပြုချင်ပါတယ်၊ ဥပမာ တွက်လို့ရလာတဲ့ တန်ဖိုးက 20 ဆိုရင် Your age is 20 လိုပေါ်ချင်ပါတယ်။ output ထုတ်ပြမှာမူး Write, Your age is ဆိုတဲ့ စာသားတို့က်ရှိက်ပေါ်ချင်လို့ single quote အတွင်းမှာ ရေးပါတယ်။ double quote ထဲ ရေးလည်း

ရပါတယ်။ ပြီးတော့ နောက်က age ပေါ်ချင်တာ၊ age ဆိုတဲ့ စကားလုံး တိုက်ရိုက်ပေါ်ချင်တာ မဟုတ်ဘူး၊ age ထဲက သိမ်းထားတဲ့ တန်ဖိုး ပေါ်ချင်တာမို့ single quote, double quote မလိုပဲ ဒီတိုင်းရေးပါတယ်။ ပြီးတော့ ရှုံးကစာသားနောက် ကပ်ပေါ်ချင်တာဖြစ်လို့ comma (,) နဲ့ ဆက်လိုက်ပါတယ်။

5. User လိုချင်တဲ့ အဖြေ ထုတ်ပြပြီးပြီး ဖြစ်လို့ program ကို ရပ်လိုက်တာက exit ဖြစ်ပါတယ်။

Function and Return Statement

အပေါ်က algorithm ကတော့ မွေးတဲ့ခုနှစ်တောင်းတယ်၊ တွက်ချက်တယ်၊ ပြီးလို အဖြေရလာတော့လဲ အဖြေကို တစ်ခါတည်း output ထုတ်ပြတာဖြစ်ပါတယ်။ တစ်ခါတည်း output ထုတ်ပြလိုက်တော့ သူထုတ်ပြတဲ့အတိုင်း မြင်ရပါမယ်။ user က ဘာမှ ဆက်လုပ်လို့ မရပါဘူး။ တကယ်လို့ user ကို စိတ်ကြိုက်လုပ်ခွင့်ပေးချင်ရင် return ပြန်ပေးဖို့လိုပါတယ်။ return ပြန်တယ်ဆိုတာ အဖြေကို user ဆီ ပြန်ပေးလိုက်တာ ဖြစ်ပါတယ်။ ဒီတော့ user က ပြန်ရလာတဲ့ အဖြေကို သူကြိုက်တဲ့ပုံစံနဲ့ output ထုတ်လို့ရသလို၊ အဲဒီရလာတဲ့ အဖြေပေါ်မှာ မူတည်ပြီး ဆက်ပြီး တွက်ချက်မှုတွေ လုပ်ချင်ပါကလည်း ဆက်လုပ်လို့ ရပါတယ်။ ဒါကြောင့် output တန်းပြီး ထုတ်ပေးထာမျိုးထက် return ပြန်ပေးတာကို ပို့သဘောကျပါတယ်။

နောက်တစ်ခုက အပိုင်းလေး ခွဲရေးတာကို function လိုက်ခေါ်ပါတယ်။ function ခွဲရေးခြင်းအားဖြင့် တစ်ခြားနေရာတွေကနေ ခေါ်ပြီး အသုံးပြုလို့ ရသွားမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် function လေးတွေခွဲရေးလေးရှိပါတယ်။ ခွဲရေးလေးရှိတယ်ဆိုတာထက် ခွဲပြီး ရေးကို ရေးကြပါတယ်။ နမူနာ ကြည့်ကြည့်ပါ။

Algorithm: Age Calculation

```

ageCalculation(birth_year)

    age = 2021 - birth_year

    return age

```

function ရေးပုံရေးနည်း ကို ပြန်ရှင်းရမယ်ဆိုရင်တော့ အောက်မှာ ဖော်ပြထားတဲ့ ပုံအတိုင်း ရေးရပါတယ်။

functionName(parameter)

Body

Function name – variable လိုပဲ သတ်မှတ်ထားတဲ့ စည်းကမ်းအတွင်းက ကိုယ်ကြိုက်တဲ့ အမည် ပေးလို့ ရပါတယ်။ နောက်တစ်ခု သိစေချင်တာက variable name နဲ့ function name လိုမျိုး programmer က စိတ်ကြိုက်ပေးတဲ့ အမည်မှန်သမျှကို Identifier လိုခေါ်ပါတယ်။ ဒါကတော့ ကြံ့တုန်းလေး ထည့်ပြောတဲ့ သဘောပါ။

Parameter – အလွယ်ပြောရင် input ပါပဲ။ တစ်ခုထက်ပိုခဲ့ရင်တော့ တစ်ခုနဲ့ တစ်ခုကြား comma (,) ခြားပြီး ရေးလေ့ရှိပါတယ်။ input လက်ခံဖို့မလိုဘူးဆိုရင်တော့ အထဲက ဘာမှ မပါပေမဲ့ function ဖြစ်တဲ့အတွက် လက်သည်းကွင်း အဖွင့် အပိတ်ကတော့ ပါကို ပါရပါမယ်။

Body - ကတော့ တွက်တာ ချက်တာ၊ အဖြေထွက်လာရင် return ပြန်တာတွေ လုပ်မှာဖြစ်ပါတယ်။

အပေါ်က ရေးပြထားတဲ့ ageCalculator function ကိုလှမ်းခေါ်ပြီး 2000 ခုနှစ်မွေးဖားဘူးအတွက် တွက်ချင်ရင် ans = ageCalculator(2000) လို့ လှမ်းခေါ်ရပါမယ်။ လှမ်းခေါ်လိုက်တာနဲ့ အပေါ်က function ကိုသွား အလုပ်လုပ်ပြီးတော့ လက်သည်းကွင်းထဲမှာ ထည့်ပေးလိုက်တဲ့ 2000 ဟာ function ရဲ့ birth_year ထဲကို ဝင်သွားမှာ ဖြစ်ပါတယ်။

နောက်တစ်ခုက `ans = ageCalculator(2000)` လိုပြေးပြီး လုမ်းချေထားတာဟာ function ကနေ တွက်ချက်ပြီး return ပြန်ပေးထားတဲ့ အဖြောက်`ans` ထဲထည့်မယ်လို့ ပြောတာဖြစ်ပါတယ်။ `ans` သည် variable name ဖြစ်တဲ့ အတွက် သတ်မှတ်ထားတဲ့ စည်းကမ်းအတွင်းကနေ မိမိနှစ်သက်ရာ အမည် ပေးလို့ရပါတယ်။

2002 မွေးဖားတဲ့သူအတွက် တွက်ချင်ရင် 2002 ထည့်ပေးပြီး နောက်တစ်ကြိမ်ထပ်ပြီး ချေလိုက်ရုပါပဲ။ `result=ageCalculator(2002)`၊ `ageCalculator` ရဲ့ parameter လက်ခံတဲ့ `birth_year` ထဲကို 2002 ရောက်သွားမှာ ဖြစ်ပြီး၊ 2021 ထဲက 2002 နှစ်လို့ရတဲ့ အဖြောက်`function` က `return` ပြန်လိုက်တဲ့အခါ ဒီဘက်က လုမ်းချေတဲ့နေရာက ဖမ်းထားတဲ့ `result` ဆိုတဲ့ variable ထဲကို ဝင်သွားမှာ ဖြစ်ပါတယ်။ ဒါဆို `function` က `return` ပြန်ရင် လုမ်းချေသုံးတဲ့ နေရာကနေပြီး အဖြေ ပြန်ဖမ်းပေးရမယ်ဆိုတာ သဘောပေါက်လောက်ပါပြီ။



ထောင့်မှန်စတုဂံ တစ်ခုရဲ့ `area` တွက်တဲ့ algorithm လေးကို input,output အစအဆုံးပုံစံ တစ်ခုနဲ့ function ပုံစံ တစ်ခုကို ကိုယ့်ဟာကိုယ် ရေးကြည့်ကြည့်ပါ။ ထောင့်မှန်စတုဂံရဲ့ `area` ပုံသေနည်းက အလျား * အနဲ့ ဖြစ်လို့ အလျားနဲ့ အနဲ့ မသိပဲ တွက်လို့မရပါဘူး။ ဒါကြောင့် user ဆီမေးရမှာက အလျား နဲ့ အနဲ့ တွက်ပေးရမှာက `area` ပါ။ ဒီလောက်ဆို ရေးတတ်လောက်ပါပြီ။



Chapter အနှစ်ချုပ်

ရှုမှာ ပြောခဲ့သလိုပဲ ယနေ့ခေတ် အသုံးပြုနေကြတဲ့ programming language တွေ အများကြီးရှုပါတယ်။ Programmer တစ်ယောက် နဲ့ တစ်ယောက်ဟာ ကျမ်းကျင်တာတွေ မတူညီကြပါဘူး။ နှမူနာ ပြောရမယ် ဆိုရင်တော့ Java ကျမ်းကျင်တဲ့ programmer တစ်ယောက်ဟာ

PHP ကိုတော့ နားလည်ချင်မှ နားလည်ပါလိမ့်မယ်။ ဆိုလိုတာက programmer တစ်ယောက်ဟာ language ပေါင်းစုံ မကျမ်းကျင်နိုင်ပါဘူး။

ဒါဆို သူတို့ ဘုံနားလည်တာ ဘာလဲ ဆိုတော့ algorithm တွေ၊ pseudocode တွေ၊ Flowchart တွေ ဖြစ်ပါတယ်။ algorithm ဆိုတာ programmer တွေ အားလုံးနားလည်ပါတယ်။ Algorithm တစ်ခုဟာ လုပ်ဆောင်ရမည့် အဆင့်ဆင့်ကို general ပြောထားတာ ဖြစ်လို့ Java သမားက အဲဒီ algorithm ကို အသုံးပြုရင် java ပုံစံနဲ့ ရေးမယ်၊ PHP သမားက အသုံးပြုမယ်ဆိုရင်လည်း PHP ပုံစံနဲ့ ပြောင်းရေးမှာပေါ်လေ။

Algorithm တစ်ခုဆိုတာ programmer တွေ အားလုံးနားလည်အောင်၊ ယူသုံးလို့ရအောင် ရည်ရွယ်တာ ဖြစ်ပါတယ်။ ဒါကြောင့် algorithm တစ်ခုဟာ

1. ဘာတွေကို input အဖြစ်လက်ခံမယ်၊ ဘယ်ဟာတွေကို output အဖြစ် ထုတ်ပေးမယ်ဆိုပြီး
input, output ကို ရှင်းရှင်းလင်းလင်း တိတိကျကျ ဖော်ပြ သတ်မှတ်ပေးထားရပါမယ်။
2. Step by step အလုပ်လုပ်ပုံတွေဟာ ဟိုလိုလို ဒီလိုလို ဒီဟ ဖြစ်စရာတွေ မပါရှိပဲ
ပြတ်သားရှင်းလင်း နေရပါမယ်။
3. Algorithm တစ်ခုဟာ programmer တိုင်း၊ programming language တိုင်းက ယူသုံးလို့
ရအောင် programming language တစ်ခုခု အပေါ်မှာ မိခိုနေတာမျိုး မဖြစ်ရပါဘူး။

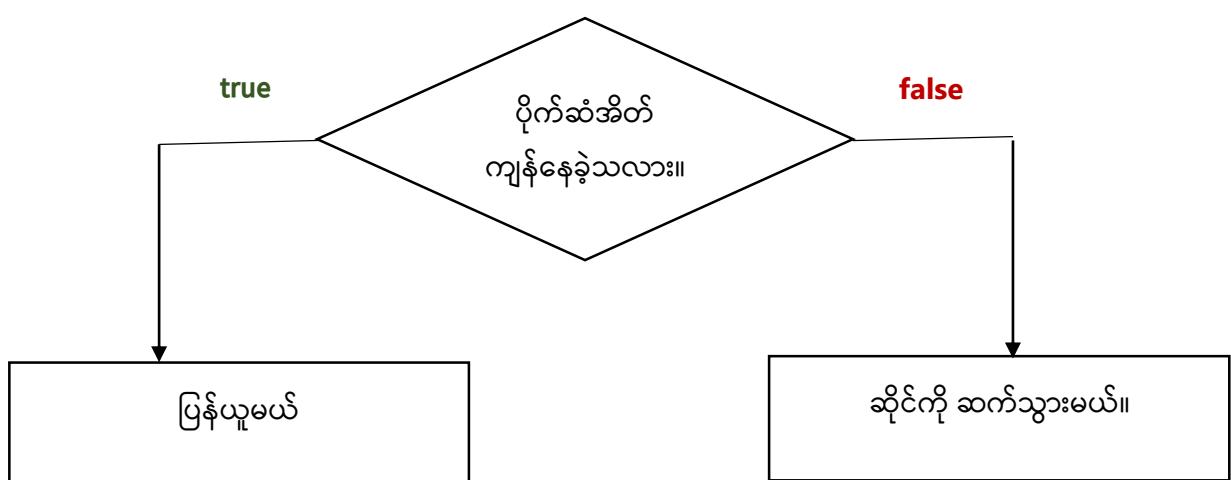
အခန်း (၇)

Control Flow and Flowchart

Conditional or Selection statement

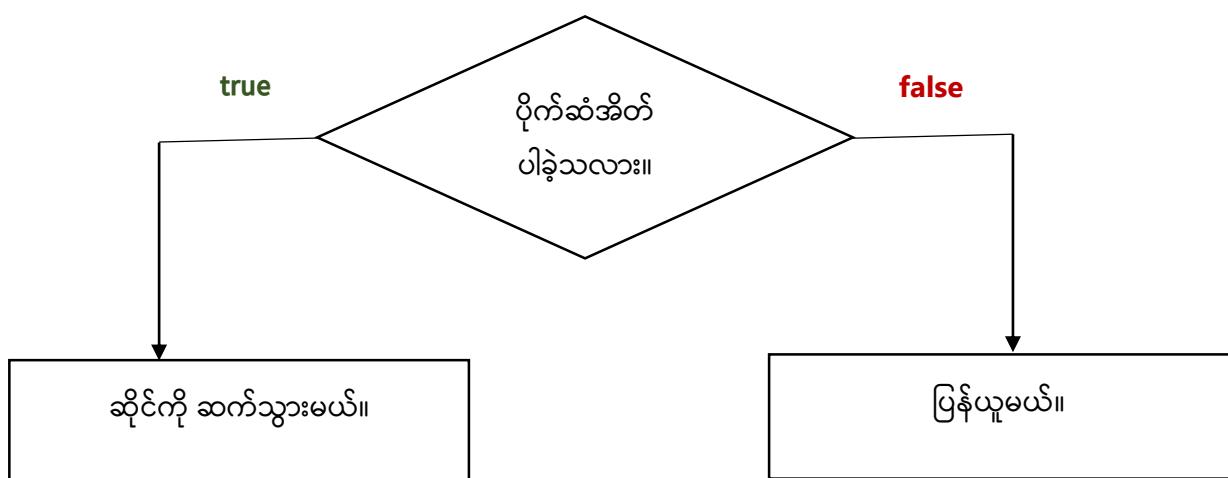
Algorithm ဆိုတာ step by step procedure - လုပ်ဆောင်ရမည့် အဆင့်ဆင့် လို ပြောခဲ့တယ်။ ဒါပေမဲ့ အဲဒီအဆင့်တွေ ထဲမှာ တချို့အဆင့်တွေက လုပ်ကိုလုပ်ရတာမျိုးဖြစ်ပေမဲ့၊ တချို့အဆင့်တွေ ကတော့ လုပ်ချင်လုပ်မယ်၊ မလုပ်ချင် မလုပ်ဘူးဆိုတဲ့ အခြေနေမျိုးတွေ ဖြစ်ပါတယ်။

နမူနာအနေနဲ့ မုန်သွားစားမယ်ဆိုပါတော့။ ဘာစားမလဲ အရင်စဉ်းစားရမယ်၊ ပြီးရင် အဲဒီရွေးချယ်လိုက်တဲ့ ဆိုင်ကိုသွားမယ်၊ ပြီးရင် မှာစားမယ်၊ ပြန်လာမယ်၊ ဒါတွေဟာ လုပ်ကို လုပ်ရမည့် အဆင့်တွေ ဖြစ်ပါတယ်။ သို့သော် ဆိုင်ကို သွားတဲ့ လမ်းတစ်ဝက်ရောက်မှ ပိုက်ဆံအိတ် မေ့ကျန်ခဲ့တာ သတိရတယ် ဆိုပါတော့။ အိမ်ကို ပြန်ပြီး ပိုက်ဆံအိတ် ပြန်ယူရပါမယ်။ အိမ်ကို ပိုက်ဆံအိတ်က အမြဲတမ်း ပြန်ယူမှာလား ဆိုတော့ မဟုတ်ပါဘူး။ မေ့ကျန်ခဲ့မှ ပြန်ယူမှာပါ။ အဲဒီလို့ ဖြစ်မှ လုပ်မှာ၊ မဖြစ်ရင် မလုပ်ဘူး ဆိုတဲ့ အခြေနေမျိုးကို conditional or selection statement လို ခေါ်ပါတယ်။



ဒီပုံလေးကို flowchart လို ခေါ်ပါတယ်။ ကြည့်လိုက်ရင် ရှင်းသွားမှာပါ။ ပိုက်ဆံအိတ် မေ့ကျန်နေခဲ့သလား စစ်တယ်။ အဲဒီလို့ စစ်တာကို condition စစ်တယ်လို ခေါ်ပါတယ်။ condition

စစ်တာကို flowchart မှာ diamond shape(စနွင်းမကင်း တုံး) ပုံစံနဲ့ ဆွဲပါတယ်။ condition စစ်လိုက်လို့ true ဆိုရင် ပိုက်ဆံအိတ်မေ့ကျန်ခဲ့တာဖြစ်တဲ့အတွက် “ပြန်ယူမယ်” ဆိုတဲ့ အလုပ် ကိုလုပ်ပါတယ်။ false ထွက်ခဲ့ရင်တော့ ပိုက်ဆံအိတ် မမေ့ကျန်ခဲ့ဘူးဆိုတဲ့ အဓိပ္ပာယ်ဖြစ်တဲ့အတွက်ကြောင့် “ဆိုင်ကိုဆက်သွားမယ်” ဆိုတဲ့ အလုပ်ကို လုပ်ပါမယ်။ “ပြန်ယူမယ်”၊ “ဆိုင်ကို ဆက်သွားမယ်” ဆိုတာတွေဟာ အလုပ်လုပ်ဆောင်တာတွေဖြစ်တဲ့အတွက် process ဖြစ်ပါတယ်။ process ကို flowchart မှာ rectangle (လေးထောင့်) ဖြင့် ကိုယ်စားပြုပါတယ်။



ဒီပုံက အပေါ်ကပုံကို နောက်တစ်မျိုးနဲ့ ပြန်ဆဲထားတာ ဖြစ်ပါတယ်။ အပေါ်က ပုံကတော့ “ပိုက်ဆံအိတ်မေ့ကျန်နေခဲ့သလား” ဆိုတာကို condition စစ်ထားတာ ဖြစ်ပြီး၊ ဒီပုံကတော့ “ပိုက်ဆံအိတ်ပါခဲ့သလား” ဆိုတဲ့ ရှုံးထောင့်ကနေ condition စစ်ထားတာဖြစ်ပါတယ်။ ဒါကြောင့် ဒီပုံမှာတော့ true ဆိုတာ ပိုက်ဆံအိတ်ပါခဲ့တယ် ဆိုတဲ့ အဓိပ္ပာယ် ဖြစ်တဲ့အတွက် “ဆိုင်ကို ဆက်သွားမယ်” ဆိုတဲ့ အလုပ်ကို လုပ်မှာဖြစ်ပြီး၊ false ကတော့ “ပိုက်ဆံအိတ် မပါခဲ့ဘူး” လို့ အဓိပ္ပာယ်ရတဲ့အတွက်ကြောင့် “ပြန်ယူမယ်” ဆိုတဲ့ အလုပ်ကို လုပ်သွားမှာဖြစ်ပါတယ်။ ဒါဆို condition ဆိုတာ ကိုယ်ကြိုက်တဲ့ ရှုံးထောင့်ကနေ စစ်လိုဂဲတယ် ဆိုတာ နဲ့ true နှင့် false ထွက်တာတွေကို သဘောပေါက်လောက်ပြီ ထင်ပါတယ်။

အဲဒီ ပုံနှစ်ပုံကို code အနေနဲ့ ရေးကြည့်ပါမယ်။ ပထမပုံဆိုရင်

If ပိုက်ဆံအိတ်ကျန်နေခဲ့သလား then:

ပြန်ယူမယ်။

Else:

ဆိုင်ကို ဆက်သွားမယ်။

ဒါလေးကို ကြည့်လိုက်ရင် else ဆိုတဲ့ false ဆိုတဲ့ အဓိပ္ပာယ်ဆိုတာ သဘော ပေါက်ကြမှာပါ။

ဒုတိယပုံဆိုရင်

If ပိုက်ဆံအိတ်ပါခဲ့သလား then:

ဆိုင်ကို ဆက်သွားမယ်။

Else:

ပြန်ယူမယ်။



စာမေးပွဲတစ်ခုမှာ Myanmar, English, math ဆိုတဲ့ ဘာသာရပ် 3 ခုဖြေရတယ်။
သုံးဘာသာလုံးအောင်ရင် စာမေးပွဲအောင်မယ်၊ တစ်ဘာသာကျတာနဲ့ စာမေးပွဲ ကျမယ်။ အောင်မှတ်က
40 ဖြစ်ပါတယ်။

If $\text{Myanmar} \geq 40$ and $\text{English} \geq 40$ and $\text{Math} \geq 40$ then:

Write “You pass the exam”

Else:

Write “You fail the exam”

ရမှတ်က 40 နဲ့ ညီလဲအောင်တယ်၊ 40 ထက်ကြီးလဲ အောင်တယ်။ ဒါကြောင့် \geq ကို သုံးတယ်။

Greater than or equal လို့ ဖတ်ပါတယ်။ Or ဆိုတဲ့ အတိုင်းတစ်ခုမှန်ရင်၊ တစ်ခုဖြစ်ရင် ရတယ်

ဆိုတဲ့အတွက် > ကြီးလည်းရတယ်။ = ညီလည်း ရတယ် ဆိုတဲ့ သဘောပါ။ ပြီး တော့ ဒါ ဘာသာလုံးအောင်မှ အောင်မှာ ဖြစ်တဲ့အတွက် ကြားထဲမှာ and နဲ့ ဆက်ပေးရပါတယ်။ ရှုံးက သင်ခန်းစာတွေကို သေချာဖတ်လာတဲ့ သူ့အတွက် အထူးအထွေ ရှင်းပြနေစရာမလိုလောက်ဘူး ထင်ပါတယ်။ သဘောပေါက်မှာပါ။ မပေါက်ရင်တော့ and နဲ့ or အကြောင်း ရှုံးမှာ ရှင်းထားတာ ပြန်ဖတ်ပါ။

ကဲ အောင်တဲ့ ရွှေထောင်ကနေ ရေးပြီးသွားတာ ဖြစ်လို့ ကျတဲ့ ရွှေထောင့်ကနေ ရေးကြည့်ကြည့်ပါမယ်။ နားလည်မှုလွှဲပြီး တစ်ခုခုဆို ၂မျိုးရေးပေးရတယ် ထင်မနေပါနဲ့အေး။ တစ်မျိုးပဲ ရေးရင် ရပါပြီ။ နားလည်အောင် ၂ မျိုးရေးပြီး ရှင်းပြတာ ဖြစ်ပါတယ်။ ကျတာကတော့ တစ်ဘာသာကျရင်ကျမယ် တစ်ခုဖြစ်တာနဲ့ ကျမှာ ဖြစ်တဲ့အတွက် or နဲ့ရေးပါမယ်။ အမှတ် 40 ထက် ငယ်ရင် ကျမှာ ဖြစ်ပါတယ်။

If Myanmar < 40 or English<40 or Math<40 then:

Write “You fail the exam”

Else:

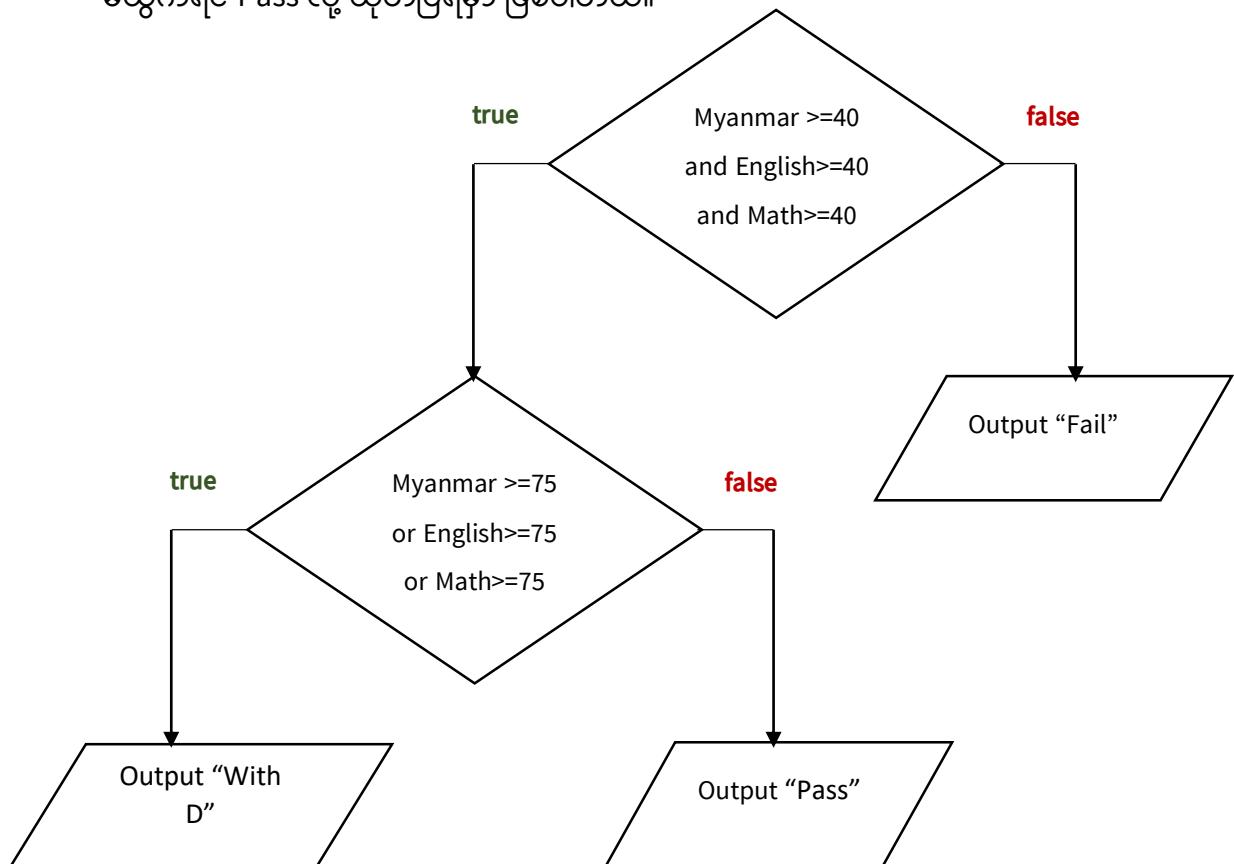
Write “You pass the exam”

Nested Conditional Statement

Nested Conditional Statement ဆိုတာ conditional statement ထဲမှာ နောက်ထပ် conditional statement ထပ်ပြီး ပါတာဖြစ်ပါတယ်။

 စာမေးပွဲတစ်ခုမှာ Myanmar, English, Math ဆိုတဲ့ ဘာသာရပ် ဒု ဒု ဖြေရတယ် ဆိုပါတော့။ အဲဒီမှာ ကျတဲ့သူကို Fail လို့ ထုတ်ပြချင်တယ်။ အနည်းဆုံး ၁ ဘာသာ ဂုဏ်ထူးထွက်ရင် with D လို့ ထုတ်ပြချင်တယ်။ ဂုဏ်ထူးမထွက်ပဲ ရိုးရိုးအောင်တဲ့ သူတွေကို Pass လို့ ထုတ်ပြချင်တယ် ဆိုပါတော့။

- **Pass / Fail** က ၃ ဘာသာလုံးအောင်မှ အောင်မှာ ဖြစ်တဲ့အတွက် အောင်တာကို စစ်ရင် $\text{Myanmar} \geq 40 \text{ and English} \geq 40 \text{ and Math} \geq 40$ ဆိုပြီး and နဲ့ စစ်ရပါမယ်။ **Fail** က တစ်ဘာသာ ကျတာနဲ့ ကျမှာ ဖြစ်တဲ့ or နှင့် စစ်ပါမယ်။ ဒါပေမဲ့ အောင်တာနဲ့ ကျတာ နှစ်ခုထဲ က တစ်ခုစစ်ရင် ရပါပြီ။ ဘာလို့လဲ ဆိုရင် မအောင်ရင် ကျပြီး၊ မကျရင် ဖြစ်လို့ ဖြစ်ပါတယ်။
- **With D** က တစ်ဘာသာ ဂုဏ်ထူးထွက်တာနဲ့ ထုတ်ပြမှာ ဖြစ်လို့ or နှင့် စစ်ရမှာ ဖြစ်ပါတယ်။ 75 က ဂုဏ်ထူးမှတ် ဖြစ်တယ် ဆိုပါစိုး။ ဒါပေမဲ့ ရမှတ်က 60, 30, 80 ဆိုရင်ကော့ တစ်ဘာသာ ကျနေတာကြီး ကို ဂုဏ်ထူး ပေးမလားဆိုရင် မပေးပါဘူး။ ဘာသာစုံအောင်တဲ့ သူကိုမှ ဂုဏ်ထူးပေးမှာ ဖြစ်ပါတယ်။ ဒါဆို အောင်မှ ပေးမှာ ဖြစ်တဲ့အတွက် အောင်သလား အရင် စစ်ရပါမယ်။ အောင်တဲ့ သူကိုမှ ဂုဏ်ထူး ထွက်သလား ထပ်စစ်၊ ထွက်ရင် With D လို့ ထုတ်ပြပြီး၊ မထွက်ရင် Pass လို့ ထုတ်ပြရမှာ ဖြစ်ပါတယ်။



Flowchart မှာ input, output ကို parallelogram နဲ့ ဆွဲပါတယ်။ တစ်ခြား symbol သုံးတာလဲ ရှိပါသေးတယ်။ အဲဒါကိုတော့ နောက်မှ ပြောပြပါမယ်။

ဒီပုံမှာ အောင်သလားဆိုတဲ့ ဘက်ခြမ်းမှာ မှ condition ထပ်လာတာကြောင့်၊ conditional statement ထဲ conditional statement ပြန်လာတာရှိ nested ဖြစ်ပါတယ်။ code အနေနဲ့ ရေးရင်တော့

If $\text{Myanmar} \geq 40$ and $\text{English} \geq 40$ and $\text{Math} \geq 40$ then:

If $\text{Myanmar} \geq 75$ or $\text{English} \geq 75$ or $\text{Math} \geq 75$ then:

 Write “With D”

Else:

 Write “Pass”

Else:

 Write “Fail”

ကဲ အောင်တဲ့ ရူးထောင့်ကနေ စစ်ပြီးတော့ ကျတဲ့ ရူးထောင့်ကနေ စစ်တာလေး နောက်တစ်မျိုး ရေး
ကြည့်ကြည့်ရအောင်ပါ။

If $\text{Myanmar} < 40$ or $\text{English} < 40$ or $\text{Math} < 40$ then:

 Write “Fail”

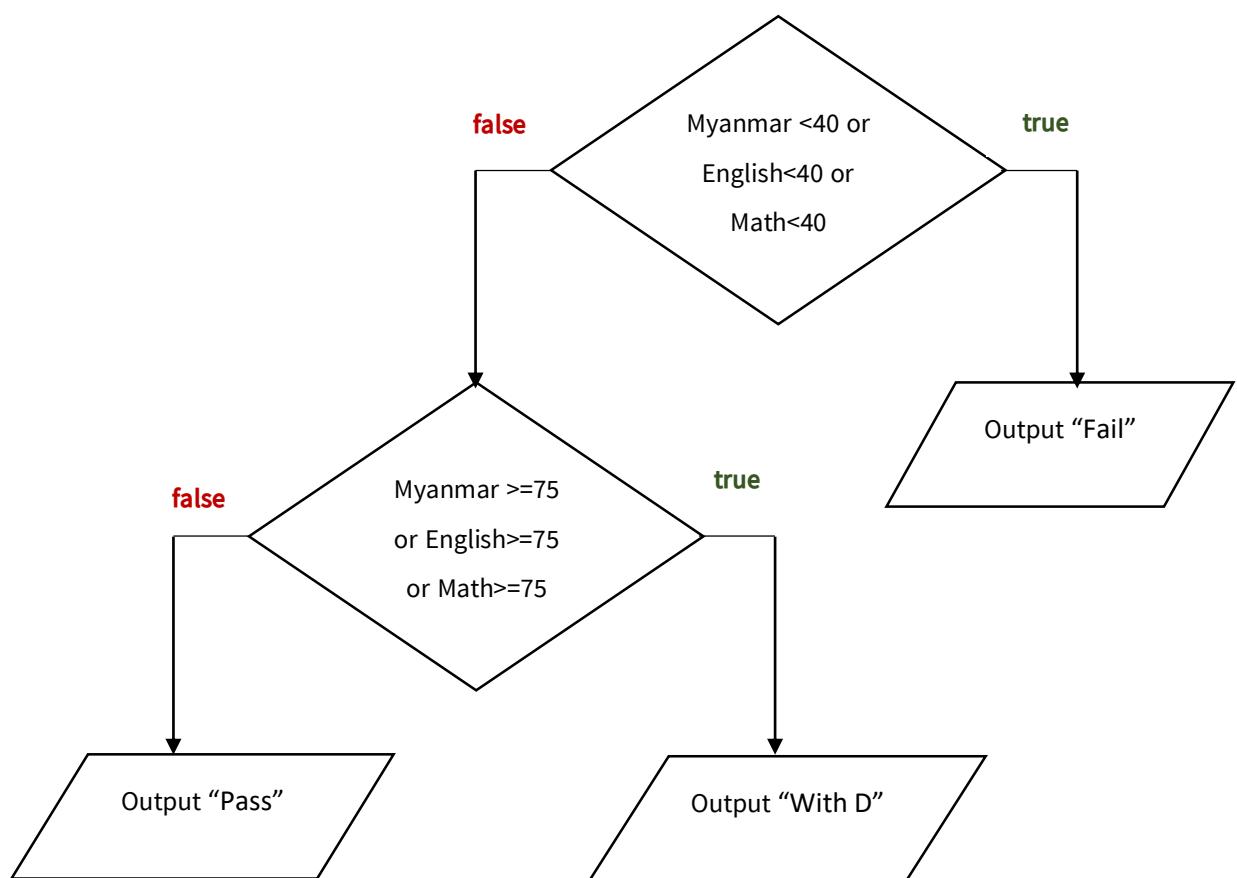
Else:

If $\text{Myanmar} \geq 75$ or $\text{English} \geq 75$ or $\text{Math} \geq 75$ then:

 Write “With D”

Else:

 Write “Pass”



ကဲ အထူးအထွေ ထပ်မရှင်းပဲ ကြည့်လိုက်တာနဲ့ သဘောပေါက်မယ် ထင်ပါတယ်။

Nested ရဲ့ နောက်တစ်နည်း:

If Myanmar<40 or English<40 or Math<40 then:

 Write “Fail”

Else If Myanmar>=75 or English>=75 or Math>=75 then:

 Write “With D”

Else:

 Write “Pass”

ရေးပြထားသလို Nested ရေးနည်းကို နောက်တမျိုးရေးလို့ ရပါသေးတယ်။ Else ရယ် သူ ရဲ့အောက်က If ရယ်ပေါင်းပြီး Else IF ဆိုပြီး ရေးလိုက်လိုလည်း ရပါတယ်။ အလုပ်လုပ်ပုံဟာ အတူပဲဖြစ်ပါတယ်။ ဒီလို ဖြစ်သွားမှာပါ။

အလွယ်မှတ်လို့ရတာက စစ်စရာ 3 ခုဆိုရင် ရှေ့ဆုံးတစ်ခုက IF နဲ့ ရေး၊ နောက်ဆုံးတစ်ခုက Else နဲ့ ရေး၊ အလယ်က ဟာတွေက Else if နဲ့ ရေး၊ Else တစ်ခုကလွှဲပြီး IF ရော၊ Else if ပါ condition လိုက်ပါတယ်။

 num ဆိုတဲ့ ကိန်းလေးကို even positive လား၊ even negative လား၊ odd positive လား၊ odd negative လား စစ်ပေးရမယ် ဆိုပါတော့။

2 => even positive

-2 => even negative

3 => odd positive

-3 => odd negative

ဒါကြောင့် ကိန်းတစ်လုံးဟာ အဲဒီ င့် မျိုးထဲက တစ်မျိုးမျိုး ဖြစ်မှာ ဖြစ်ပါတယ်။ ရှေ့ဆုံးတစ်ခုကို if နဲ့ စစ်မယ်၊ နောက်ဆုံးတစ်ခုကို else နဲ့ စစ်မယ် ဆိုတော့ အလယ်မှာ J ခုကျန်ပါတယ်။ အဲဒီ J ခုကတော့ else if နဲ့ ရေးရမှာ ဖြစ်ပါတယ်။ Else တစ်ခုကလွှဲပြီး ကျန်တဲ့ if ရော၊ else if တွေပါ condition ပါရမှာ ဖြစ်ပါတယ်။ even ဆိုတာ 2 နဲ့ စားပြတ်တာဖြစ်ပြီး၊ odd က 2 နဲ့ စားလို့ မပြတ်(အကြောင်း 0 နဲ့မညီတာ)၊ positive ကတော့ ≥ 0 ဖြစ်ပြီး၊ negative ကတော့ < 0 ဖြစ်တာ ဖြစ်ပါတယ်။

If (num mod 2) == 0 and num>=0 then:

Write num, “ is even positive”

Else If (num mod 2) == 0 and num < 0 then:

Write num, “ is even negative”

Else If (num mod 2) != 0 and num>=0 then:

Write num, “ is odd positive”

Else:

Write num, “ is odd negative”

အလုပ်လုပ်ပုံက

- IF နောက်က condition မှန်တာနဲ့ If အောက်က Write num, “ is even positive” ဆိုတာကို အလုပ်လုပ်ပြီး အောက်က Else if တွေ ထွေကို အလုပ်လုပ်မှာ မဟုတ်တော့ပါဘူး။ ဘာကြောင့်လဲဆိုရင် selection ဆိုတာ အများကြီးထဲက တစ်ခုကို ရွှေးချယ်တာ ကြောင့်ဖြစ်ပါတယ်။ သို့သော် နောက်ထပ် IF တစ်ခု ထပ်လာခဲ့ရင်တော့ အလုပ်လုပ်မှာ ဖြစ်ပါတယ်။ ဘာကြောင့်လဲဆိုရင် IF ဆိုတာ selection အသစ်တစ်ခု စတယ် လို့ အဓိပ္ပာယ် သက်ရောက်တာ ကြောင့် ဖြစ်ပါတယ်။
- If နောက်က condition မှားတယ်ဆိုမှာ ပထမ else if ကို ရောက်လာပြီး သူရဲ့ conditon ကို စစ်မှာ ဖြစ်ပါတယ်။ သူ့ condition မှန်ခဲ့ ရင် သူအောက်က Write num, “ is even negative” ကို လုပ်ပြီး အောက်က else if နဲ့ else ကို လုပ်မှာ မဟုတ်တော့ပါဘူး။
- ပထမ else If နောက်က condition မှားတယ်ဆိုမှာ ဒုတိယ else if ကို ရောက်လာပြီး သူရဲ့ conditon ကို စစ်မှာ ဖြစ်ပါတယ်။ သူ့ condition မှန်ခဲ့ ရင် သူအောက်က Write num, “ is odd positive” ကို လုပ်ပြီး အောက်က else ကို လုပ်မှာ မဟုတ်တော့ပါဘူး။

- ဒုတိယ else if ပါမှားတယ်ဆိုမှ else ကို အလုပ်လုပ်မှာ ဖြစ်ပါတယ်။ ဒါကြောင့် else ဆိုတာ အပေါ်က ဟာတွေ အားလုံးမှားတယ်ဆိုမှ အလုပ်လုပ်တာဖြစ်ပါတယ်။



အခုပြောသွားတဲ့ even positive လား၊ even negative လား၊ odd positive လား၊ odd negative လား စစ်တာကို nested ပုံစံနဲ့ ပြန်ရေးပေးပါ။

Conditional Statement for Range

Range တွေကိုစစ်တဲ့အခါ and နဲ့ စစ်ပေးရပါတယ်။ ဥ ပမာ 1 ကနေ 100 အတွင်းဆိုတာကို or ဖြင့်ရေးခဲ့ရင်

If $n \geq 1$ or $n \leq 100$ then:

ဝင်လာတဲ့ n တန်ဖိုးက 2000 ဆိုရင် အလုပ်လုပ်သွားမှာပါ။ ဘာလို့လဲ ဆိုရင် n တန်ဖိုး 2000 ဟာ 1 ထက်ကြီးနေတာဖြစ်လို့ $n \geq 1$ မှာ true ထွက်သွားပါတယ်။ or ဆိုတော့ တစ်ခုမှန်တာနဲ့ အလုပ်လုပ်တာဖြစ်လို့ လုပ်သွားမှာ ဖြစ်ပါတယ်။ အမှန်က 1 နဲ့ 100 ကြားဆိုတဲ့ အခြေနေဟာ ≥ 1 နှင့် ≤ 100 ဆိုတဲ့ အခြေအနေ နှစ်ခုလုံး မှန်မှ လုပ်ချင်တာဖြစ်လို့ and နဲ့ စစ်ရမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် ဘယ်နှင့် ဘယ်ကြားဆိုရင် and နှင့် စစ်တာဖြစ်ပါတယ်။

If $n \geq 1$ and $n \leq 100$ then:

လို့ ရေးရမှာ ဖြစ်ပါတယ်။



ရမှတ် ၉၀ နဲ့ အထက် ဆိုရင် grade A, ၈၀ - ၇၀ ကြားဆို grade B, ၆၀ - ၄၀ ဆိုရင် grade C, ၄၀ အောက်ဆိုရင် grade D လို့ ဆိုပါစို့။

If $mark \geq 90$ then:

Write “grade A”

Else If mark<90 and mark>=70 then:

Write “grade B”

Else If mark<70 and mark>=40 then:

Write “grade C”

Else:

Write “grade D”

တစ်ခုနဲ့ တစ်ခု နိုင်းယှဉ်ရင် ဖြစ်နိုင်ချေ ၃ ခုပဲရှိပါတယ်။ ငယ်တာရယ်၊ ညီတာရယ်၊ ကြီးတာရယ် ဖြစ်ပါတယ်။ ပထမ else if ကို ရောက်လာတာဟာ if နောက်က condition မှားလို့ ရောက်လာတာ ဖြစ်ပါတယ်။ if မှာက ≥ 90 (90 ထက်ကြီးသလား ညီသလား) စစ်ထားတာဖြစ်လို့ if မှားတယ်ဆိုတာ မကြီးလည်းမကြီးဘူး၊ ညီလည်းမညီဘူး ဆိုတော့ ငယ်တာဖြစ်ပါတယ်။ ဒါကြောင့် မထမ else if ကို ရောက်လာကတည်းက 90 ထက်ငယ်ပြီးသား ဖြစ်ပါတယ်။ ဒါကို 90 ထက်ငယ်သလား ထပ်စစ်တာစာ စစ်ဖို့ အချိန်ကုန်တာအပြင် မရှိပါဘူး။ ဒါကြောင့် ပထမ else if မှာ <90 ကို ဖြုတ်ခဲ့လို့ရပါတယ်။

ထို့အတူပါပဲ။ ဒုတိယ else if ကို ရောက်လာတာဟာ ပထမ else if မှားလို့ ဖြစ်တယ်။ ဒါကြောင့် 70 ထက်ငယ်ပြီးသားဖြစ်လို့ <70 ကို ဖြုတ်လို့ရပါတယ်။ ပြန်ရေးပြရမယ်ဆိုရင်

If mark ≥ 90 then:

Write “grade A”

Else mark ≥ 70 then:

Write “grade B”

Else If mark ≥ 40 then:

Write “grade C”

Else:

Write “grade D”

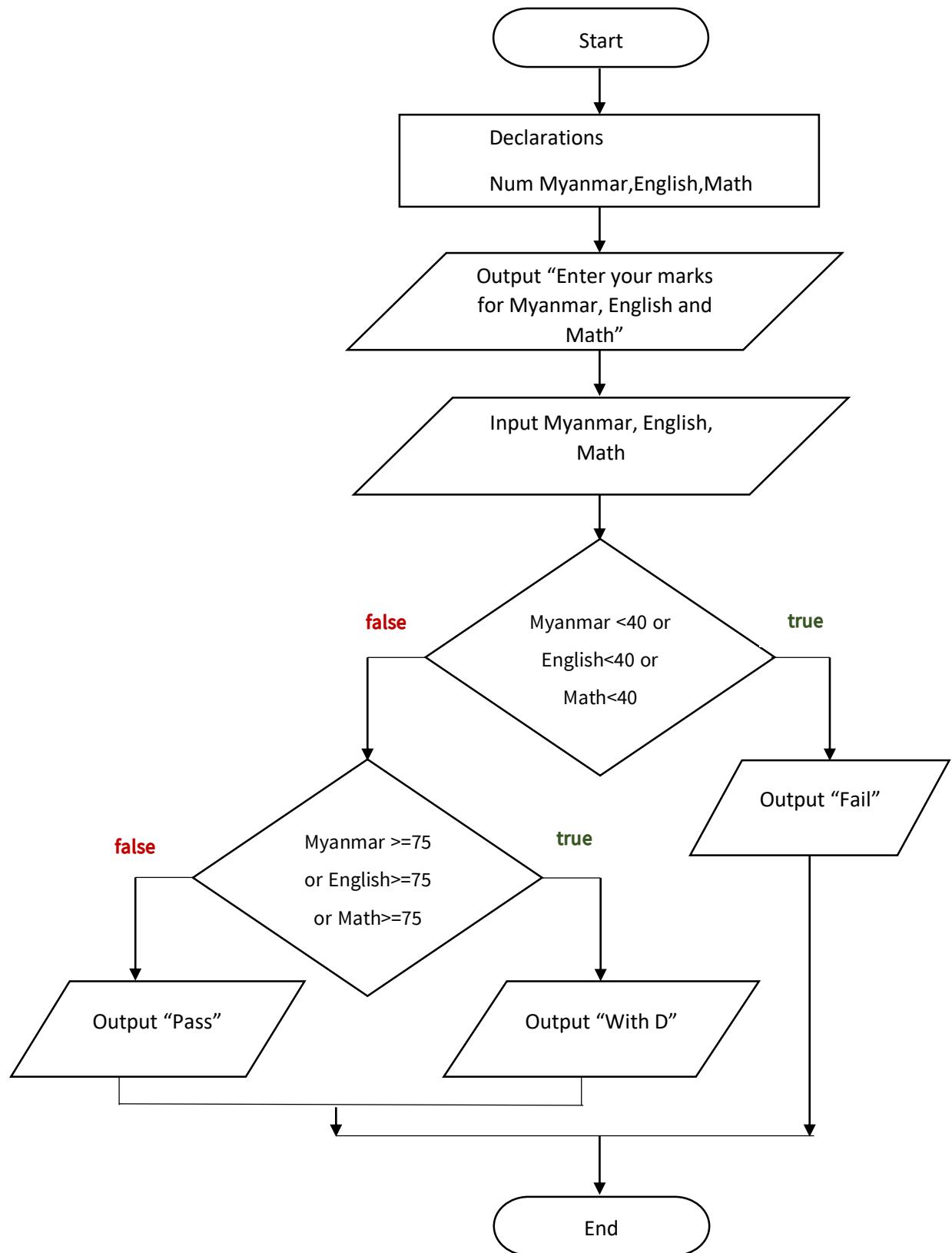
ဆိုပြီး ဖြစ်သွားပါမယ်။ Condition တစ်ခုလျော့သွားလို့ စစ်ရတာ အချိန်ကုန် သက်သာပါမယ်။

Flowchart Symbols

Symbol	အမည်	မှတ်ချက်
	Start/End	Program တစ်ခုရဲ့ အစနဲ့ အဆုံးကို ကိုယ်စားပြုပါတယ်။
	Process	တွက်တဲ့ ချက်တဲ့ အလုပ်လုပ်ဆောင်တာတွေ
	Input/ Output	User ဆီက input တောင်းတာနဲ့ user ကို output ထုတ်ပြုတဲ့ နေရာမှာ သုံးပါတယ်။
	Decision	Condition စစ်တဲ့ အခြေအနေမှာ သုံးပါတယ်။
	Arrow	ဘာပြီးရင် ဘာလုပ်မယ် ဆိုပြီး တစ်ခုနဲ့ တစ်ခု ဆက်တဲ့ အခါမှာ သုံးပါတယ်။
	On-page Connector	စာမျက်နှာ တစ်မျက်နှာထဲမှာရှိတဲ့ flowchart အစိတ်အပိုင်းတွေကို ချိတ်ချွင်ရင် သုံးပါတယ်။
	Off-page Connector	ပုံဆွဲလို့ တစ်မျက်နှာနဲ့ မဆုံးတွေ့ရင် တစ်မျက်နှာနဲ့ နောက် တစ်မျက်နှာ ကို ချိတ်ဆက်ကြောင်း ပြချွင်ရင် သုံးပါတယ်။

နှမူနာလေးဆွဲကြည့်ရအောင်။ မဆွဲခင် လေး တစ်ခုထပ်ပြောချွင်တာက ကလေးလေး တစ်ယောက် မွေးလာရင် အမည်ပေး ကပွန်းတပ် လုပ်ရသလိုပဲ ဘာ variables တွေသုံးမလဲဆိုတာကို **DataType** **variableName** ပုံစံနှင့် ကြောငြာပေးရပါတယ်။ Declaration လုပ်တယ်လို့ ခေါ်ပါတယ်။ ဒီမှာတော့

Myanmar, English, Math ဆိတဲ့ variable ၏ ခုံပါမယ်။ ၏ ခုံလုံးဟာ အမှတ်သိမ်းမှာ ဖြစ်လို့ data type က ငောန်း num ဖြစ်ပါမယ်။





လေ့ကျင့်ကြည့်ရအောင်။

User ထံမှ ခုနှစ် တစ်ခုကို တောင်းပါ။ ပြီးလျှင် leap year ဟုတ်မဟုတ် စစ်ပါ။ ခုနှစ်တစ်ခုသည် 4 ဖြင့် စားပြတ်ပြီး 100 ဖြင့် စားလို့ မပြတ်လျှင် သို့မဟုတ် 400 ဖြင့် စားပြတ်လျှင် leap year ဖြစ်ပါတယ်။ စားပြတ်တယ်ဆိုတာ အကြောင်း (mod တွက်တာ 0) 0 ရတာဖြစ်ပြီး စားမပြတ်ဘူး ဆိုတာ အကြောင်း 0 နဲ့ မညီတာ ဖြစ်ပါတယ်၊ ($=$ ညီတာ, $!=$ မညီတာ)

$2000 \Rightarrow 400 \neq$ စားပြတ်လို့ 2000 is leap year လို့ ထုတ်မယ်

$1900 \Rightarrow 400 \neq$ လည်း မပြတ်ဘူး။ 4 ဖြင့် ပြတ်ပြီး 100 နဲ့ ပါစားပြတ်နေလို့ 1900 is not leap year

$1904 \Rightarrow 4 \neq$ ပြတ်ပြီး 100 နဲ့ မပြတ်လို့ 1904 is leap year

Looping or Iteration

တူညီတဲ့ အလုပ်ကို ထပ်ခါထပ်ခါလုပ်တာကို looping ပတ်တယ် လို့ ပြောပါတယ်။ Iteration လို့လည်း ခေါ်ပါတယ်။ ပေါင်းတဲ့ အလုပ်ကို ထပ်ခါထပ်ခါလုပ်တယ်၊ နှုတ်တဲ့ အလုပ်ကို ထပ်ခါထပ်ခါလုပ်တယ်၊ စကားတစ်ခွန်းကို ထပ်ခါထပ်ခါ ပြောတယ်၊ စသည်ဖြင့် အဲဒီလို့ ထပ်ခါထပ်ခါ လုပ်တာ တွေအားလုံးဟာ looping တွေ ပါပဲ။ အလွယ်နည်းဆိုရင်တော့ looping ဆိုတာ အမူးသမားလို့သာ မှတ်ထား။ မူလားရင် ရစ်တယ်၊ ရစ်တယ်ဆိုတာ တူညီတဲ့ အကြောင်းအရာကို ထပ်ခါထပ်ခါ ပြောတာ။ အဲဒါ looping or iteration ပဲ။

ဤမာ 1, 3, 5, 7, 9, ..., 97, 99 ဆိုပြီး ထုတ်ပြချင်တယ် ဆိုပါတော့။ ကိုန်းစဉ်တန်းကို ကြည့်ပါ။

1 နောက်မှာ comma ပါတယ်၊ 3 နောက်မှာ comma ပါတယ်၊ 97 နောက်အထိ comma ပါတယ်။ 99 နောက်မှာ ကြ comma မပါတော့ဘူး။

looping ဆိုတာ တူညီတဲ့ အလုပ်ကို ထပ်ခါထပ်ခါ လုပ်တာဖြစ်လို့ မတူတဲ့ 99 ကို ခဏ ဖယ်ထားမယ် 1 ကနေ 97 အထိ ထုတ်လို့ ပြီးမှ သူကို သပ်သပ် လုပ်မယ်။

looping တစ်ခု မှာ အပိုင်း ၃ ပိုင်းပါပါတယ်။

- Initialization - ဘယ်ကစမှာလဲ
- Condition - ဘယ်အထိ အလုပ်မှာလဲ
- Update - တစ်ကြိမ်ပြီးတိုင်း ဘာလုပ်မှာလဲ။

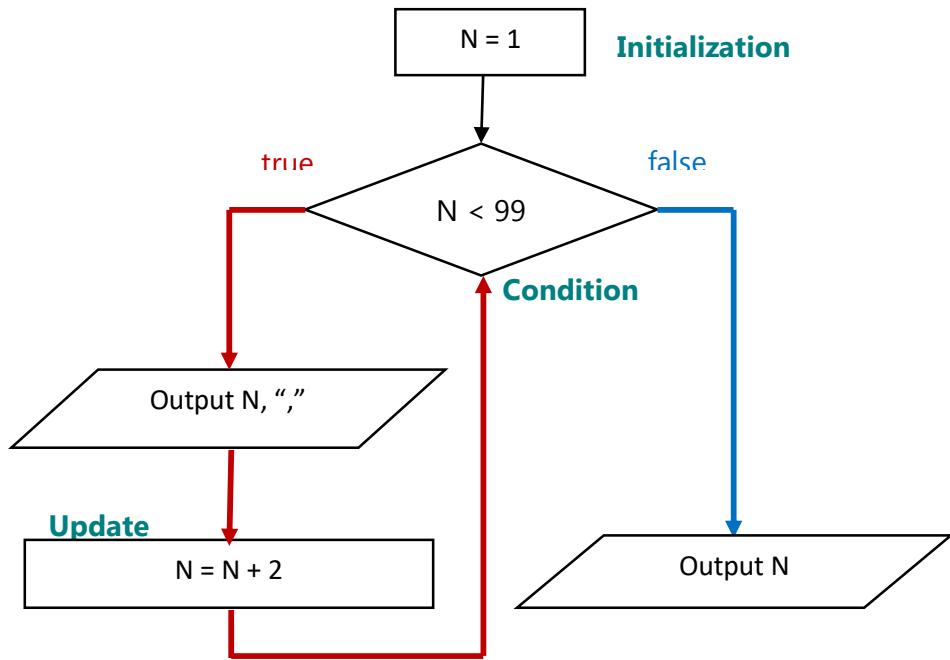
အခု ထုတ်ချင်တာက 1, 3, 5, 7, ..., 95, 97

ဒီတော့ ဘယ်က စ မလဲ 1 က စပါတယ်၊ ဘယ်ထိ လုပ်မလဲ 97 ထိ၊ တိတိကျကျ ပြောရရင် အရှေ့ကို
ပြန်ကြည့်လိုက်တော့ 1, 3, 5, ..., 95 ဆိုတာ 97 ထက်ငယ်တယ်၊ 97 ကတော့ 97 နဲ့ ညီတယ်၊ ဒီတော့ 97
ထက်ငယ်ပြီး ညီတဲ့ အထိ(≤ 97) လို့ ရေးလို့ရသလို့၊ 99 ထက်ငယ်တဲ့ အထိ ဆိုရင်လည်း (< 99)
အတူတူပါပဲ။

တစ်ကြိမ်ပြီးတိုင်း ဘာလုပ်မလဲ။ ကိန်းစဉ်တန်းမှာ 1 ပြီးရင် ၃၊ ၃ ပြီး ရင် ၅ စသည်ဖြင့် တစ်ခုပြီး
တစ်ခု ၂ တိုးသွားလို့ ဖြစ်ပါတယ်။ ဒါကြောင့် တစ်ကြိမ်ပြီး တိုင်း ၂ တိုး (၂ ပေါင်း) သွားပါမယ်။ အဲဒါကို
တန်ဖိုးပြောင်းတယ်ဟု update လုပ်တယ် လို့ ခေါ်ပါတယ်။ **Looping** တစ်ခုရဲ့ အသက်ဟာ update ပဲ
ဖြစ်ပါတယ်။ update မေ့ကျနဲ့ရင် N တန်ဖိုးဟာ 1 ကနေ မပြောင်းတော့ပါဘူး၊ ဒီတော့ N ဟာ 99 ထက်
အမြဲငယ်ပြီး looping ကနေ မထွက်တော့ပဲ တစ်သက်လုံး ပတ်နေပါတော့တယ်။ infinite loop
လို့ခေါ်ပါတယ်။ ဒါကြောင့် infinite loop မဖြစ်ရအောင် update မမေ့ဖို့ လိုပါတယ်။

ကဲ စလိုက်ကြရအောင်။

ဂဏန်းတွေ သိမ်းဖို့ variable name ကို N လို့ ပေးလိုက်ပါမယ်။



ဗုံမှာ initialization, condition, update ကို မြင်သာအောင် label တပ်ပေးထားပါတယ်။

Flowchart ကို ရတ်တရက် ကြည့်လိုက်ရင် conditional statement နဲ့ တူသလို ထင်ရပါတယ်။ လုံးဝ မတူပါဘူး။ conditional statement က အပေါ်ကို ပြန်မတက်ပါဘူး၊ looping က ပြန်တက်ပါတယ်။ true ဖက်ခြမ်းမှာ အပေါ်ကိုပြန်တက်တာကို မြင်သာအောင် နီညိုရောင်လေးနဲ့ ဆွဲပြထားပါတယ်။ Condition မှာ arrow က အောက်ကို ဆင်းသွားလို့ တစ်ခါပဲ လုပ်ပါတယ်။

Looping ကတော့ condition စစ်တယ်၊ မှန်ရင် အောက်ဆင်းပြီး လုပ်စရာရှိတာ လုပ်တယ်၊ ပြီးတာနဲ့ arrow ဟာ condition ဆီ ပြန်တက်တယ်၊ ထပ်စစ်တယ် မှန်ရင် အောက်ဆင်းပြီး လုပ်စရာရှိတာ လုပ်မယ်၊ ပြီးတာနဲ့ arrow ဟာ condition ဆီ ပြန်တက်တယ်၊ အဲဒီလိုနဲ့ looping ပတ်နေတာ၊ ဘယ်အချိန် ရပ်မလဲဆိုတော့ condition စစ်တာ false ထွက်ရင် looping ထဲ က ထွက်သွားပြီး ကျန်တဲ့ လုပ်စရာရှိတာ ဆက်လုပ်မယ်။

အပေါ်က Flowchart အပိုင်းအစလေးကို အစ အဆုံး trace လိုက်ကြည့်ရအောင်။ Trace မလိုက်ခင် Flowchart ကို စာအုပ်ထဲ ချခွဲထားပါ။ ပြီးမှ trace လိုက်ပါ။ ဒါမှ မျက်စီမံလည်မှုပါ။

ပထမ စစချင်း $N=1$ ဆိုပြီး N ထဲကို 1 ထည့်လိုက်တယ်။ ပြီးတော့ $N < 99$ ဆိုတဲ့ condition ကို စစ်တော့ N တန်ဖိုးသည် ယခုလက်ရှိ 1 ဖြစ်တဲ့အတွက် 99 ထက်ငယ်တယ်၊ ဒီတော့ မှန်တယ်၊ မှန်တော့ မှန်တဲ့ဘက် အောက်ဆင်းတော့ N နဲ့ comma ကို ဆက်ထုတ်တယ်၊ 1, ဆိုပြီးထွက်မယ်၊ ပြီးရင် N ကို 2 ပေါင်းပြီး N ထဲ ပြန်ထည့်တယ်။

ယခုလက်ရှိ N 3 ဖြစ်သွားမယ်၊ update ပြီးတာနဲ့ arrow ဟာ အပေါ်ပြန်တက်ပြီး condition ဆီရောက်တယ်၊ ယခုလက်ရှိ N တန်ဖိုး 3 ဟာ 99 ထက်ငယ်တယ်။ မှန်တော့ အောက်ဆင်းလုပ်တော့ 3, ထပ်ထုတ်တော့ 1,3, ဆိုပြီး output ထွက်ပြီးသားဖြစ်မယ်။

ပြီးရင် update လုပ်ပြီး 2 တိုးတော့ N တန်ဖိုး 5 ဖြစ်မယ်၊ condition စစ်တယ်၊ N တန်ဖိုး 5 ဟာ 99 ထက်ငယ်တယ်။ ဒီတော့ မှန်တော့ 1, 3, 5, ရမယ်၊ ဒီလိုနဲ့ 99 ထက်ငယ်တဲ့ 97, ထို ထုတ်သွားမယ်။
1, 3, 5, 9, ..., 97,

ပြီးတာနဲ့ N ကို 2 ပေါင်းပြီး update လုပ်တော့ N တန်ဖိုး 99 ဖြစ်မယ်၊ update ပြီးတာနဲ့ condition ဆီ ပြန်တက်တော့ N တန်ဖိုး 99 ဟာ 99 ထက် မငယ်တော့ဘူး၊ ဒီတော့ condition မှားတယ်၊ မှားတော့ looping ကနေ ထွက်ပြီး false ဘက်ခြမ်းသွားလုပ်တော့ N ကို comma မပါပဲ ထုတ်တယ်။

1, 3, 5, 9, ..., 97, 99

ဆိုပြီး ဖြစ်သွားမယ်။ ဒီလောက်ဆိုရင် looping ဆိုတာရယ်၊ looping နှင့် condition ကွာခြားချက်ရယ် ကို သဘောပေါက်လောက်ပါပြီ။ code နဲ့ ရေးကြည့်ရင်

$N = 1$ (initialization)

Repeat while $N < 99$: (condition)

Write N , “,”

$N = N + 2$ (update)

End of Loop

Write N

ဆိုပြီး ပါမယ်။ နောက်တန်ည်း ရေးပြပါအံ့ဌယ်။

Repeat for N=1 to 97 by 2: ←

N=1, initialization N တန်ဖိုး 1 ကနေစမယ်

Write N, “,”

To 97, condition 97 ရောက်သည်ထိ လုပ်မယ်

End of Loop

By 2, update တစ်ကြိမ်ပြီး တိုင်း J ပေါင်းမယ်

Write N

ဒီလောက်ဆိုရင် looping သဘောတရား နားလည်လောက်ပါပြီ။



လေ့ကျင့်ကြည့်ရအောင်။



100, 95, 90, 85, ..., 5, 0

(0 နောက်မှာ comma မပါလို 0 ကို သပ်သပ်ဖယ်ထား၊ looping ပြီးမှ လုပ်မယ်။ ဒီဆို looping ဘယ်ကစ 100 ကစ၊ ဘယ်အထိလဲ 5 အထိ၊ ရှေ့ကိုပြန်ကြည့်တော့ 100 တို့ 95 တို့ဟာ 5 ထက်ကြီးတယ်။ ဒီတော့ ဘယ်အထိလဲဆိုရင် ≥ 5 (5 ထက်ကြီးပြီး ညီတဲ့အထိ)၊ 100 ပြီးရင် 95, 95 ပြီးရင် 90 စသည်ဖြင့်သွားတော့ တစ်ကြိမ်ပြီးတိုင် 5 နှတ်)

i =1

Repeat while i ≥ 5 :

 Write i, “,”

 i = i - 5

End of Loop

 Write i

Repeat for i=100 to 5 by -5:

 Write i, “,”

End of Loop

 Write i

 1, 4, 9, 16, 25, 100

 $1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{n}$ (n တန်ဖိုးကို user တံမှေ တောင်းပါ။ n တန်ဖိုး 5 ဆို $\frac{1}{5}$ အထိ
 n တန်ဖိုး 100 ဆို $\frac{1}{100}$ အထိ စသည်ဖြင့် လုပ်သွားမှာ ဖြစ်ပါတယ်)

Nested loop

Conditional statement လိုပါပဲ Looping မှာ လည်း nested loop ရှိပါတယ်။ looping ထဲ
looping ပြန်ပါတာ ဖြစ်ပါတယ်။ အပြင်ဘက်က looping ကို outer loop လို့ ခေါ်ပြီး၊ အတွင်းထဲက
looping ကိုတော့ inner loop လို့ ခေါ်ပါတယ်။

အလုပ်လုပ်ပုံကတော့ outer loop တစ်ကြိမ်မှာ inner loop ပြီးအောင်လုပ်၊ outer loop
နောက်တစ်ကြိမ်မှာ inner loop ပြီးအောင်လုပ် စတဲ့ ပုံစံနဲ့ အလုပ်လုပ်ပါတယ်။ ဒါကြောင့် တစ်ကြိမ်
တစ်ကြိမ် ဆိုပြီး လုပ်ချင်တဲ့ ကောင်သည် outer loop ဖြစ်ပြီး၊ အဲဒါ တစ်ကြိမ်ထဲမှာ အများကြီး
လုပ်ချင်တဲ့ ကောင်သည် inner loop ဖြစ်ပါတယ်။ နမူနာလေး ကြည့်ကြည့်ရအောင်ပါ။

No 1.

Question 1

Question 2

Question 3

Question 4

No 2.

Question 1

Question 2

Question 3

Question 4

No 3.

Question 1

Question 2

Question 3

Question 4

အခုပြထားတာလေးကို ကြည့်လိုက်ရင် No 1. မှာ question 4 ခါ၊ No 2. မှာ question 4 ခါ၊ No 3. မှာ question 4 ခု ဖြစ်ပါတယ်။ No ဆိုတာ တစ်ကြိမ်စီမှာ Question 4 ခု အလုပ်လုပ်သွားတာဖြစ်ပါတယ်။ ဒါကြောင့် တစ်ကြိမ်စီ လုပ်တဲ့ No သည် outer loop ဖြစ်ပြီး၊ Question ကတော့ inner loop ဖြစ်မှာ ဖြစ်ပါတယ်။ No အတွက် variable တစ်ခုနဲ့ Question အတွက် variable တစ်ခုလိုပါတယ်။ N နှင့် Q ဆိုပြီး အမည် ပေးလိုက်ပါမယ်။ Output ထူတ်တဲ့ အချိန်မှာလည်း No 1. နှင့် Question 1 ဆိုတဲ့ ပုံစံနဲ့ ပေါ်ချင်တာ သတိထားပါ။ တိုက်ရိုက်ပေါ်ချင် single quote or double quote တဲ့ရေး၊ တန်ဖိုးပေါ်ချင် ဒီတိုင်းရေး၊ အဲဒီနှစ်ခုကို ဆက်ပြီး ပေါ်ချင်ရင် comma နဲ့ဆက်။

1. $N = 1$
2. Repeat while $N < 4$
3. Write 'No ', N, '\n'
4. $Q = 1$
5. Repeat while $Q < 5$
6. Write 'Question ', Q, '\n'
7. $Q = Q + 1$
8. End of Loop
9. $N = N + 1$
10. End of Loop

1. Repeat for $N=1$ to 3 by 1:
2. Write 'No ', N, '\n'
3. Repeat for $Q=1$ to 4 by 1:
4. Write 'Question ', Q, '\n'
5. End of Loop
6. End of Loop

ကဲ while နဲ့ ရေးထားတာလေးကို trace လိုက်ပြပါမယ်။ trace လိုက်ပြရင် ပြောရတာ လွယ်အောင် ဘေးမှာ line no တပ်ထားပေးပါတယ်။ line no 8 က end of loop ဆိုတာ inner loop ရဲ့ အပိတ်ဖြစ်လို

inner loop ရဲ condition ကို ဖြန်တက်မှာ ဖြစ်ပါတယ်။ Line 10 ကတေသ့ outer loop ရဲ အပိတ် ဖြစ်တဲ့ အတွက်ငြောင် outer loop ရဲ condition ကို ဖြန်တက်မှာ ဖြစ်ပါတယ်။

Line No	N	Q	အလုပ်လုပ်ပုံ	Output
1	1		N ထဲကို 1 ထည့်တယ်။	
2			$N < 4 \rightarrow 1 < 4$ မှန်တေသ့ line 3 သွားမယ်။	
3			\n ဆိုတာ Enter ပါပဲ။ Write Write 'No ', N, '\n' ဆိုတေသ့ output အကွက်မှာ ပြထားတဲ့ အတိုင်း output ထွက်မယ်။	No 1.
4		1	Q ထဲကို 1 ထည့်တယ်။	
5			$Q < 5 \rightarrow 1 < 5$ ဆိုတေသ့ မှန်တယ် line 6 ကိုသွားမယ်။	
6			Write 'Question ', Q,' \n' ဆိုတေသ့ output အကွက်မှာ ပြထားတဲ့ အတိုင်း ထုတ်ပြပြီး အောက်တစ် <u>ငြောင်</u> : ဆင်းမယ်။	Question 1
7	2		$Q = Q + 1$	
8			End of loop ဆိုတာ inner loop ရဲ အပိတ် ဖြစ်တဲ့ အတွက် inner loop ရဲ condition ဖြစ်တဲ့ line 5 ကို တက်မယ်။	
5			$Q < 5 \rightarrow 2 < 5$ မှန်တေသ့ line 6 ကိုသွားမယ်	
6			Write 'Question ', Q,' \n'	Question 2
7	3		$Q = Q + 1$	
8			End of loop ဆိုတာ inner loop ရဲ အပိတ် ဖြစ်တဲ့ အတွက် inner loop ရဲ condition ဖြစ်တဲ့ line 5 ကို တက်မယ်။	
5			$Q < 5 \rightarrow 3 < 5$ မှန်တေသ့ line 6 ကိုသွားမယ်	

6		Write 'Question ', Q,'\n'	Question 3
7	4	$Q = Q + 1$	
8		End of loop ဆိုတာ inner loop ရဲ့ အပိတ် ဖြစ်တဲ့အတွက် inner loop ရဲ့ condition ဖြစ်တဲ့ line 5 ကို တက်မယ်။	
5		$Q < 5 \rightarrow 4 < 5$ မှန်တော့ line 6 ကိုသွားမယ်	
6		Write 'Question ', Q,'\n'	Question 4
7	5	$Q = Q + 1$	
8		End of loop ဆိုတာ inner loop ရဲ့ အပိတ် ဖြစ်တဲ့အတွက် inner loop ရဲ့ condition ဖြစ်တဲ့ line 5 ကို တက်မယ်။	
5		$Q < 5 \rightarrow 5 < 5$ ဆိုတော့ 5 ဟာ 5 ထက်မယ်တော့ inner loop condition မှားသွားတဲ့ အတွက် inner loop ကနေ အပြင်ထွက်မယ်။ inner loop အပြင်ဆိုတာ inner loop ရဲ့ အပိတ် line no 8 ရဲ့ အပြင်ဆိုတော့ line no 9 ကိုသွားမယ်။	
9	2	$N = N + 1$	
10		End of loop ဆိုတာ outer loop ရဲ့ အပိတ် ဖြစ်တဲ့အတွက် outer loop ရဲ့ condition ဖြစ်တဲ့ line no 2 ကို ပြန်တက်မယ်။	
2		$N < 4 \rightarrow 2 < 4$ ဆိုတော့ မှန်တယ်။ line 3 ကိုသွားမယ်။	
3		Write Write 'No ', N, '.\n'	No 2.
4	1	$Q \text{ ထဲကို } 1 \text{ ထည့်တယ်။}$	
5		$Q < 5 \rightarrow 1 < 5$ ဆိုတော့ မှန်တယ် line 6 ကိုသွားမယ်။	

6		Write 'Question ', Q,'\n' ဆိုတော့ output အကွက်မှာ ပြထားတဲ့ အတိုင်း ထုတ်ပြပြီး အောက်တစ်ကြာင်း ဆင်းမယ်။	Question 1
7	2	$Q = Q + 1$	
8		End of loop ဆိုတာ inner loop ရဲ့ အပိတ် ဖြစ်တဲ့အတွက် inner loop ရဲ့ condition ဖြစ်တဲ့ line 5 ကို တက်မယ်။	
5		$Q < 5 \rightarrow 2 < 5$ မှန်တော့ line 6 ကိုသွားမယ်	
6		Write 'Question ', Q,'\n'	Question 2
7	3	$Q = Q + 1$	
8		End of loop ဆိုတာ inner loop ရဲ့ အပိတ် ဖြစ်တဲ့အတွက် inner loop ရဲ့ condition ဖြစ်တဲ့ line 5 ကို တက်မယ်။	
5		$Q < 5 \rightarrow 3 < 5$ မှန်တော့ line 6 ကိုသွားမယ်	
6		Write 'Question ', Q,'\n'	Question 3
7	4	$Q = Q + 1$	
8		End of loop ဆိုတာ inner loop ရဲ့ အပိတ် ဖြစ်တဲ့အတွက် inner loop ရဲ့ condition ဖြစ်တဲ့ line 5 ကို တက်မယ်။	
5		$Q < 5 \rightarrow 4 < 5$ မှန်တော့ line 6 ကိုသွားမယ်	
6		Write 'Question ', Q,'\n'	Question 4
7	5	$Q = Q + 1$	
8		End of loop ဆိုတာ inner loop ရဲ့ အပိတ် ဖြစ်တဲ့အတွက် inner loop ရဲ့ condition ဖြစ်တဲ့ line 5 ကို တက်မယ်။	

5			$Q < 5 \rightarrow 5 < 5$ ဆိုတော့ 5 ဟာ 5 ထက်မယ်တော့ inner loop condition မှားသွားတဲ့ အတွက် inner loop ကနေ အပြင်ထွက်မယ်။ inner loop အပြင်ဆိုတာ inner loop ရဲ့ အပိတ် line no 8 ရဲ့ အပြင်ဆိုတော့ line no 9 ကိုသွားမယ်။	
9	3		$N = N + 1$	
10			End of loop ဆိုတာ outer loop ရဲ့ အပိတ်ဖြစ်တဲ့ အတွက် outer loop ရဲ့ condition ဖြစ်တဲ့ line no 2 ကို ပြန်တက်မယ်။	
2			$N < 4 \rightarrow 3 < 4$ ဆိုတော့ မှန်တယ်။ line 3 ကိုသွားမယ်။	
3			Write Write 'No', N, '\n'	No 3.
4		1	$Q \text{ ထဲကို } 1 \text{ ထည့်တယ်။}$	
5			$Q < 5 \rightarrow 1 < 5$ ဆိုတော့ မှန်တယ် line 6 ကိုသွားမယ်။	
6			Write 'Question', Q, '\n' ဆိုတော့ output အကွက်မှာ ပြထားတဲ့ အတိုင်း ထုတ်ပြပြီး အောက်တစ်ကြာင်း ဆင်းမယ်။	Question 1
7		2	$Q = Q + 1$	
8			End of loop ဆိုတာ inner loop ရဲ့ အပိတ်ဖြစ်တဲ့ အတွက် inner loop ရဲ့ condition ဖြစ်တဲ့ line 5 ကို ထက်မယ်။	
5			$Q < 5 \rightarrow 2 < 5$ မှန်တော့ line 6 ကိုသွားမယ်	
6			Write 'Question', Q, '\n'	Question 2
7		3	$Q = Q + 1$	

8		End of loop ဆိုတာ inner loop ရဲ့ အပိတ် ဖြစ်တဲ့အတွက် inner loop ရဲ့ condition ဖြစ်တဲ့ line 5 ကို တက်မယ်။	
5		$Q < 5 \rightarrow 3 < 5$ မှန်တော့ line 6 ကိုသွားမယ်	
6		Write 'Question ', Q,'\n'	Question 3
7	4	$Q = Q + 1$	
8		End of loop ဆိုတာ inner loop ရဲ့ အပိတ် ဖြစ်တဲ့အတွက် inner loop ရဲ့ condition ဖြစ်တဲ့ line 5 ကို တက်မယ်။	
5		$Q < 5 \rightarrow 4 < 5$ မှန်တော့ line 6 ကိုသွားမယ်	
6		Write 'Question ', Q,'\n'	Question 4
7	5	$Q = Q + 1$	
8		End of loop ဆိုတာ inner loop ရဲ့ အပိတ် ဖြစ်တဲ့အတွက် inner loop ရဲ့ condition ဖြစ်တဲ့ line 5 ကို တက်မယ်။	
5		$Q < 5 \rightarrow 5 < 5$ ဆိုတော့ 5 ဟာ 5 ထက်မငယ်တော့ inner loop condition မှားသွားတဲ့ အတွက် inner loop ကနေ အပြင်တွက်မယ်။ inner loop အပြင်ဆိုတာ inner loop ရဲ့ အပိတ် line no 8 ရဲ့ အပြင်ဆိုတော့ line no 9 ကိုသွားမယ်။	
9	4	$N = N + 1$	
10		End of loop ဆိုတာ outer loop ရဲ့ အပိတ် ဖြစ်တဲ့အတွက် outer loop ရဲ့ condition ဖြစ်တဲ့ line no 2 ကို ပြန်တက်မယ်။	
2		$N < 4 \rightarrow 4 < 4$ ဆိုတော့ မှားတယ်၊ မှားတော့ outer loop ထဲက ထွက်မယ်ဆိုတော့ outer loop အပိတ် line no 10	

		အပြင်ကို သွားတော့ ဘာမှမရှိတော့လို့ (line no 11) မရှိတော့လို့ ပြီးသွားတယ်။ output colum ကို ကြည့်ကြည့်ပါ။ အဖြေ လိုချင်တဲ့ အတိုင်း ထွက်နေတာ တွေရပါလိမ့်မယ်။	
--	--	--	--

အပေါ်က while looping ပါ စာအုပ်ထဲ ချရေးပြီးတော့ trace လိုက်တာ ဟာ
အကောင်းဆုံးနည်းပါပဲ။ ကြည့်ရင် သဘောပေါက်ကောင်း သဘောပေါက်ပါလိမ့်မယ်၊ ဒါပေမဲ့ ချရေးပြီး
trace လိုက်လိုက်ရင် သဘောပေါက်တာ ဟိုဘက်ရောက်ပြီး သဘောတရား
ကောင်းကောင်းသဘောပေါက်ပါလိမ့်မယ်။သင်ခန်းစာအားလုံးကို note စာအုပ်ထဲ မှတ်ပါ။
ထွက်စရာရှိတာတွေ အကုန်ချထွက်ပါ။



အောက်မှာ ပြထားတဲ့ ပုံစံလေး ပေါ်အောင် လေ့ကျင့်ကြည့်ကြရအောင်ပါ။

Week 1

1 2 3 4 5 6 7

Week 2

8 9 10 11 12 13 14

Week 3

15 16 17 18 19 20 21

Week 4

23 24 25 26 27 28



Chapter အနှစ်ချုပ်

- ဘယ်အခြေအနေမှာမှ ဘယ်ဟာကိုလုပ်မယ်ဆိုတာက selection or conditional statement ဖြစ်ပါတယ်။ တစ်ကြိမ်ပဲ အလုပ်လုပ်ပါတယ်။
- တူညီတဲ့ အလုပ်ကို ထပ်ခါ ထပ်ခါ လုပ်တာကို looping or iteration လို့ ခေါ်ပါတယ်။ looping မှာ while နဲ့ for ဆိုပြီး ရေးနည်း ၂ နည်းရှိပါတယ်။ ဘယ်ရေးနည်းဖြစ်ဖြစ် အပိုင်း ၃ ပိုင်းပါပါတယ်။ ဘယ်ကစမှာလဲဆိုတဲ့ initialization ရယ်၊ ဘယ်အခြေအနေထိ လုပ်မှာလဲဆိုတဲ့ condition ရယ်၊ တစ်ကြိမ်ပြီးတိုင်း ဘာလုပ်မှာလဲဆိုတဲ့ update ရယ် ပဲ ဖြစ်ပါတယ်။ looping တစ်ခုရဲ့ အသက်ဟာ update လုပ်တာ ဖြစ်တဲ့အတွက် update မမေ့ဖို့ အထူးအရေးကြီးပါတယ်။ update မေ့ရင် ဘယ်တော့မှ မရပ်တော့တဲ့ infinite loop ဖြစ်သွားနိုင်ပါတယ်။
- Selection ထဲ selection ပြန်ပါတာကို nested selection statement လို့ခေါ်ပြီး၊ looping ထဲ looping ပြန်ပါတာကို nested loop လို့ခေါ်ပါတယ်။
- Nested selection မှာ အပြင်က condition မှန်ပါမှ အတွင်းက condition ကို ထပ်စစ်ပါတယ်။ နမူနာ အနေနဲ့ ဆိုရင်တော့ စာမေးပွဲ အောင်မြင်မှ D ပေးတာမျိုးဖြစ်ပါတယ်။
- Nested loop မှာ အပြင် loop ကို outer loop လို့ခေါ်ပြီး အတွင်းထဲက ရေးတဲ့ loop ကို inner loop လို့ ခေါ်ပါတယ်။ outer loop တစ်ကြိမ်မှာ inner loop ပြီးအောင်လုပ်ပါတယ်။ ဒါကြောင့် တစ်ကြိမ်မှာ အများကြီး လုပ်တဲ့ ပုံစံဆိုရင် nested loop နဲ့ ရေးပါတယ်။ တစ်ကြိမ် လုပ်တဲ့ ကောင်က outer loop ဖြစ်ပြီး၊ အများကြီး လုပ်တဲ့ ကောင်ကတော့ inner loop ဖြစ်ပါမယ်။
- Flowchart ကတော့ ထင်သာမြင်သာအောင် ပုံနဲ့ ပြတာဖြစ်ပြီး Pseudocode တို့ algorithm တို့လိုပဲ programmer တိုင်း နားလည်ပါတယ်။

“တချို့လူတွေ ရှိတယ်လေ।

ဗိုက်ကဆာနေပြီ ဘယ်နှစ်နာရီမထိုးသေးလို့ ထမင်း မစားသေးဘူးတို့၊

စနစ်ဇယားနဲ့ စာအုပ်ကြီးသမားတွေပေါ့။

တစ်ခုခုဆို သူတို့ သတ်မှတ်ချက်တွေနဲ့ condition စစ်လို့မပြီးတော့ဘူး၊

အမူးသမားက မမူးတဲ့အချိန်ဆို လူကောင်း

မူးတဲ့အချိန်မှ အောင့်ထားသမျှ မကျေနပ်ချက်တွေကို ထပ်ခါထပ်ခါ ပြောရစ်တော့တာပဲ၊

Looping သိပ်ကြိုက်တဲ့သူတွေပေါ့”

အခန်း (၈)

Efficiency and Basic Algorithms

Efficiency of Algorithm

“A good algorithm is correct, but a great algorithm is both correct and efficient.”

ကောင်းမွန်တဲ့ algorithm တစ်ခုဟာ မှန်ကန်မှုရှိတယ်၊ ပိုပြီးတကယ်ကောင်းတဲ့ algorithm က မှန်လည်းမှန်တယ်၊ efficient လည်းဖြစ်တယ်။ Efficient ဖြစ်ဖို့ဆိုရင် “minimum use of resources” ဖြစ်ရပါမယ်။ time တို့၊ memory spaceတို့ စတဲ့ resourcesတွေကို အနည်းဆုံး အသုံးပြုရမှာ ဖြစ်ပါတယ်။

ဆိုကြပါစို့ သချိုာပုစ္စာတစ်ပုဒ်ကို ကျောင်းသားနှစ်ယောက်တွက်တယ်။ နှစ်ယောက်လုံးဟာ မှန်တယ်၊ ဒါပေမဲ့ တစ်ယောက်က 15 min နဲ့ ပြီးပြီးတော့ ကျွန်ုတစ်ယောက်က 25 min ကြာမှုပြီးတယ်။ ဘယ်ကလေး performance ကောင်းသလဲ၊ မြန်သလဲဆိုရင် အချိန်အနည်းဆုံး (အချိန်တို့တို့ အတွင်း) ပြီးတဲ့ 15 min ပဲ ကြာတဲ့ ကျောင်းသားက ပိုမြန်တာပေါ့။ ဒါကြောင့် time ယူတာနည်းလေ မြန်လေ ဖြစ်ပါတယ်။

ကစားလို့ ကောင်းတာချင်းအတူတူ game တစ်ခုက လေးတယ်၊ memory များတဲ့ စက်မှာမှ အလုပ်လုပ်တယ်၊ စက်ရွေးတယ်ပေါ့၊ ကျွန်ု game ကတော့ စက်သိပ်မရွေးဘူး၊ ပေါ့တယ်၊ memory အများကြီး မရှိတဲ့ စက်မှာလဲ ကစားလို့ရတယ် ဆိုရင် ဘယ်တစ်ခု ပိုကောင်းသလဲ။ ရှင်းနေတာ ပါပဲ။ memory နည်းနည်းနဲ့ စက်တော်တော်များများမှာ ကစားလို့ရတဲ့ ဒုတိယ game က ပိုကောင်းတာပေါ့။ ဒါကြောင့် memory အသုံးပြုမှု နည်းလေလေ ပေါ့လေလေ စက်တော်တော်များများမှာ အလုပ်လုပ်လေ ဖြစ်ပါတယ်။

ဒါကြားနှင့် algorithm တွက် ဘယ် algorithm က ပိုကောင်းသလဲ ဆိုရင် 1. **Correctness**, 2. **Time complexity**, 3. **Space complexity** စသည့် အချက် ၃ ချက်ဖြင့် နိုင်းယူဉ်လေ့ရှိပါတယ်။ အဲဒီထဲကမှ time complexity ကို တွက်ချက်တဲ့အခါမှာ တွက်နည်း ၃ မျိုးရှိပါတယ်။

- **Worst Case** – Worst case ဆိုတဲ့အတိုင်း အဆိုးဆုံးအခြေအနေအတွက် အများဆုံး ယူနိုင်တဲ့ ကြာချိန်(maximum time)ကို တိုင်းတာတာဖြစ်ပါတယ်။ Big O notation (O) နဲ့ ကိုယ်စားပြုပါတယ်။
- **Average Case** – Average case ဆိုတဲ့အတိုင်း အများဆုံးကြာချိန်နဲ့ အနည်းဆုံးကြာချိန် တို့ပေါ်မှုတည်ပြီး မျမှ်းမျှကြာချိန်ကို တွက်ချက်တာဖြစ်ပါတယ်။ Theta Notation (Θ) ကတော့ average case အတွက် အများဆုံး ကြာချိန်နှင့် အနည်းဆုံးကြာချိန်ကို နိုင်းယူဉ်ပြုဖို့အတွက် သုံးပါတယ်။
- **Best Case** – Worst case နဲ့ ဆန့်ကျင်ဘက် အနည်းဆုံးကြာမည့်အချိန်ကို တိုင်းတာ တာဖြစ်ပါတယ်။ Omega Notation (Ω) နဲ့ ကိုယ်စားပြုပါတယ်။

Prime Number

အလွယ်ပြောရမယ်ဆိုရင်တော့ မိမိကိုယ်တိုင် နှင့် ၁ ကဆွဲပြီး တစ်ခြားကိန်းဖြင့် စားလို့ မပြတ်သောကိန်းကို Prime Number (သူ့ခုကိန်း) လို့ ခေါ်ပါတယ်။ Prime Number ဟာ 2 ကနေစပါတယ်။ ၁ တော့ ကိန်းကဏ္ဍားတိုင်းကို စားပြတ်နေတာဖြစ်တဲ့အတွက် ထည့်မပြောတော့ပါဘူး။ ဒီတော့ ကဏ္ဍား တစ်လုံးဟာ prime number ဟုတ်မဟုတ် သိချင်ရင် 2 နဲ့ စပြီး စားကြည့်ရပါမယ်။ ပြတ်သည် အထိ စားကြည့်ပါမယ်။ စားလို့ ပြတ်သွားတဲ့ အချိန်မှ စားတဲ့ကဏ္ဍား နဲ့ အစားခံရတဲ့ ကဏ္ဍားမတူဘူးဆိုရင် မိမိကိုယ်တိုင်နဲ့သာ စားပြတ်တာဖြစ်လို့ prime number ဖြစ်ပြီး၊ စားတဲ့ကဏ္ဍား နဲ့ အစားခံရတဲ့ ကဏ္ဍားမတူဘူးဆိုရင် တစ်ခြားကိန်းနဲ့ စားပြတ်တာဖြစ်လို့ prime number မဟုတ်ပါဘူး။

ဤပမာ 5 ကို prime number ဟုတ်မဟုတ် စစ်ကြည့်မယ် ဆိုရင် 2 ကနေ စပီး စားကြည့်တယ် မပြတ်ဘူး၊ ဒီတော့ စားတဲ့ကိန်းကို 1 တိုးတိုးပြီးစားကြည့်မယ်။ 3 နဲ့ ထပ်ပြီးစားကြည့်တယ် မပြတ်ဘူး၊ 4 နဲ့ စားကြည့်တယ် မပြတ်ဘူး၊ 5 နဲ့ စားကြည့်တယ်တယ်၊ ပြတ်တယ်၊ စားပြတ်တဲ့အတွက် အစားခံရတဲ့ကိန်း နဲ့ စားတဲ့ကိန်း နှစ်ခုကို တိုက်ကြည့်လိုက်တော့ 5 ချည်း ဖြစ်နေတဲ့အတွက် တူနေတယ်။ ဒီတော့ မိမိကိုယ်တိုင်နဲ့ပဲ စားပြတ်တာဖြစ်တဲ့အတွက် 5 သည် prime number ဖြစ်တယ်။

ဤပမာ 8 ကို prime number ဟုတ်မဟုတ် စစ်ကြည့်မယ် ဆိုရင် 2 ဖြင့် စပီး စားကြည့်ပါမယ်၊ စားလို့ ပြတ်တဲ့အတွက် အစားခံရတဲ့ကိန်းနှင့် စားတဲ့ကိန်းကို တိုက်ကြည့်တဲ့အခါ 8 ရယ် 2 ရယ် ဖြစ်တဲ့အတွက် တစ်ခြားကိန်းဖြင့် စားပြတ်နေတာဖြစ်လို့ 8 သည် prime number မဟုတ်ပါ။

Algorithm: Prime Number (1)

1. isPrime(num)
 2. If num<=1 then
 3. Return false
 4. d=2
 5. Repeat while (num mod d) != 0:
 6. d = d+1
 7. End of loop
 8. If num == d then:
 9. Return false
 10. Else:
 11. Return true
-

Trace လိုက်မပြခင် Algorithm ကို နည်းနည်းလောက် အရင် ရှင်းပြပါမယ်။ Prime number ဟုတ်မဟုတ် user က စစ်ချင်တဲ့ ကိန်းကို num ဆိုပြီး parameter လက်ခံလိုက်ပါတယ်။

ယုံကြည်စိတ်ချရမှုကို ရရှိဖို့ဆိုရင် algorithm တစ်ခုဟာ ဖြစ်နိုင်တဲ့ အမှားတွေကို ကြိုတင် စစ်ဆေးပြီးကာကွယ် ထားပေးဖို့လိုပါတယ်။ ဥပမာ အသက်ဆိုရင် ဂဏန်းမှန်ပေမဲ့ ဂဏန်းတိုင်း မမှန်ပါဘူး။ အနုတ်တို့၊ 130 ကျော်တာတို့ဆို မှားပါတယ်။ ဒါတွေကို သေချာ စစ်ထုတ်ဖို့လိုပါတယ်။ မဟုတ်ပဲ ဘာရှိက်ရှိက် လက်ခံနေတယ်ဆိုရင်တော့ အသုံးပြုတဲ့ user ဟာ ဘယ်လိုမှ ယုံကြည်မှုရှိမှာ မဟုတ်ပါဘူး။ ဒါက input error ကိုနှမ်နာဖြောပြတာပါ။ တွက်လိုက်မှ error တပ်တယ်၊ အဖြေထွက်မလာဘူးတို့ ဆိုရင်လည်း user က ယုံကြည်မှုရှိမှာ မဟုတ်ပါဘူး။ ဒါကြောင့် အဲဒီအရာတွေကိုလည်း ကာကွယ်ထားဖို့ လိုပါတယ်။

ဒီနေရာမှာ prime number သည် 2 ကစတယ်။ 2 ထက်ငယ်တဲ့ ဂဏန်းမှန်သမျှဟာ prime number မဟုတ်ဘူး။ အဲဒါလေးကို function စစ်ဆေး ကတည်းက စစ်ထားပေးတယ်။ ဒီတော့ 2 အောက်ငယ်တာတွေ ထည့်ခဲ့ရင် တန်းပြီး false return ပြန်မှာကြောင့် အောက်က အလုပ်တွေ ဆက်ပြီးလုပ်စရာမလိုတော့ပါဘူး။ return ပြန်တယ်ဆိုတာ function ထဲကနေထွက်၊ လုမ်းခေါ်တဲ့ နေရာ(function call) ကိုသွားပြီး အဖြေပြန်ပေးတာဖြစ်ပါတယ်။ ဒါကြောင့် false return ပြန်လိုက်တာနဲ့ function ထဲကနေ ထွက်သွားမှာ ဖြစ်လို့ အောက်က အလုပ်တွေတာ ဖြစ်ပါတယ်။ ပြီးတော့ ဒီနေရာမှာ 2 ထက်ငယ်သလား စစ်တဲ့ conditional or selection statement သာမပါခဲ့ရင် infinite loop ဖြစ်သွားမှာပါ။ ဥပမာ ဝင်လာတာက 1 ဆိုပါတော့။ num တန်ဖိုး 1 ဖြစ်မယ်။ selection စစ်ထားတာ မပါခဲ့ဘူးဆိုရင် looping ကနေပြီးအလုပ်လုပ်မှာ ဖြစ်ပါတယ်။ 2 ကနေ စပြီး မပြတ်မချင်းစားမယ်။ 1 ကို 2 နဲ့ စားမပြတ်ဘူး၊ ဒီတော့ d ကို 1 တိုးပြီး 3 နဲ့စားတယ် မပြတ်ဘူး၊ ဒီလိုနဲ့ တိုးသွားလိုက်တာ 1ကို သူ့ထက်ကြီးတာတွေနဲ့ချည်း သွားသွား စားနေတော့ ဘာနဲ့မှ စားလို့မပြတ်ဘူး။ ဒီတော့ စားပြတ်မှ ရပ်မှာဖြစ်တဲ့ looping က မရပ်တော့ပဲ infinite loop ဖြစ်သွားမှာ ဖြစ်ပါတယ်။ ဒီတော့ algorithm တစ်ခုဟာ ယုံကြည်စိတ်ချရဖို့ ဆိုရင် ဖြစ်နိုင်တဲ့ error တွေအားလုံး စစ်ထားပေးဖို့ ကာကွယ်ထားဖို့ လိုပါတယ်။

Selection စစ်ထားတာကို နားလည်လောက်ပြီဆိုတော့ အောက်က line 4 ကနေ ဆက်ပြီးရှင်းပါမယ်။ ဝင်လာတဲ့ num ကို 2 ကနေ စပြီးစားမှာ ဖြစ်တဲ့ အတွက် စားဖို့အတွက် အသုံးပြုမည့် variable d ထဲကို 2 လိုက်ပါတယ်။ အဲဒါဟာ **initialization** ပါ။

ပြီးတာနဲ့ မပြတ်မချင်း ထပ်ခါ ထပ်ခါ စားမှာ ဖြစ်တဲ့ အတွက် looping ပတ်ပါတယ်၊ ဘယ်အထိလဲ ဆိုတော့ မပြတ်မချင်း (num ကို d ဖြန့်စားတာ အကြွင်း 0 နှင့် မည်မချင်း) ပတ်ပါမယ်။ num mod d != 0 ဟာ **condition** ဖြစ်ပါတယ်။ မပြတ်ရင် စားတဲ့ d ကို ၁ တိုးတိုးပြီး စားမှာ ဖြစ်လို့ d = d + 1 ဆိုတာ update ဖြစ်ပါတယ်။ ဒီမှာက စားလို့ပြတ်တော့မှ prime ဟုတ်မဟုတ် စစ်ချင်တာဖြစ်လို့ မပြတ်ရင် ဘာမှ မလုပ်ချင်တဲ့ အတွက် while looping ထဲမှာ လုပ်ချင်တဲ့ body မပါ ပါဘူး၊ update ပဲပါ ပါတယ်။ စားပြတ်တာနဲ့ looping ထဲက ထွက်ပါမယ်။

Looping ကနေ ထွက်တဲ့ အချိန်မှာ အစားခံရတဲ့ ကောင်နဲ့ စားတဲ့ ကောင်နဲ့ တူမတူ စစ်ပါတယ်၊ တူရင် prime number ဖြစ်တဲ့ အတွက် true return ပြန်ပါမယ်။ မတူရင် prime number မဟုတ်လို့ false return ပြန်ပါမယ်။ ဒီလောက်ဆို algorithm ကို နားလည်လောက်ပြီ ဖြစ်လို့ trace လိုက်ကြရအောင်။ ထုံးစံအတိုင်း trace လိုက်မယ်ဆို မှတ်စုစာအုပ်ရယ်၊ ရေးစရာရယ်ယူထားပြီး တစ်ခါတည်း လိုက်လုပ်ပါနော်။

 Ans = isPrime(0)

Line No	num	d	အလုပ်လုပ်ပုံ	Result
1	0			
2			num < 2 စစ်တော့ 0 < 2 ဖြစ်လို့ မှန်ပါတယ်။ ဒီတော့ false return ပြန်ပါမယ်။	Return false

Return ပြန်လိုက်တဲ့ false ဟာ function ထဲကနေထွက်ပြီး function call ခေါ်ထားတဲ့ နေရာကို ပြန်သွားပြီး function call က assign(=) လုပ်ထားတဲ့ Ans ထဲ ရောက်သွားမှာ ဖြစ်ပါတယ်။

 Ans = isPrime(5)

Line	num	D	အလုပ်လုပ်ပုံ	Result
1	5			
2			num < 2 စစ်တော့ 5 < 2 ဖြစ်လို့ မှားတယ်၊ ဒီတော့ အောက်က return ကိုအလုပ်မလုပ်တော့ဘူး။	
4		2	d = 2	
5			Num mod d -> 5 mod 2 = 3 != 0 ဆိုတော့မှန်တယ်၊ မှန်တော့ line 5 ကို သွားမယ်။	
6		3	d = d +1	
7			End of loop ဆိုတော့ condition ရှိရှာ line 4 ကိုပြန်တက်မယ်။	
5			Num mod d -> 5 mod 3 = 2 != 0 ဆိုတော့မှန်တယ်၊ မှန်တော့ line 5 ကို သွားမယ်။	
6		4	d = d +1	
7			End of loop ဆိုတော့ condition ရှိရှာ line 4 ကိုပြန်တက်မယ်။	
5			Num mod d -> 5 mod 4 = 1 != 0 ဆိုတော့မှန်တယ်၊ မှန်တော့ line 5 ကို သွားမယ်။	
6		5	d = d +1	
7			End of loop ဆိုတော့ condition ရှိရှာ line 4 ကိုပြန်တက်မယ်။	

5		<p>Num mod d -> 5 mod 3 = 0 != 0</p> <p>0 != 0 ဆိုတော့မှားတယ်၊ looping ကနေထွက်မယ် ဆိုတော့</p> <p>end of loop အပြင်ဘက် line 8 ကိုသွားမယ်။</p>	
8		<p>Num == d စစ်တော့ 5 == 5 မှန်တဲ့အတွက် return true ဆိုပြီး</p> <p>function ထွက်ပြီး call ထားတဲ့နေရာ ရောက်မယ်။</p>	Return true

Return ပြန်လိုက်တဲ့ true ဟာ function ထဲကနေထွက်ပြီး function call ခေါ်ထားတဲ့ နေရာကို
ပြန်သွားပြီး function call က assign(=) လုပ်ထားတဲ့ Ans ထဲ ရောက်သွားမှာဖြစ်ပါတယ်။



Ans = isPrime(6)

Line	num	D	အလုပ်လုပ်ပုံ	Result
1	5			
2			<p>num < 2 စစ်တော့ 6 < 2 ဖြစ်လို့ မှားတယ်၊ ဒီတော့</p> <p>အောက်က return ကိုအလုပ်မလုပ်တော့ဘူး။</p>	
4		2	d = 2	
5			<p>Num mod d -> 6 mod 2 = 0 != 0 မှားတော့ looping</p> <p>ထဲက ထွက်မှာဖြစ်လို့ end of loop အပြင်ဘက် line 8</p> <p>ကို သွားမယ်။</p>	
8			<p>Num == d စစ်တော့ 6 == 2 ဆိုတော့ မှားတဲ့အတွက်</p> <p>Else ကို သွားအလုပ်လုပ်တော့ return false ဆိုပြီး</p> <p>function ထဲကနေ ထွက်သွားမယ်။</p>	Return false

Return ပြန်လိုက်တဲ့ false ဟာ function ထဲကနေတွက်ပြီး function call ခေါ်ထားတဲ့ နေရာကို
ပြန်သွားပြီး function call က assign(=) လုပ်ထားတဲ့ Ans ထဲ ရောက်သွားမှာဖြစ်ပါတယ်။

Prime number မှာ 2 ထက်ငယ်တာရယ်၊ 2 ထက်ကြီးပြီး prime မဟုတ်တာရယ်၊ prime
ဟုတ်တာရယ်ဆိုပြီး case 3 ခုရှိပါတယ်၊ ဒီမှာ ၃ ခုလုံး trace လိုက်ပြထားပါတယ်။ testing လုပ်ရင်
case အားလုံး cover ဖြစ်အောင် စစ်ပေးရပါတယ်။ ဥပမာ မြန်မာစာနဲ့ ပတ်သက်တာကို
လုပ်တယ်ဆိုရင် မြန်မာစာ စာအုပ်ထဲက data နဲ့တင် စစ်ဆေးပေးလို့မရပါဘူး။ မြန်မာစာ စာအုပ်ဆိုတာ
အရေးအသား အဖြတ်အတောက်မှန်ပြီးသား ဖြစ်ပါတယ်။ ဒါကြောင့် အရေးအသားအဖြတ်အတောက်
မမှန်တတ်တဲ့ Facebook ပေါ်က post တွေတို့ messenger ထဲက စာသားတွေတို့နဲ့ ပါ စစ်ဆေးပေးဖို့
လိုပါတယ်။

Algorithm: Prime Number (2)

1. isPrime(num)
2. If num<=1 then:
3. Return false
4. j=2
5. Repeat while j<[(num/2)+1] and (num mod j) != 0:
6. j=j+1
7. End of loop
8. If j == num then:
9. Return true
10. ELSE:
11. Return false

Prime Number ဟုတ်၊ မဟုတ် စစ်တဲ့ algorithm ကို နောက်တစ်မျိုးဖြစ်ပါတယ်။
ကွာခြားချက်လေး လေ့လာကြည့်ကြရအောင်ပါ။

စားဖို့အတွက် အသုံးပြုမည့် variable name ကို j လိုပေးထားတယ်။ j ထက်စာရင် အပေါ်က algorithm မှာလို divisor အတိုခေါက် d လိုပေးတာ ပိုကောင်းပါတယ်။ **variable name** ကို အဓိပ္ပာယ်ရှိတာ ပေးတာဟာ ဒီ **variable** ကတေသာ ဘာအတွက်ဆိုတာမျိုး အထူးအထွေမှတ်သားစရာ ခေါင်းထဲထည့်ထားစရာ မလိုတဲ့အတွက် ကောင်းတဲ့ အလေ့အကျင့်တစ်ခုဖြစ်ပါတယ်။ programmer တိုင်း လိုက်ကိုလိုက်နာသင့်တဲ့ အလေ့အကျင့်ကောင်းတွေထဲက တစ်ခု ဖြစ်ပါတယ်။

အခိုက ကွာခြားချက်က ဘယ်နေရာမှာလဲဆိုရင် line number 5, while looping ရဲ condition စစ်ထားတဲ့နေရာမှာ ဖြစ်ပါတယ်။ Algorithm1 က စားလို့ မပြတ်မချင်း looping ပတ်မယ်လို့ စစ်ထားတယ်၊ Algorithm 2 က စားမည့် j တန်ဖိုးဟာ ဝင်လာတဲ့ကိန်းရဲ့ တစ်ဝက် + ၁ ထက် ငယ်မယ် (တန်ညီးအားဖြင့်ပြောရမယ်ဆိုရင် ဝင်လာတဲ့ကိန်းရဲ့ တစ်ဝက်ကိန်းအထိ အလုပ်လုပ်မယ်)၊ စားလို့လည်း မပြတ်ဘူး ဆိုတဲ့ အချင်နှစ်ချက်လုံး မှန်နေသရွှေ့ ဆက်ပြီး looping ပတ်နေမှာ ဖြစ်ပါတယ်။ and ဖြင့် ဆက်ထားလို့ အဲဒီ condition နှစ်ခုထဲ က တစ်ခုမှားတာနဲ့ ထွက်သွားမှာ ဖြစ်ပါတယ်။

ကိန်းကဏ္ဍးတစ်ခုဟာ သူရဲ့ တစ်ဝက်ကိန်းထိမှ စားလို့မပြတ်ဘူးဆိုရင် သေချာပေါက် အဲဒီ ကဏ္ဍးဟာ သူကိုယ်သူနဲ့ပဲ စားပြတ်မှာဖြစ်တယ်၊ ဒါကြောင့် အဲဒီကိန်းဟာ prime number ဖြစ်တယ် ဆိုတာရှိပါတယ်။ ဒါကြောင့် ဒုတိယ algorithm ဟာ အဲဒါကို အသုံးချထားတာဖြစ်လို့ သူဟာ အများဆုံး ဝင်လာတဲ့ကိန်းရဲ့ တစ်ဝက်ထိ ပဲ ထွက်ချက်မှာ ဖြစ်လို့ သူက အလုပ်လုပ်ရတဲ့ time သက်သာမှာ သေချာပေါက်ပါပဲ။ အပေါ်မှာပြောခဲ့တဲ့ time complexity case တွေနဲ့ ပြောမယ်ဆိုရင်

- Best case: တစ်ကြိမ်တွက်ရုံးနဲ့ အဖြေထွက်တဲ့ $O(2)$ ထက်ငယ်လို့ if မှာတင် တန်းပြီးအဖြေထွက်) နဲ့ $6(2 \times 1\text{ပြီး စားတာနဲ့ တန်းပြတ်ပြီး အဖြေထွက်})$ လိုမျိုး ကိန်းတွေအတွက်ဆိုရင် algorithm နှစ်ခုလုံးဟာ ကွာခြားချက် သိပ်မရှိပါဘူး။ 6 လို့ looping တစ်ကြိမ်ပတ်ရတဲ့ အခြေအနေမှာ ဒုတိယ algorithm ရဲ့ while မှာ condition

2 ခုကို and နဲ့ဆက်ထားလို့ပထမ algorithm ရဲ့ while condition ထက် condition

တစ်ခု ပိုပြီး မဆိုသလောက် ပိုလုပ်လိုက်ရတာလေးပဲ ရှိပါတယ်။

- မိမိကိုယ်တိုင်နဲ့သာ စားပြတ်တဲ့ ကိန်းကဏ္ဍးတွေမှာတော့ ကိန်းကဏ္ဍးတန်ဖိုးကြီးလေလေ algorithm နှစ်ခုရဲ့ ကွာခြားချက်က ကြီးလေလေ ဖြစ်ပါတယ်။ ဥပမာ 997 ဆိုရင် ပထမ algorithm က 2 ကနေ 997 ထိ စားကြည့်မှာဖြစ်ပြီး၊ ဒုတိယ algorithm ကတော့ 997 ရဲ့ တစ်ဝက် 498 ထိပဲ တွက်မှာမူးတစ်ဝက်တိတိ တွက်ရတာ သက်သာပါတယ်။ အချိန်ကုန်မှု time တစ်ဝက်တိတိ သက်သာတယ်လို့ ပြောလို့ရပါတယ်။ ဒါကြောင့် worst case မှာ ဒုတိယ algorithm က ပိုပြီး ကောင်းတာ သိသာထင်ရှားတာဖြစ်ပါတယ်။

Algorithm တစ်ခုနဲ့ တစ်ခု complexity တွေကို နိုင်းယဉ်ကြတဲ့ အခါမှာ best case ထက် worst case မှာ တန်းပြီး သိသာသာ အဖြေပေါ်လေ့ရှိပါတယ်။ ဒါကြောင့် algorithm တွေကို worst case နဲ့ နိုင်းယဉ်လေ့ ရှိကြပါတယ်။

နောက်ပြီး ဒီမှာ algorithm2 က ပိုပြီး ကောင်းတယ်ဆိုတာ သူဟာ ဝင်လာတဲ့ ကိန်းရဲ့ တစ်ဝက်ကိန်းအထိ စားကြည့်လို့မှု မပြတ်ရင် ဝင်လာတဲ့ ကိန်းဟာ prime number ဖြစ်တယ်ဆိုတဲ့ အချက် တန်ည်းအားဖြင့် သူရေးသားရမည့် prime number အကြောင်းကို ပိုသိလိုဖြစ်ပါတယ်။ ဒါကြောင့် programmer တစ်ယောက်ဟာ code မရေးသားခင် မိမိရေးသားပေးမည့် အကြောင်းအရာကို အချိန်ယူပြီး ကောင်းစွာသိအောင် ကြိုးစားဖို့လိုပါတယ်။ ကောင်းကောင်းသိလေ ကောင်းတဲ့ code တစ်ခု တွက်လေ ဖြစ်ပါမယ်။

အတိရေးနည်း

If num == d then: Return false Else: Return true	Return num==d
---	---------------

အပေါ်က ပြောသူမှာ ဖြစ်ထားတဲ့ J ခု ဟာ အတူတူပဲဖြစ်ပါတယ်။ ပထမ တစ်ခုကိုတော့ ရှင်းပြုပြီးသား နားလည်ပြီးသား ဖြစ်ပါလိမ့်မယ်၊ **Return num==d** ဆိုတဲ့ ဒုတိယတစ်ခုပဲကို ရှင်းပြပါမယ်။

- num တန်ဖိုးဟာ 6 ဆိုရင် d တန်ဖိုး 2 နဲ့ တင် စားပြတ်တယ်။ ဒါတော့ return 6==2 ဆိုပြီး ဖြစ်မယ်၊ 6==2 က မညြိတော့ false ဖြစ်တော့ return false ဖြစ်သွားတယ်။
- num တန်ဖိုးဟာ 5 ဆိုရင် d တန်ဖိုး 5 နဲ့ စားမှ ပြတ်တော့ return 5==5 ဆိုပြီး ဖြစ်မယ်၊ 5==5 က ညြိတော့ true ဖြစ်တော့ return true ဖြစ်သွားတယ်။

အလုပ်လုပ် ပုံက အတူတူပဲ ဆိုတာ သဘောပေါက်လောက်ပါပြီ။ အလုပ်လုပ်ပုံတူရင် အတိုရေးနည်းက ရေးရတာ သက်သာတယ်၊ မိုက်တယ်။ ဒါကြောင့် များများလုပ်ရင် အလေ့အကျင့် ဖြစ်တယ် ဆိုတဲ့အတိုင်း အလေ့ အကျင့် ဖြစ်သွားရအောင် အတိုရေးနည်းလေးတွေကို တခုတ်တရ ရေးဖို့လိုပါတယ်။

ဖြစ်နိုင်သမျှ Error တွေကို စစ်ထားပေးမယ်။

ဖြစ်နိုင်သမျှ တပ်လာနိုင်တဲ့ error ဆိုတာကို မြင်သာထင်သာ ရှိအောင် $ax^2 + bx + c = 0$ ဆိုတဲ့ quadratic equation ကို စဉ်းစားကြည့်ကြရအောင်ပါ။ အဲဒီ equation ကနေ x တန်ဖိုးကို တွက်ချင်ရင်

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

ဆိုပြီး တွက်ရပါတယ်။ အဲဒီမှာ $\sqrt{b^2 - 4ac}$ ဆိုတာလေးသတိထားရပါမယ်။ square root (1/2) ထဲမှာ အနှစ်ဖြစ်လို့မရတာ ရှုံးမှာ ပြောခဲ့ပြီးပါပြီ။ $b^2 - 4ac$ က အနှစ်ထွက်နေရင် square root ရှာလိုက်တဲ့အချိန် error တပ်မှာဖြစ်ပါတယ်။ ဒါကြောင့် $b^2 - 4ac$ ကို အရင်တွက်ပြီး condition စစ်ပေးရမှာ ဖြစ်ပါတယ်။ $b^2 - 4ac$ တန်ဖိုးဟာ

1. အနှစ်ဖြစ်ခဲ့ရင် အဖြေမရှိတဲ့ no solution ပုစ္စာဖြစ်ပါတယ်။
2. 0 ထွက်ခဲ့ရင် 0 ကို square root ရှာရင် 0 ပဲထွက်မှာဖြစ်လို့ ± တွက်စရာမလိုတော့ပဲ $-b/2a$ ဆိုပြီး အဖြေတစ်ခုပဲထွက်မှာ ဖြစ်ပါတယ်။
3. >0 ဆိုရင်တော့ square root ရှာပါမယ်။ ပြီးမှ $x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$ ရယ်၊ $x = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$ ဆိုပြီး အဖြေနှစ်ခု ထုတ်ပြရမှာ ဖြစ်ပါတယ်။

quadraticEquation(a, b, c)

1. Set Temp = $b * b - 4 * a * c$
2. If Temp ≤ 0 then:
 3. Write "No Solution"
 4. Else if Temp == 0 then:
 5. Set ans = $-b / (2 * a)$
 6. Write "One Solution :", ans
 7. Else:
 8. Temp = $\sqrt{\text{Temp}}$
 9. Ans1 = $(-b + \text{Temp}) / (2 * a)$
 10. Ans2 = $(-b - \text{Temp}) / (2 * a)$
 11. Write "Two Solutions :", Ans1, " and ", Ans2

Summation Algorithm

User ထံမှ ကိန်းကဏ္ဍာ တစ်လုံးလက်ခံပါမယ်၊ အဲဒီကဏ္ဍာကို n ထဲ ထည့်ပါမယ်။ ပြီးရင် 1 ကနေ n ထိ ပေါင်းလိုက်တဲ့ အဖြောက် တွက်မှာ ဖြစ်ပါတယ်။

Algorithm: Summation (1)

1. summation(num)
2. total = 0
3. n = 1
4. Repeat while $n <= \text{num}$:
5. total = total + n
6. n = n + 1
7. End of loop
8. return total

Trace အရင် လိုက်ကြည့်လိုက်ရအောင်။ ထိုးစံအတိုင်း စာအုပ်ရယ် ခဲတံရယ် အသင့်ပြင်ပါ။

total = Summation (5)

Line	num	total	n	အလုပ်လုပ်ပုံ
1	5			
2		0		Total = 0 (ပေါင်းမယ်ဆိုရင် လာပေါင်းတဲ့ ကဏ္ဍာ ပြန်ပြီးရဖို့ အဖြောက်မည့် variable ထဲကို 0 initialize လုပ်ပေးထားတာ ဖြစ်ပါတယ်။ မြောက်မယ်ဆိုရင် လာမြောက်တဲ့ ကဏ္ဍာ ပြန်ရဖို့ 1 initialize လုပ်ပေးရပါတယ်။)
3			1	n=1
4				$n <= \text{num}$, $1 <= 5$ မှန်တယ်၊ မှန်တော့ line 5 ကိုသွားမယ်

5		1		Total = total + n, total = 0 +1, total =1
6		2		n = n+1 (update)
7				End of loop ဖြစ်တာကြောင့် line 4 condition ကို ပြန်တက်မယ်
4				n <=num, 2<=5 မှန်တယ်၊ မှန်တော့ line 5 ကိုသွားမယ်
5		3		Total = total + n, total = 1 +2, total =3
6		3		n = n+1 (update)
7				End of loop ဖြစ်တာကြောင့် line 4 condition ကို ပြန်တက်မယ်
4				3 <=num, 3<=5 မှန်တယ်၊ မှန်တော့ line 5 ကိုသွားမယ်
5		6		Total = total + n, total = 3 +3, total =6
6		4		n = n+1 (update)
7				End of loop ဖြစ်တာကြောင့် line 4 condition ကို ပြန်တက်မယ်
4				n <=num, 4<=5 မှန်တယ်၊ မှန်တော့ line 5 ကိုသွားမယ်
5		10		Total = total + n, total = 6 +4, total =10
6		5		n = n+1 (update)
7				End of loop ဖြစ်တာကြောင့် line 4 condition ကို ပြန်တက်မယ်
4				n <= num, 1<=5 မှန်တယ်၊ မှန်တော့ line 5 ကိုသွားမယ်
5		15		Total = total + n, total = 10 +5, total =15
6		6		n = n+1 (update)
7				End of loop ဖြစ်တာကြောင့် line 4 condition ကို ပြန်တက်မယ်
4				n <= num, 6<=5 မှားတယ်၊ မှားတော့ looping ထဲက ထွက်မှာဖြစ်လို့ looping အပြင်ဘက် line 8 ကိုသွားမယ်

8			Return total ဆိတ္တအတွက် return 15 ဖြစ်ပြီး function ထဲက ထွက်ပြီး call တဲ့ နေရာကို အဖြေဖြန့်ပေးမှာဖြစ်ပါတယ်။
---	--	--	--

Return ပြန်လိုက်တဲ့ တန်ဖိုးဟာ call မှာ ဖမ်းထားတဲ့ total ထဲဝင်သွားမှာ ဖြစ်ပါတယ်။ ဒါမှာ လုမ်းချေတဲ့ နေရာက ဂဏန်း 5 ထည့်ပေးလိုက်လို့ trace လိုက်တဲ့ ပေါ်ယေားမှာ 5 ကြိမ် ၁၀၉ ပတ်သွားတာ၊ 5 ကြိမ်တွက်သွားရတာ ဖြစ်ပါတယ်၊ 100 ထည့်ရင် အကြိမ် 100 ၁၀၉ ပတ်ပြီး အကြိမ် 100 တွက်ရမှာ ဖြစ်ပါတယ်။

Algorithm: Summation (2)

1. summation(num)
2. total = num * (num+1) /2
3. return total

total = summation(5)

Line	Total	Num	အလုပ်လုပ်ပုံ
1		5	
2			total = num * (num+1) /1, total = 5 * 6 /2 = 15
3			Return 15

ကဲ ဘာ looping မှ ပတ်စရာမလိုပဲ line 2 တစ်ကြောင်း တွက်လိုက်တာနဲ့ အဖြေတန်းထွက်သွားပါပြီ။ ဒီတော့ ဘယ် algorithm က မြန်မလဲ run time သက်သာမလဲ

အထူးပြောစရာမလိုဘူးထင်ပါတယ်။ ဒါဟာ သချို့ formula ကို အသုံးချတဲ့ အကျိုးကျေးဇူးပါပဲ။ ဒါကြောင့် သချို့ formula တွေ သိဖို့ လိုအပ်တာပါ။

ရှေ့မှာ ပြောခဲ့တဲ့ worst case Big O notation (O) နဲ့ ဖော်ပြရရင်

Summation(1) မှာ

1. $n = 1$
2. Repeat while $n <= \text{num}$:
3. $\text{total} = \text{total} + n$
4. $n = n + 1$
5. End of loop

Looping ကို အဲဒီလို ရေးထားတဲ့အတွက် n တန်ဖိုးသည် 1 ကနေစတယ်၊ num အထိ looping ပတ်ပြီးပေါင်းရတဲ့အတွက်

num တန်ဖိုး 1 ဆိုရင် $\Rightarrow 1$ ကြိမ် looping ပတ်

num တန်ဖိုး 2 ဆိုရင် $\Rightarrow 2$ ကြိမ် looping ပတ်

num တန်ဖိုး 3 ဆိုရင် $\Rightarrow 3$ ကြိမ် looping ပတ်

num တန်ဖိုး K ဆိုရင် $\Rightarrow K$ ကြိမ် looping ပတ်

num တန်ဖိုး n ဆိုရင် $\Rightarrow n$ ကြိမ် looping ပတ်

ဒါကြောင့် $O(n)$ လို့ ပြောလို့ရပါတယ်။ Big O ဟာ input တန်ဖိုး n အတွက်ရေးလွှဲရှိပါတယ်။

Summation 2 မှာကတော့

$\text{total} = \text{num} * (\text{num}+1) / 2$

ဘာလာလာ ဒီတစ်ကြောင်းပဲတွက်တာဖြစ်လို 0(1) လို့ ပြောလိုရပါတယ်။ num တန်ဖိုး 1 ဖြစ်လဲ တစ်ကြိမ်ပဲ တွက်မယ်။ num တန်ဖိုး 2 ဖြစ်လဲ 1 ကြိမ်ပဲတွက်မယ်ဆိုတဲ့ အဓိပ္ပာယ်ဖြစ်ပါတယ်။ 0(1) ဆိုတာ input ဘာဖြစ်ဖြစ် constant time ကြောတယ်လို့ ဆိုလိုတာဖြစ်ပါတယ်။

Factorial Algorithm

$0! = 1$, $1! = 1$, $n! = n * (n-1) * \dots * 3 * 2 * 1$ ဆိုတာ ရှေ့မှာ ရှင်းပြခဲ့ ပြီးမဖြစ်ပါတယ်။ ဒီတော့ factorial ရှာဖို့အတွက်ဆိုရင် user ပေးတဲ့ ကဏ္ဍာန်းဟာ 0 နဲ့ အထက် ဖြစ်ရပါမယ်။

Algorithm: Factorial

```
/*input number must be greater than or equal to 0*/
1. factorial(n)
2.   If n<=1 then
3.     Return 1
4.   ans = 1
5.   Repeat while n>0:
6.     ans = ans * n
7.     n = n - 1
8.   End of loop
9.   return ans
```

Line 1 မှာ /* နှင့် */ အတွင်း ရေးတာကို comment ရေးတာလိုပေါ်တယ်။ comment ဟာ တွက်တာချက်တာ မဟုတ်ပါဘူး၊ algorithm အသုံးပြုမည့်သူကို ပြောချင်တဲ့ စကားကို ရေးတာဖြစ်ပါတယ်။ Programmer တစ်ယောက်ဟာ ကိုယ်ရေးထားတဲ့ code ကို လိုအပ်ချက်အရ ရေးခဲ့ပြီး နှစ်အတော်ကြောမှ ပြန်ပြင်ရတာမျိုးလဲ ရှိပါတယ်။ အဲဒီအချိန် code ကိုပြန်ကြည့်ရင် ကိုယ်ဘာတွေ ရေးထားခဲ့မှန်း မမှတ်စိတော့ပါဘူး။ ပြင်တာထက် ကိုယ့်ရဲ့ code ကို နားလည်အောင် ပြန်လုပ်ရတာက အချိန်အတော်ယူပါတယ်။ Code ကို စတင်ရေးသားခဲ့တဲ့ အချိန်ကသာ comment

ရေးထားခဲ့ရင် comment ဖတ်လိုက်ရနဲ့ နားလည်လွယ်တာမို့ အားစိုက်ရ သက်သာသွားမှာ ဖြစ်ပါတယ်။ နောက်တစ်ခုက ကိုယ်က code ရေးခဲ့တယ်၊ ပြီးတော့ အလုပ်ထွက်သွားတယ်၊ ပြင်ဖို့လိုတယ်၊ ကျွန်းခဲ့တဲ့ programmer က ပြင်ရတော့မှာ ဖြစ်ပါတယ်။ ကိုယ်ကသာ comment လေးတွေ ရေးပေးခဲ့ရင် ပြင်ရမည့် programmer အတွက် အတော်လေး သက်သာစေမှာဖြစ်ပါတယ်။ **ဒါကြာင့် comment** ရေးတာဟာ **programmer** တွေမှာ ရှိသင့်တဲ့အလေ့အကျင့်ကောင်းတွေထဲက တစ်ခုဖြစ်ပါတယ်။ ဒီ algorithm မှာတော့ input သွင်းတဲ့ တန်ဖိုးဟာ zero ထက်ကြီးပြီး ညီရမယ်ဆိုတာကို အသိပေးထားတာဖြစ်ပါတယ်။

Line 2 & 3 ကတော့ user ≥ 0 ကိုပဲ input သွင်းဖို့ ပြောထားတာဖြစ်လို့၊ $n \leq 1$ ဆိုတာ 0! နဲ့ 1! အတွက် 1 return ပြန်ပေးဖို့ ရေးသားထားတာဖြစ်ပါတယ်။

Line 4 ကတော့ မြောက်ရင် မြောက်တဲ့ တန်ဖိုးရစေဖို့ အဖြေထည့်မည့် ans ထဲကို 1 initialize လုပ်ထားတာ ဖြစ်ပါတယ်။ initialize လုပ်တယ် ဆိုတာ ထည့်တာကို ပြောတာပါ။

Looping ကတော့ n ကနေ 1 အထိ 1 လျှော့လျှော့ ပြီး မြောက်သွားတာ ဖြစ်ပါတယ်။ မြောက်လို့ရတဲ့ အဖြေကို ans ထဲ ထည့်ပါတယ်။ တကယ်က n ကနေ 2 အထိပဲ loop ပတ်ရင်တောင် ရပါတယ်။ ဘာလို့လဲဆိုရင် 1 ထိ 100 ပတ်ထားပြီး 1 နဲ့ မြောက်တာ အဖြေကို မပြောင်းလဲစေတဲ့အတွက်ကြာင့် ဖြစ်ပါတယ်။

Line 9 ကတော့ looping ပြီးသွားတဲ့ အချိန်မှာ အဖြေ ans ကို return ပြန်ပေးတာ ဖြစ်ပါတယ်။



ကဲ ဒီတစ်ခါ trace လိုက်ရမှာကတော့ စာဖတ်နေတဲ့သူရဲ့ အလုညွှေဖြစ်ပါတယ်။

result = factorial (0)

result = factorial (1)

```
result = factorial (5)
```

Permutation

$${}_n P_r = \frac{n!}{(n-r)!}$$

Algorithm: Permutation

```
/*n must be greater than or equal to r*/
```

1. permutation(n,r)
2. If n < r then
3. Return 0
4. ans = factorial (n)/factorial (n-r)
5. return ans

အပေါ်မှာ ရေးခဲ့တဲ့ factorial function ကို လုမ်းချေပြီး တွက်ထားတာဖြစ်ပါတယ်။ ဒီမှာတော့ line 4 လေးပဲ ရှင်းစရာလိုမယ် ထင်ပါတယ်။ factorial(n) ဆိုပြီး လုမ်းချေလိုက်တော့ factorial function က parameter ထဲကို n တန်ဖိုး ဝင်သွားပြီးတော့ factorial function က return ပြန်ပေးလိုက်တဲ့ n! ရဲ့ အဖြေက အဲဒီလုမ်းချေတဲ့နေရာ ဝင်လာမှာဖြစ်ပါတယ်။ factorial(n - r) ဆိုပြီး လုမ်းချေလိုက်တော့ factorial function က parameter ထဲကို n-r ဝင်သွားပြီးတော့ factorial function က return ပြန်ပေးလိုက်တဲ့ n-r! ရဲ့ အဖြေက အဲဒီလုမ်းချေတဲ့နေရာ ဝင်လာမှာဖြစ်ပါတယ်။ အဲဒီမှာမူ ဝင်လာတဲ့ တန်ဖိုး ၂ ခုကို စားပြီး ရလာတဲ့ အဖြေကို ans ထဲ ထည့်လိုက်တာဖြစ်ပါတယ်။ ဒီလောက်ဆို function တွေ အချင်းချင်းလုမ်းချေသုံးလို့ရတာလဲ သဘောပေါက်လောက် ပါတယ်။ function ခဲ့ရေးတာကိုက ကြိုက်တဲ့ နေရာကနေ ချေသုံးလို့ ရနို့ဖြစ်ပါတယ်။

ဒီတစ်ခါ trace လိုက်ရမှာကလည်း စာဖတ်သူရဲ့ အလှည့်ပဲ ဖြစ်ပါတယ်။ Trace လိုက်ဖို့ မပျင်းပါနဲ့။ များများ trace လိုက်လေလေ များများ မြင်လာလေလေ၊ များများ နားလည်လာလေလေ ဖြစ်မှာ ဖြစ်ပါတယ်။

```
result = permutation(2,3)
```

```
result = permutation(3,2)
```

```
result = permutation(3,3)
```



လေ့ကျင့်ကြည့်ကြရအောင်ပါ။

- 1 ကနေ 100 အတွင်းမှာ ရှိတဲ့ prime number တွေကို output ထုတ်ပြပေးပါ။ လုပ်မှာက 1 ကနေ 100၊ သို့သော် prime number သည် 2 က စတာဖြစ်လို့ 2 ကနေ 100 မထိ looping ပတ်မှာဖြစ်ပါတယ်။ 2 ကနေ 100 အတွင်း တစ်လုံးဝင်လာတာဘဲ prime number ဖြစ်မဖြစ် စစ်ဖို့ 2 ကနေစပြီး မပြတ်မချင်းစားတာကို တွက်ရမှာဖြစ်ပါတယ်။ ဒီတော့ ဂဏန်းတစ်လုံးဝင်လာတိုင်း ထပ်ခါထပ်ခါ စားတာလုပ်မှာဖြစ်တဲ့အတွက်၊ တစ်လုံးချင်းစိဝင်လာမည့် ဂဏန်းတွေသည် outer loop ဖြစ်ပြီး ထပ်ခါထပ်ခါစားမှာက inner loop ဖြစ်မှာ ဖြစ်ပါတယ်။
2. quadraticEquation(1, -6, 9) ကို Trace လိုက်ပြပါ။
3. quadraticEquation(1, -3, -10) ကို Trace လိုက်ပြပါ။
4. quadraticEquation(1, , 3, 6) ကို Trace လိုက်ပြပါ။



Chapter အနှစ်ချုပ်

- Algorithm တစ်ခုဟာ မှန်လည်းမှန်ရမယ်၊ မြန်လည်းမြန်ရမယ်၊ ပေါ့လည်း ပေါ့ရပါမယ်။
- မှန်တာခြင်းတူခဲ့ရင် မြန်သလား(time complexity) နဲ့ ပေါ့သလား (space complexity) တွေနဲ့ နိုင်းယှဉ်လေ့ရှိပါတယ်။
- Complexity တွက်တဲ့နေရာမှာ worst case, average case နဲ့ best case ဆိုပြီး ၃ မျိုးရှိပါတယ်။
- worst case မှာ ကဲ့ပြားတာ သိသာထင်ရှားတာဖြစ်လို့ algorithm တွေကို နိုင်းယှဉ်တော့မယ် ဆိုရင် worst case (big O) နဲ့ နိုင်းယှဉ်လေ့ရှိပါတယ်။
- Trace များများလိုက်လေ algorithm ကို နားလည်လေလေ၊ အလုပ်လုပ်ပုံတွေ နားလည်လေလေ ဖြစ်ပါတယ်။ “က” တောင်မရေးတတ်ပဲ “ကာ” ရေးဖို့ဆိုတာ မဖြစ်နိုင်ပါဘူး။ ဒီလိုပါပဲ algorithm တွေ trace တောင်မလိုက်တတ်ဘူးဆိုရင် ကိုယ်ကိုယ်တိုင် code ရေးဖို့မလွယ်ပါဘူး။ ဒါကြောင့် စိတ်ဝင်တစား စာအုပ်ထဲမှာ ချုပြီး trace လိုက်ပါ။
- အဓိပ္ပာယ်ရှိတဲ့ variable name ပေးတာ၊ comment ရေးတာ၊ formula တွေကိုအသုံးချတာတွေဟာ programmer တစ်ယောက်မှာ ရှိသင့်တဲ့ အရည်ချင်းကောင်းတွေ ဖြစ်ပါတယ်။

“ရှင်းအောင်ပြောရရင်

Function ဆိုတာ ခပ်ချေချေရယ်

သူကိုလာမခေါ်ရင် ဘယ်သူမှ အဖက်မလုပ်ဘူး၊

တစ်ဖက်က ဟေးဆိုပြီး function call ခေါ်လိုက်ရင်တော့

function call ခေါ်တဲ့သူကို တစ်ခုခုနဲ့

Return ပြန် (တုပြန်) တတ်တယ်”

အခန်း (၉)

Recursion

Recursion ဆိတာဘာလဲ။

Recursion ဆိတာ မိမိ function ထဲကနေပြီး မိမိကိုမိမိ ပြန်ပြီး function call ခေါ်တာဖြစ်ပါတယ်။ recursion ဆိတာ ဘာနဲ့ တူသလဲ ဆိုရင် looping နဲ့တူပါတယ်။ looping ရဲ့ အစား သုံးတာဖြစ်ပါတယ်။ Recursion မှာ အရေးကြီးဆုံး သတိထားရမည့် အချက်ကတော့ exit point တွေ ထားပေးရပါတယ်။ ထားမပေးရင် infinite loop သဘောမျိုး မရပ်တော့ပဲ အလုပ်လုပ်နေမှာ ဖြစ်လို့ အဖြေထွက်လာမှာ မဟုတ်ပါဘူး။

Factorial

$$n! = n * (n-1)!$$

အဲဒီမှာဆိုရင် n factorial ကနေ $n-1$ factorial ကို ပြန်ခေါ်ထားတာဖြစ်ပါတယ်။ factorial ထဲကနေ factorial ကို ပြန်ခေါ်တာဖြစ်လို့ recursion ဖြစ်ပါတယ်။

Algorithm: Factorial

1. factorial(n)
2. If $n \leq 1$ then:
3. Return 1
4. Else:
5. Return ($n * \text{factorial}(n-1)$)

ဒီ algorithm မှာ line 5 က recursion ဖြစ်ပါတယ်။ return statement ထဲမှာ factorial (n-1) ဆိုပြီး မိမိ function ကို မိမိ ပြန်ခေါ်ထားလို့ ဖြစ်ပါတယ်။

Exit point ကတေသာ့ $0! \neq 1!$ အတွက် loop ပတ်စရာမလိုပဲ အဘဲ့ 1 ကို return ပြန်ပေးဖို့ ရေးထားတဲ့ line 2 နဲ့ 3 ပဲ ဖြစ်ပါတယ်။ Trace လိုက်ပြလိုက်ရင် သဘောပေါက်သွားမှာပါ။

$f = \text{factorial}(5)$

Line	n	အလုပ်လုပ်ပုံ	Result
1	5		
2		$n \leq 1 \rightarrow 5 \leq 1$ ဆိုတော့မှားတယ်၊ မှားတော့ false ထွက်တော့ else ရဲ့ return $n * \text{factorial}(n-1)$ ကို အလုပ်လုပ်မယ်။ return $5 * \text{factorial}(4)$ ဆိုပြီး ဖြစ်မယ်။ $\text{factorial}(4)$ ဆိုတာ function call ဖြစ်တဲ့အတွက် function အစ line 1 ကို သွားအလုပ်လုပ်မယ်။ Result မှာ ပြထားသလို $5 * \text{factorial}(4)$ လို့ရမယ်။	Return $5 * \text{factorial}(4)$
1	4	$\text{Factorial}(4)$ လို့ ခေါ်တာဖြစ်လို့ n ထဲကို 4 ဝင်သွားမယ်။	
2		$n \leq 1 \rightarrow 4 \leq 1$ ဆိုတော့မှားတယ်၊ မှားတော့ false ထွက်တော့ else ရဲ့ return $n * \text{factorial}(n-1)$ ကို အလုပ်လုပ်မယ်။ return $4 * \text{factorial}(3)$ ဆိုပြီး ဖြစ်မယ်။ $\text{factorial}(3)$ ဆိုတာ function call ဖြစ်တဲ့အတွက် function အစ line 1 ကို	Return $5 * 4 * \text{factorial}(3)$

		<p>သွားအလုပ်လုပ်မယ်။ Result မှာ ပြထားသလို factorial(4) နေရာမှာ 4 * factorial(3) ဆိုပြီး အစားထိုးသွားမယ်။</p>	
1	3	<p>Factorial (3) လို ခေါ်တာဖြစ်လို n ထဲကို 3 ဝင်သွားမယ်။</p>	
2		<p>$n \leq 1 \rightarrow 3 \leq 1$ ဆိုတော့မှားတယ်၊ မှားတော့ false ထွက်တော့ else ရဲ့ return $n * \text{factorial}(n-1)$ ကို အလုပ်လုပ်မယ်။ return 3 * factorial(2) ဆိုပြီး ဖြစ်မယ်။ factorial (2) ဆိုတာ function call ဖြစ်တဲ့အတွက် function အစ line 1 ကို သွားအလုပ်လုပ်မယ်။ Result မှာ ပြထားသလို factorial(3) နေရာမှာ 3 * factorial(2) ဆိုပြီး အစားထိုးသွားမယ်။</p>	<p>Return $5 * 4 *$ $3 * \text{factorial}(2)$</p>
1	2	<p>Factorial (2) လို ခေါ်တာဖြစ်လို n ထဲကို 2 ဝင်သွားမယ်။</p>	
2		<p>$n \leq 1 \rightarrow 2 \leq 1$ ဆိုတော့မှားတယ်၊ မှားတော့ false ထွက်တော့ else ရဲ့ return $n * \text{factorial}(n-1)$ ကို အလုပ်လုပ်မယ်။ return 2 * factorial(1) ဆိုပြီး ဖြစ်မယ်။ factorial (2) ဆိုတာ function call ဖြစ်တဲ့အတွက် function အစ line 1 ကို သွားအလုပ်လုပ်မယ်။ Result မှာ ပြထားသလို</p>	<p>Return $5 * 4 * 3 *$ $2 * \text{factorial}(1)$</p>

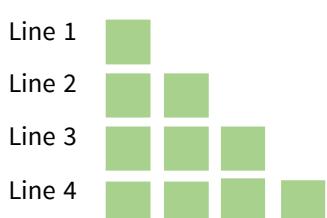
		factorial(2) နေရာမှာ 2 * factorial(1) ဆိုပြီး အစားထိုးသွားမယ်။	
1	1	Factorial (1) လို ခေါ်တာဖြစ်လို n ထဲကို 1 ဝင်သွားမယ်။	
2		n<=1 → 1<=1 ဆိုတော့မှန်တယ်၊ မှန်တော့ true ထွက်တော့ else အောက်က recursionကို မလုပ်တော့ပဲ if ရဲ့ အောက်က exit point ဖြစ်တဲ့ return 1 ကို အလုပ်လုပ်သွားမယ်။ ဒီတော့ factorial(1) နေရာမှာ အဖြေ 1 ဝင်သွားမှာ ဖြစ်တယ်။ result အကွက်ကို <u>ကြည့်ပါ။</u>	Return 5 * 4 * 3 * 2 * 1

Exit point ကနေ ထွက်လာပြီး return 5*4*3*2*1 ဆိုတော့ return 120 ဆိုပြီး မူရင်း function call ဆီ အဖြေပြန်ပို့ပေးတော့ f ထဲကို 120 ဝင်သွားမှာ ဖြစ်ပါတယ်။

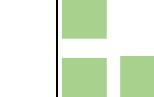
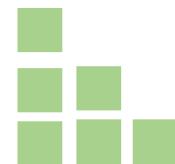
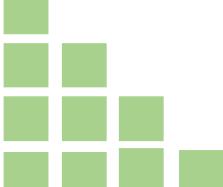
F = factorial (7)

F = factorial (0) တို့အတွက် trace လိုက်ကြည့်ပါ။

Triangle



ပုံလေးက line 4 အတိပဲ နမူနာ ဆွဲပြထားတာ ဖြစ်ပါတယ်။ Line 1 တစ်ကြောင်းထဲဆိုရင် အတူးလေး ၁ တုံးထဲ ဖြစ်ပါမယ်။ Line 2 ထိ ဆိုရင် အတူးအရေအတွက်က $2 + 1$ (line 2+ line 1) ဖြစ်ပါမယ်။

Line အရေအတွက်	1	2	3	4
				
No of Rectangle:	1	$2 + 1$	$3 + 2 + 1$	$4 + 3 + 2 + 1$

$$\text{triangle}(1) = 1$$

$$\text{triangle}(2) = 2 + 1 = 2 + \text{triangle}(1)$$

$$\text{triangle}(3) = 3 + 2 + 1 = 3 + \text{triangle}(2)$$

$$\text{triangle}(n) = n + \text{triangle}(n-1) \text{ //recursion}$$

Algorithm: Triangle

1. triangle(l)
 2. If $l==1$ then:
 3. Return 1
 4. Else:
 5. Return $(l + \text{triangle}(l-1))$
-

ဒီတစ်ခါတော့ Trace ကို အတိုချုံးပြီး လိုက်ပြရင် သဘောပေါက်လောက်ပါပြီ။

$$\text{triangle}(4) = 4 + \text{triangle}(3)$$

$$= 4 + 3 + \text{triangle}(2)$$

$$= 4 + 3 + 2 + \text{triangle}(1) = 4 + 3 + 2 + 1 = 10$$



လေ့ကျင့်ကြည့် ကြရအောင်။

1. New Year Count down အတွက် recursion function ရေးပါ။

n 5 4 3 2 1 Happy New Year

သဘောကတော့ n တန်ဖိုး input လက်ခံ၊ ပြီးရှင် n ကနေ စပြီး count down လုပ်မယ်။ 0

ရောက်တာနဲ့ Happy New Year လို့ ထုတ်မယ်။ ဒီတော့ 0 ဟာ exit point ဖြစ်ပါမယ်။

ရေးကြည့်ပါ။ ရေးပြီး ဒီတိုင်းမထားပဲ မှန်မမှန် trace ပြန်လိုက်ပါနော်။

2. Power function ကို recursion ရေးပါ။ Parameter 2 ခု လက်ခံပါမယ်။ b နဲ့ p ဖြစ်ပါတယ်။

b က base ဖြစ်ပြီး p က power ဖြစ်ပါတယ်။ နမူနာပြောပြရရင်

$$\text{power}(2, 1) = 2$$

$$\text{power}(2, 2) = 2 * 2 = 2 * \text{power}(2, 1)$$

$$\text{power}(2, 3) = 2 * 2 * 2 = 2 * \text{power}(2, 2)$$

မည်သည့်ကိန်းမဆို ထပ်ကိန်း 0 တင်ရင် 1 ရပါတယ်။

3. ရှုံးက သချို့အခန်းမှာ ပြောခဲ့တဲ့ Fibonacci အတွက် recursion function ရေးပါ။

User ထံမှ ၃ ခု လက်ခံပါမယ်။ ပထမကိန်းရယ်၊ ဒုတိယကိန်းရယ်၊ ထုတ်ပြချင်တဲ့

အကြိုးဆုံးကိန်းရယ် ဆိုပြီး လက်ခံပါမယ်။ နမူနာပြောရရင် user က

ပထမကိန်း 0

ဒုတိယကိန်း 1

အကြီးဆုံးကိန်း 20 လို့ ရှိက်ထဲတယ်ဆိုပါတော့။ ဒါဆို အဖွဲ့က

0 1 1 2 3 5 8 13

လိုတွက်ရပါမယ်။ 8+13 က 21 ဆိုတော့ အကြီးဆုံးကိန်း 20 ထက်ကြီးတော့ မထုတ်တော့ပါဘူး။

Fibonacci ပုံသေနည်းက

$F_n = F_{n-1} + F_{n-2}$ ဖြစ်ပါတယ်။ မမှန်မိရင် ပြန်ရှာဖတ်ပါ။ ရေးပြီး ဒီတိုင်းမထားပဲ မှန်မမှန် trace ပြန်လိုက်ပါနော်။



Chapter အနှစ်ချုပ်

- Recursion ဆိုတာ looping အစား သုံးတာဖြစ်ပါတယ်။
- Exit point သာ သေသေချာချာ ထည့်မရေးခဲ့ရင် infinite loop ဖြစ်သွားနိုင်ပါတယ်။
- Function တစ်ခု လုမ်းခေါ်လိုက်ရင် memory ပေါ်မှာ နေရာယူပြီးတော့၊ function ထဲက နေ return ပြန်တာဖြစ်ဖြစ် အပြီး ထွက်သွားမှသာ အဲဒီ function အတွက် နေရာယူထားတာတွေကို ဖျက်ပစ်တာ ဖြစ်ပါတယ်။ ဒါကြောင့် recursion ဟာ ပထမ call ထားတာ မပြီးခင်၊ အဖြေမထွက်သေးခင် အဲဒီ call အတွက် အဖြေထွက်ဖို့ မိမိ function ကို မိမိ ထပ်ခါ ထပ်ခါ ပြန်ပြန်ခေါ်နေရတဲ့ အတွက် memory နေရာ ပိုစားပါတယ်။ space complexity ပိုများပါတယ်။
- Code ကိုကြည့်လိုက်ရင် တော်တော်လေး ရှင်းလင်းနေတာဖြစ်လို့ Code complexity(code ရဲ့ ရှုပ်ထွေးမှုကတော့) နည်းပါတယ်။

“တခို့လူတွေ

ရစ်ကလည်း ရစ်ချင်တယ်၊ ရစ်ဖို့ရာလဲ စိတ်မရှည်ဘူး။

‘ပြောပြန်ရင် ဟိုနေ့စကားထပ်ပြောရအုံမယ်’

ရစ်တာတောင် Recursion ခေါ်ရစ်တာ”

အခန်း (၁၀)

Data များသိမ်းဆည်းပုံ

Data သိမ်းဆည်းခြင်း

Data တွေကို သိမ်းဆည်းတဲ့ နည်း၂ နည်းရှိတာ Data structure မိတ်ဆက် အခန်းမှာ ပြောခဲ့ပြီး ဖြစ်ပါတယ်။

- တန်ဖိုး တစ်ခုသာ သိမ်းဆည်းခြင်း** - လူတစ်ယောက်မှာ အသက် ဘယ်နှစ်နှစ်လဲ ဆိုရင် လက်ရှိ အသက်ဟာ တန်ဖိုး တစ်ခုပဲ ရှိပါတယ်။ အဲဒီလို တစ်ခုပဲ ရှိတဲ့ တန်ဖိုးတွေကို သိမ်းဆည်းချင်ရင် primitive data types နဲ့ သိမ်းဆည်းရပါတယ်။
- တန်ဖိုးများစွာ စုစုပေါင်း သိမ်းဆည်းခြင်း** - မွေးချင်းမောင်နှစ်မဲ တွေကို သိမ်းမယ်ဆိုရင် တချို့က မောင်နှစ်မဲ တစ်ယောက်မှ မရှိဘူး၊ တစ်ချို့က ၁ ယောက်ရှိတယ်၊ တချို့က ၂ ယောက်ရှိတယ်။ တချို့က အများကြီး ရှိတယ်။ ဒီတော့ အားလုံး အဆင်ပြုဆိုရင် အများကြီး စုစုပေါင်းသိမ်းဆည်းလို့ ရအောင် ဖန်တီးပေးထားရမှာ ဖြစ်ပါတယ်။

Variable Declaration

Variable Declaration အကြောင်း မပြောခင်မှာ programming languages တွေကို strongly typed နဲ့ loosely typed languages ဆိုပြီးခဲ့လို့ရတဲ့ အကြောင်း အရင်ပြောပြပါမယ်။

- **strongly typed language** ဆိုတာ data type ကို programmer ကပြောပေးရတာ၊ သတ်မှတ်ပေးရတာဖြစ်ပါတယ်။ ဥပမာ အနေနဲ့ ဆိုရင် C, Java, C# တို့ ဖြစ်ပါတယ်။

- **Loosely Typed Language** ဆိတာ data type ကို programmer က ပြောပေးစရာမလိုပါဘူး။ programmer assign လုပ်လိုက်တဲ့ တန်ဖိုးပေါ်မှုတည်ပြီး language က auto သတ်မှတ်ပေးသွားတာ ဖြစ်ပါတယ်။ ဤပမာ အနေနဲ့ကတော့ PHP, JavaScript တို့ပဲ ဖြစ်ပါတယ်။

ကလေးလေးတစ်ယောက် မွေးဖားလာရင် အမည်ပေး ကပွန်းတပ် လုပ်ရသလိုပဲ variable ကိုလည်း declaration လုပ်ရတဲ့ အကြောင်း ရှုမှာပြောခဲ့ပြီးပါတယ်။

```
Data_type variableName [=value];
```

ဒီရေးနည်း က strongly typed language မှာ variable declaration ရေးနည်း ဖြစ်ပါတယ်။ [=value] ဆိတာကတော့ ထောင့်ကွင်းထဲ ထည့်ရေးထားတာက option ဖြစ်ပါတယ်။ အဓိပါယ်ကတော့ ပါလည်းရတယ်၊ မပါလည်းရတယ် ဆိုတဲ့ အဓိပ္ပာယ် ဖြစ်ပါတယ်။ value ဆိုတဲ့ တန်ဖိုး assign လုပ်ချင် (ထည့်ချင်) ရေး၊ မထည့်ချင် မရေးနဲ့ပေါ့။

```
varialbeName [=value];
```

ဆိတာကတော့ loosely type language တစ်ခုဖြစ်တဲ့ PHP မှာ variable declaration လုပ်တဲ့နည်းဖြစ်ပါတယ်။ သူက data type ပြောစရာမလိုပါဘူး။ Programmer ထည့်လိုက်တဲ့ တန်ဖိုးပေါ်ကြည့်ပြီး data type ကို language က auto သတ်မှတ်ပေးသွားမှာ ဖြစ်ပါတယ်။

Language တစ်ခုနဲ့ တစ်ခု ရေးနည်းလေးတွေ အနည်းငယ်တော့ ကဲ့တတ်ပါတယ်။ Programming Langauge အများစုံမှာ စာကြောင်းတစ်ကြောင်းဆုံးရင် ';' ဖြင့် အဆုံးသပ် ပေးရပါတယ်။ English စာကြောင်းတစ်ကြောင်းမှာ ';' နဲ့ အဆုံးသပ်သလို ဖြစ်ပါတယ်။ ဒါပေမဲ့ တချို့ language မှာ အဆုံးသပ် ';' က မလိုအပ်ပြန်ပါဘူး။ သိပ်ပြီးခေါင်းရှုပ်မခံပါနဲ့။ Programming Language တစ်ခုနဲ့တစ်ခု ကဲ့ပြားတတ်တယ် ဆိတာရယ်၊ algorithm ဆိတာ programmer

အားလုံးနားလည်တဲ့ အရာဆိုတာရယ် ဒီလောက် သဘောပေါက်ရင် ရပါ ဖြို့။ ဒီမှာတော့ လိုအပ်တဲ့ နေရာလေးတွေမှာ Java နဲ့ PHP ကို အသုံးပြုပြီး ရှင်းပြသွားမှာ ဖြစ်ပါတယ်။

တန်ဖိုး တစ်ခုသာ သိမ်းဆည်းခြင်း(Primitive Data Types)

တန်ဖိုး တစ်ခုသာ သိမ်းဆည်းနိုင်တဲ့ data type တွေကို Primitive Data Types လို ခေါပါတယ်။ Data types ဆိုတာက အထဲ မှာ သိမ်းမည့် တန်ဖိုး(value) ရဲ့ အမျိုးအစားကို ဆိုလိုတာဖြစ်ပါတယ်။ programming language တစ်ခုနဲ့တစ်ခု ခွင့် ပြုတဲ့ data types တွေ မတူညီကြပါဘူး။ ဥပမာ Java programming language မှာ ခွင့်ပြုတဲ့ data types နဲ့ PHP မှာ ခွင့်ပြုထားတဲ့ data types တွေ မတူညီကြပါဘူး။ ဒီတော့ ဒီမှာ အသုံးများတဲ့ primitive data types တွေဖြစ်တဲ့ Integer, Double/Float နဲ့ Character သုံးခုကိုပဲ အရင်ပြောကြရအောင်ပါ။

- Interger - data ရဲ့ တန်ဖိုးက ကိန်းပြည့်ဆိုရင် interger data type ပါ။ အသက်ဆိုရင် ငါ နှစ် ဤ နှစ် စသည်ဖြင့် ကိန်းပြည့်ပြောလေ့ရှိပါတယ်။ ဒသေမနဲ့ ပြောလေ့မရှိပါဘူး။ ဒါကြောင့် အသက်ဆိုရင် integer data type ပါ။

```
int age = 5;
```

ဒါကတော့ Java မှာ variable declaration လုပ်တာလေး နမူနာရေးပြထားတာဖြစ်ပါတယ်။

Data type က int(integer), variablename က age, value က 5 ဖြစ်ပါတယ်။ တကယ်လို value တန်ဖိုး မထည့်ပဲ ကြော်ချင်တော့

```
int age;
```

လို့ ရေးရပါမယ်။

```
$age=5;
```

ဒါကတော့ loosely typed language ဖြစ်တဲ့ PHP မှာ ရေးတဲ့ ပုံစံဖြစ်ပါတယ်။ loosely type language ဖြစ်တဲ့အတွက် data type ပြောစရာမလိုပါဘူး။ ထည့်လိုက်တဲ့ တန်ဖိုး 5 ကိုကြည့်ပြီး integer data type လို့ auto သတ်မှတ်သွားမှာ ဖြစ်ပါတယ်။ PHP ရဲ့ ထူးခြားချက်က variable name ဟာ \$ နဲ့ စရပါတယ်။

- Double/float - data ရဲ့ တန်ဖိုးက အသေမ ဆိုရင် double သို့မဟုတ် float သုံးလို့ရပါတယ်၊ ဥပမာ PI တန်ဖိုးက 3.142 ဆိုတော့ Pi ဟာ Double/float data type ပါ။ Java မှာ ကြော်ကြော်ဆိုရင်


```
double pi = 3.142;
```

 လို့ ရေးရမှာ ဖြစ်ပါတယ်။
 - စာလုံးလေးတစ်လုံး ဥပမာ 'A' ဆိုတာလေး သိမ်းချင်ရင် character data type ဖြစ်ပါတယ်။ single quote ထဲမှာ ထည့်ပြီးရေးပေးရပါတယ်။ A ဆို တစ်လုံးပဲ သိမ်းလို့ရပါတယ်၊ AB နှစ်လုံး သိမ်းလို့ မရပါဘူး။ Java မှာ ကြော်နည်းလေး နမူနာပြရရင်


```
char ch = 'A';
```

 လို့ ရေးပါတယ်။ Java မှာ character data type ကို char လို့ ရေးရပါတယ်။
- ဒီလောက်ဆိုရင် တန်ဖိုး တစ်ခုတည်း သိမ်းလို့ရတဲ့ primitive data types ကို သဘောပေါက်လောက်ပြီ ထင်ပါတယ်။

တန်ဖိုး များစွာကို စုစုည်းသိမ်းဆည်းပုံ

တန်ဖိုး အများကြီးကို စုစုည်း သိမ်းဆည်းတဲ့ နည်းတွေ အများကြီးရှုပါတယ်။ ဒီအခန်းမှာတော့ အဲဒီထဲက

- String
- File

- Array (one-dimensional array, multi-dimensional array)
- Linked list
- Tree

တို့ အကြောင်း အနည်းငယ် ရှင်းပြသွားပါမယ်။

String

String ဆိတာ character တွေ အတွဲလိုက်ကြီး စုထားတာဖြစ်ပါတယ်။ character အလုံးအရေအတွက် ကြိုက်သလောက်ပါလို့ရပါတယ်။ character ဆိုလို a to z လိုပဲ မထင်လိုက်ပါနဲ့။ keyboard ပေါ်မှုရှိတဲ့ key တစ်ခုချင်းဟာ(special characters တွေကော့၊ digit တွေကော့ စသည်ဖြင့် အကုန်ပါ ပါတယ်။) single quote နဲ့ရေးရင် character တွေပါပဲ။ ဤ ပမာ အနေနဲ့ကတော့ လူနာမည်။ မြို့အမည်၊ အမိမိလိပ်စာ စတာတွေ ဖြစ်ပါတယ်။ value ကို ရေးရင် double quote နဲ့ ရေးရပါတယ်။

```
String school = "StudyRightNow";
```

```
String schoolType = "Technology School";
```

```
$city = "Mandalay";
```

ဒါကတော့ java နဲ့ PHP မှာ String data type ရေးနည်းကို နမူနာ ရေးပြတာဖြစ်ပါတယ်။ data type ထည့်ရေးထားတာက Java နဲ့ ရေးပြထားတာ ဖြစ်ပြီး၊ data type မပါဘဲ variable name \$ (dollar sign) နဲ့ စထားတာက PHP နဲ့ ရေးပြထားတာ ဖြစ်ပါတယ်။

File

Roll_No	Name	Major	Year
1	Aye Aye	CT	Second Year
2	Su Su	CT	Second Year
3	Aye Aye	CT	Second Year
4	Aung Aung	CT	Second Year

Student Table

ဒီလို သိမ်းတဲ့ ပုံစံကို File လိုခေါ်ပါတယ်။ ခေါင်းစဉ်တွေဖြစ်တဲ့ Roll_No, Name, Major, Year တို့ကို **attribute or field or properties** လို့ ခေါ်ပါတယ်။ 1 , Aye Aye, CT စသည်ဖြင့် table မှာ အစိမ်းနှုရောင်နဲ့ ခြေထွက်ပြထားတာ အားလုံးကို **values** လိုခေါ်ပါတယ်။

1	Aye Aye	CT	Second Year
---	---------	----	-------------

ဒါကတော့ ကျောင်းသားတစ်ယောက်ကို ကိုယ်စားပြုတာဖြစ်လို့ **record** လိုခေါ်ပါတယ်။ Student Table မှာဆိုရင် record 4 ခုပါပါတယ်။ အဲဒီလို records တွေ ပေါင်းထားတာကို **File** လို့ ခေါ်ပါတယ်။

နောက်တစ်ခု သိမေးချင်တာက unique ဖြစ်တာပါ။ unique ဖြစ်တယ်ဆိုတာ မထပ်တာ သူကိုပြောလိုက်တာနဲ့ ဘယ်သူလဲ ဘာလဲ တန်းသိမေးတာကို ပြောတာဖြစ်ပါတယ်။ Student Table ကိုကြည့်ပါ။ Aye Aye ဆိုပြီး အမည်တူ နှစ်ယောက်ရှုပါတယ်။ နာမည်ဆိုတာ တူနှစ် ထပ်နှစ်လို့ unique မဖြစ်ပါဘူး။ ဒါပေမဲ့ roll-no ကတော့ မထပ်ပါဘူး။ unique ဖြစ်ပါတယ်။ ဒါကြောင့် ဒီ table မှာ roll-no တစ်ခုကို ပြောလိုက်တာနဲ့ ဘယ်သူလဲ တန်းပြီး သိပါတယ်။ ဥပမာ roll-No 1 ဆိုတာနဲ့

1	Aye Aye	CT	Second Year
---	---------	----	-------------

Aye Aye, CT, Second Year ကို ပြောမှန်း တန်းပြီး သိတာမျိုးပါ။ အဲဒီလို unique ဖြစ်တဲ့ကောင်ကို primary key လိုခေါ်ပါတယ်။

နောက်ထပ် table တစ်ခု ကြည့်ကြည့်ရအောင်ပါ။

Roll_No	Name	Major	Year
1	Aye Aye	CS	Second Year
2	Su Su	CS	Second Year
1	Min Min	CT	Second Year
2	Aung Aung	CT	Second Year

အပေါ်က student table ဟာ CT major တစ်ခုတည်း ပြတာဖြစ်လို့ roll-no မထပ်ပါဘူး။ ယခု table ကတော့ CT, CS ဆိုတဲ့ major J ခု ရောသိမ်းထားတာ ဖြစ်လို့ roll-no 1 ဆိုလည်း 2 ယောက် roll_no 2 ဆိုလည်း 2 ယောက် ဖြစ်နေပါတယ်။ ဒါကြောင့် roll-no ဟာ unique မဖြစ်ဘူး။ roll-no ပြောရုံနဲ့ ဘယ်သူလဲ မသိနိုင်ပါဘူး။ ဒါပေမဲ့ CS roll-1 ဆိုရင်တော့ unique ဖြစ်ပါတယ်။ Aye Aye ကို ပြောမှန်းသိပါတယ်။ ဒါကြောင့် ဒီ table ရဲ့ primary key ကတော့ Roll_no နှင့် Major နှစ်ခုတဲ့ ထားတာ ဖြစ်ပါတယ်။

ဒီ File J ခုမှာ roll-no က 1,2,3 စသည်ဖြင့် သိမ်းတာဖြစ်လို့ int data type ဖြစ်ပါမယ်။ ကျွန်ုတဲ့ field 3 ခုကတော့ စာသားတွေ သိမ်းတာဖြစ်လို့ String ဖြစ်ပါမယ်။



Field တစ်ခုချင်းရဲ့ data type ကိုသတ်မှတ်ပေးပါ။ primary key ဘာဖြစ်မလဲ စဉ်းစားပေးပါ။

1.

Building_No	Room_No	Bed_No	Patient	Doctor	Age	Address

2.

Car_No	Owner	Color	Engine_No	City

Linear Array or One-Dimensional Arrays

မွေးချင်းညီအစ်ကိုများကို သိမ်းမည်ဆိုလျှင် တစ်ယောက်ထက်မက ရှိနိုင်တာမှို့ array ကို သုံးပြီး သိမ်းပါမယ်။ ဥပမာ ကိုထွေးမှာ Ko Ko, Ko Let, Nyi Nyi ဆိုတဲ့ မွေးချင်း ၃ ယောက်ရှိတယ်ဆိုပါစိုး။ ဒါဆို သူမှာ ညီအစ်ကို ၃ ယောက်ရှိလို့ Array အခန်း ၃ ခန်း နေရာယူပါမယ်။

0	Ko Ko
1	Ko Let
2	Nyi Nyi

siblings

- **Array** ရဲ့ အမည် ဟာ **variable name** ဖြစ်လို့ ကိုယ်ကြိုက်တာ ပေးလို့ရတယ်၊ ဒီမှာ siblings လို့ ပေးလို့က်မယ်။

- Array မှာ အခန်းနံပါတ်ရှိတယ်၊ အခန်းနံပါတ်တွေဟာ 0,1,2 အစဉ်လိုက်ဖြစ်ပါတယ်။ Java မှာက array အခန်းနံပါတ်ဟာ programmer သတ်မှတ်ခွင့်မရှိပါဘူး။ အစဉ်လိုက် auto သတ်မှတ်ပေးပြီး အမြဲ 0 က စပါတယ်။ PHP မှာတော့ 0 က စတဲ့ auto သတ်မှတ်ပေးတဲ့ အခန်းနံပါတ်အစဉ်လိုက်လဲရသလို၊ မိမိ နှစ်သက်ရာ အခန်းနံပါတ် သတ်မှတ်လို့ရပါတယ်။
- 0,1,2 ဆိုတဲ့ အခန်းနံပါတ်ကို index လို့ခေါ်ပါတယ်။
- VariableName [index] ဆိုရင် အထဲမှာထည့်ထားတဲ့ value ရပါတယ်။ ဥပမာ siblings[0] ဆိုရင် အထဲက value ဖြစ်တဲ့ Ko Ko ဆိုတာ ရပါမယ်။
- ထောင့်ကွင်း [] ထဲ ရေးတဲ့ဟာကို အခန်းနံပါတ် index လို့ခေါ်ပါတယ်၊ ဒီမှာ Ko Let ကိုလိုချင်ရင် ထောင့်ကွင်းထဲ 1 ဆိုတဲ့ index တစ်ခုရေးလိုက်တာနဲ့ သိရတယ်။ ဒါကြောင့် index တစ်ခုပဲ ရေးရလို့ one-dimensional array လို့ခေါ်ပါတယ်။
- Values တွေ ဖြစ်တဲ့ Ko Ko, Ko Let, Nyi Nyi တို့ ဆိုတာ စာသားတွေ၊ တနည်းအားဖြင့် character တွေ စုထားတာဖြစ်လို့ siblings array ဟာ String data type ဖြစ်ပါလိမ့်မယ်။

Two-Dimensional Arrays

ဒီးလှမှာ XYZ လို့ခေါ်တဲ့ စတိုးဆိုင်ခွဲပေါင်း 4 ခုရှိတယ်။ အမည်တွေကို ဖွင့်တဲ့ အစဉ်လိုက် XYZ1, XYZ2, XYZ3, XYZ4 လို့ပေးထားတယ်။ စတိုးဆိုင် တစ်ခုချင်းစီရဲ့ တစ်လချင်းစီရဲ့ အမြတ်ကို သိမ်းချင်တယ်။ ဒါဆိုရင် စတိုးဆိုင် အမှတ် ၁ ရဲ့ January လ အမြတ်ကို သိချင်တယ်ဆိုရင် XYZ1, January ဆိုပြီး ဆိုင်အမည်ကော့၊ လကော့ J ခုပြောမှ သိရမယ်။ ဒီလို့ J ခုပြောမှ သိရတာဖြစ်လို့ Two-Dimensional array လို့ခေါ်ပါတယ်။

	January	February	...	December
XYZ1	2,000,000	1,000,000		1,000,000

XYZ2	1,500,000	700,000		500,000
XYZ3	3,000,000	2,500,000		4,000,000
XYZ4	3,000,000	1,000,000		1,000,000

profits

- Array အမည် variable name ဟာ ကြိုက်တာပေးလိုရတယ်၊ ဒီမှာတော့ profits လိုပေးထားတယ်။
- Profits [XYZ1][January] လို့ ရေးရင် 2,000,000 ဆိုတာ ထွက်လာမှာဖြစ်တယ်။
- ထောင့်ကွင်းနှစ်ခု [] ကြားရေးတဲ့ ဟာကို အခန်းနံပါတ် index လို့ခေါ်တယ်၊ ဒါကြောင့် XYZ1 ရယ်၊ January ရယ် ဆိုကာက အခန်းနံပါတ် index တွေဖြစ်တယ်။
- ဒါကြောင့် တစ်ခုသိရှိ စတိုးဆိုင် အမည်ရယ်၊ လအမည်ရယ် ဆိုတဲ့ index ၂ ခုပြောမှ သိမှာဖြစ်တဲ့အတွက်ကြောင့် two-dimensional array လို့ခေါ်တယ်။ ပြောင်းပြန် ပြန်ပြောမယ် ဆိုရင်တော့ two dimensional array မှာ အထဲက value ကို လိုချင်ရင် variableName[row][column] ဆိုပြီး ပြောရပါတယ်။
- အထဲမှာ သိမ်းထားတာတွေက ဂဏန်းတွေက ကိန်းပြည့်တွေဖြစ်လို့ integer data type ပါ။ ဒဿာ တွေပါနိုင်တယ်၊ ဒဿာမပါ သိမ်းချင်တယ်ဆိုရင်တော့ profits ဆိုတဲ့ two-dimensinal array လေးဟာ double/float data type ဖြစ်သွားပါမယ်။

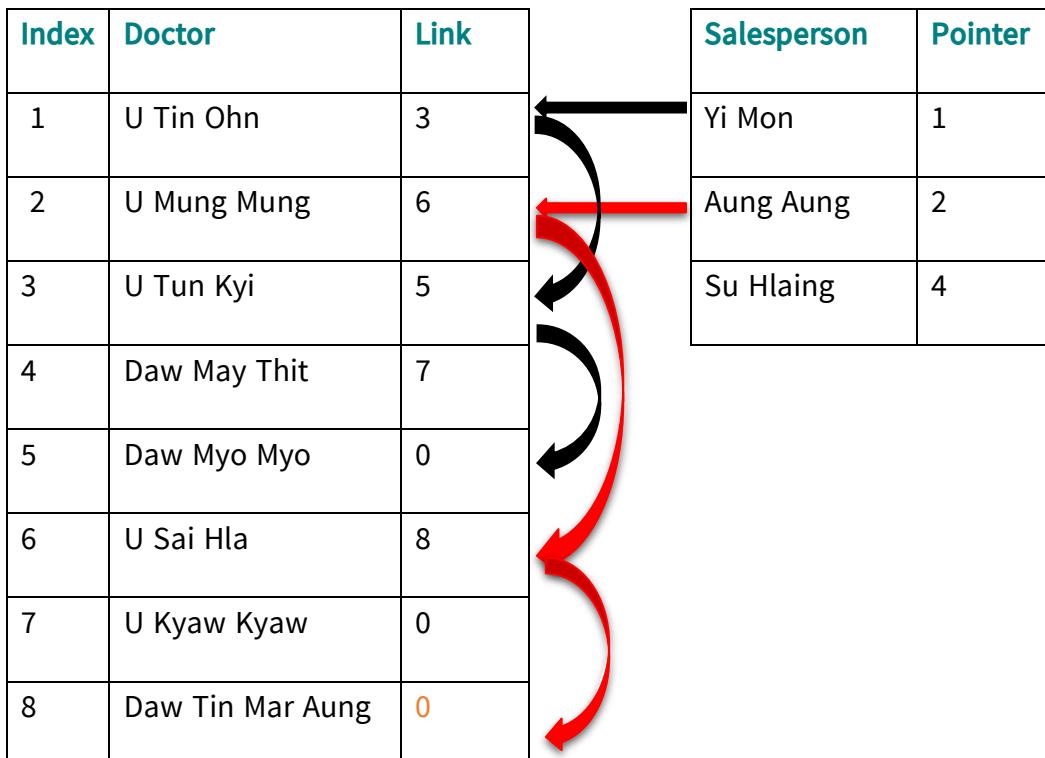
Linked Lists

ဒီတစ်ခါ ဆေး company တစ်ခုကို စဉ်းစားကြည့်ကြရအောင်ပါ။ ဆေး company မှာ အရောင်းဝန်ထမ်း (salesperson) တွေ ရှိပါတယ်။ သူတို့ဟာ သူတို့ တာဝန်ယူရတဲ့ ဆရာဝန် (doctor) တွေဆီကို company က ဆေးပစ္စည်းတွေ ရောင်းချဖို့ တာဝန်ယူရပါတယ်။ အရင် ချိတ်ဆက်မိတဲ့ ဆရာဝန်က နံပါတ်စဉ် ၁ ဖြစ်ပြီး၊ ၃ တို့ ချိတ်ဆက်မိတဲ့ ဆရာဝန်က နံပါတ်စဉ် ၂ စသည်ဖြင့် သိမ်းထားပါတယ်။

ဆိုလိုတာက ဆရာဝန်တွေကို စတင်ချိတ်ဆက်မိတဲ့ ရက်စွဲအလိုက် ငယ်စဉ်ကြီးလိုက် စီပြီး သိမ်းထားတာဖြစ်ပါတယ်။

	Doctor	Salesperson
1	U Tin Ohn	Yi Mon
2	U Mung Mung	Aung Aung
3	U Tun Kyi	Yi Mon
4	Daw May Thit	Su Hlaing
5	Daw Myo Myo	Yi Mon
6	U Sai Hla	Aung Aung
7	U Kyaw Kyaw	Su Hlaing
8	Daw Tin Mar Aung	Aung Aung

သူတို့ တာဝန်ယူထားရတဲ့ စာရင်း သိမ်းဆည်းထားတဲ့ပုံကို နမူနာ ကြည့်ပါ။ Yi Mon ဆိုတဲ့ အရောင်းဝန်ထမ်း တာဝန်ယူတဲ့ ဆရာဝန်စာရင်ကို သိချင်ရင် အားလုံးလိုက်ကြည့်ပြီး sale person မှာ Yi Mon တွေ လိုက်ရှာရပါတယ်။ မလွယ်ပါဘူး။ ဒီမှာ data နည်းလို့ (၈ ကြောင်းတည်းရှိသေးလို့) ကြည်ရတာအဆင်ပြေနေသေးတာပါ။ data တွေ အရမ်းများလာရင် (ထောင်ချီ သောင်းချီ ဖြစ်လာရင်) Yi Mon လိုက်ရှာရတာ မလွယ်ကူးပါဘူး။ ဒါဆို ရှာရလွယ်အောင် ဘယ်လို့ သိမ်းကြမလဲ၊ linked list ပုံစံနဲ့ သိမ်းပျေမယ်။ ကြည့်ကြည့်ရအောင်ပါ။



- Yi Mon တာဝန်ယူထားတဲ့ doctors စာရင်းကို ကြည့်ချင်ရင် salesperson table က Yi Mon ရဲ့ pointer တန်ဖိုးကို ကြည့်ပါ။ တန်ဖိုးက 1 ဖြစ်နေပါတယ်။ ဒီတော့ doctor table က index 1 ဖြစ်တဲ့ သူကို ကြည့်တော့ U Tin Ohn ဖြစ်ပါတယ်။ သူရဲ့ link တန်ဖိုး ကြည့်လိုက်တော့ 3 ဖြစ်နေပါတယ်။ ဒီတော့ index 3 ကို လိုက်ကြည့်ပါတယ်။ U Tun Kyi ပါ။ link တန်ဖိုး ဆက်ကြည့်ပါမယ်။ 5 ဆိုတော့ index 5 ကို သွားကြည့်ပါတယ်။ Daw Myo Myo ပါ။ သူမရဲ့ link ဆက်ကြည့်တော့ 0 ဖြစ်တဲ့ အတွက် ဆက်သွားစရာ မလိုတော့ပါဘူး။ ဒီတော့ Yi Mon တာဝန်ယူထားတဲ့ ဆရာဝန် ၃ ဦးရှိပါတယ်။ U Tin Ohn, U Tun Kyi နဲ့ Daw Myo Myo တို့ ဖြစ်ပါတယ်။ U Tin Ohn ကတော့ Yi Mon ပထမဆုံး စချင်တဲ့ ဆရာဝန်ဖြစ်ပြီး၊ ဒုတိယချင်တဲ့ ဆရာဝန်ကတော့ U Tun Kyi ဖြစ်ပါတယ်။ နောက်ဆုံး ချင်တဲ့ ဆရာဝန်ကတော့ Daw Myo Myo ပါ။ မြင်သာအောင် ပုံမှာ arrow အနက်ရောင်နဲ့ ပြထားပါတယ်။
- Aung Aung အတွက်တော့ arrow အနီးရောင်နဲ့ ပြထားပေးပါတယ်။ Aung Aung နဲ့ Su Hlaing အတွက် လေ့ကျင့်ကြည့်ပါ။

- Doctor table မှာ ပထမ column ဖြစ်တဲ့ index နဲ့ link ဟာ ကိန်းပြည့်တွေဖြစ်တာကြောင့် integer data type ဖြစ်ပြီး၊ doctor ကတော့ စာသားတွေဖြစ်တာကြောင့် string data type ဖြစ်ပါလိမ့်မယ်။ Salesperson table ကိုတော့ မိမိဟာ မိမိ ဘာ data type တွေ ဖြစ်မလဲဆိုတာ စဉ်းစားကြည့်ပါ။



လေ့ကျင့်ကြည့် ကြရအောင်။

တကယ်လို့ ဆရာဝန်တွေနဲ့ စတင်ချိတ်ဆက်တဲ့ရက် ကိုပါသိမ်းချင်ရင် ဘယ်လို့ လုပ်မလဲ စဉ်းစားကြည့်ပါ။ အောက်ကစာ ဆက်မဖတ်ပါနဲ့အုံး၊ စဉ်းစားပြီးမှ ဆက်ဖတ်ပါ။

ဆရာဝန်တွေနဲ့ စတင်ချိတ်ဆက်တဲ့ရက် ဆိုတာ Doctor နဲ့ သက်ဆိုင်တဲ့ အချက်လက်ဖြစ်တဲ့အတွက် Doctor table မှာ ထပ်ထည့်မှာပါ။ Salesperson table ကတော့ ဒီအတိုင်း ရှိနေမှာပါ။

Index	Doctor	Join_Date	Link

salesperson တစ်ဦးချင်းစီ အလုပ်ဝင်တဲ့ နေ့ကိုပါသိမ်းချင်တယ်ဆိုရင် ဘယ်လို့ လုပ်မလဲ စဉ်းစားကြည့်ပါ။ ထုံးစံအတိုင်း အောက်ကစာ ဆက်မဖတ်ပါနဲ့အုံး၊ စဉ်းစားပြီးမှ ဆက်ဖတ်ပါ။

Salesperson နဲ့ သက်ဆိုင်တာဖြစ်လို့ Salesperson table မှာ သွားထည့်မှာ ဖြစ်ပါမယ်။

Salesperson	Start_Date	Pointer



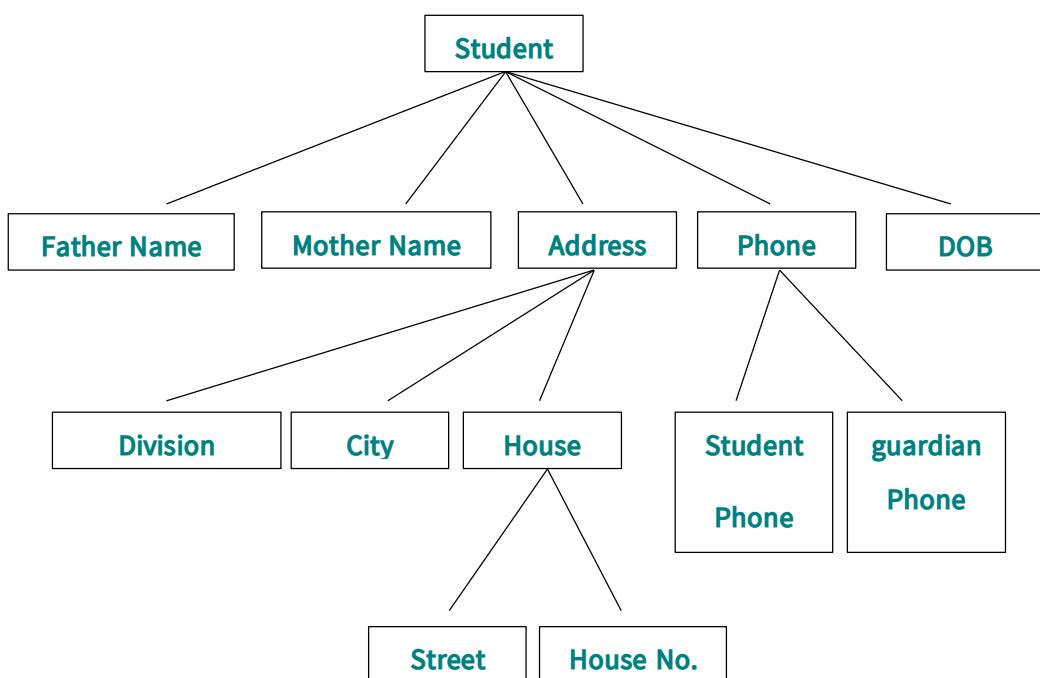
ဒါဆို အလုပ်စဝင်တဲ့ နောက်၊ ဆရာဝန်နဲ့ စချိတ်တဲ့ နောက် သိမ်းချင်ရင် ဘယ်လိုလုပ်မလဲ။

အလုပ်စဝင်တဲ့ နောက် Salesperson နဲ့ ဆိုင်တာမှို့ Salesperson table မှာ ထည့်ပြီး၊ ဆရာဝန်နဲ့ စချိတ်တဲ့ နောက် ဆရာဝန်တစ်ဦးချင်းစီနဲ့ ဆိုင်တာမှို့လို့ Doctor table ဆရာဝန်ရဲ့ ဘေးမှာ ထည့်ပါမယ်၊ ကဲဒီလောက်ဆိုရင် သဘော ပေါက်လောက်ပြီထင်ပါတယ်။



အခုပြောသွားတဲ့ linked list နမူနာကို အလွယ်ပြောရရင် one to many relationship လိုပြောလိုရပါတယ်။ Salesperson တစ်ယောက်ဟာ ဆရာဝန်တွေ အများကြီးနဲ့ စချိတ်ဆက်ထားတဲ့ အတွက် one to many relationship ပါ။ အဲဒီလို အခြေနေမျိုးအတွက် linked list ကို အသုံးပြုတတ်ပါတယ်။

Tree Structure or Hierarchical Structure



Hierarchical ဆိုတဲ့ အတိုင်း အဆင့်ဆင့် အထပ်ထပ် ခွဲခဲ့သွားတဲ့ ပုံစံနဲ့ သိမ်းချင်တဲ့ အခါမျိုးမှာ သုံးပါတယ်။ သူကို မြင်သာအောင် ပြရ ရင်တော့

01 Student

02 Father Name

02 Mother Name

02 Address

03 Division

03 City

03 House

04 Street

04 House No.

02 Phone

03 Student Phone

03 Guardian Phone

02 DOB

- Tree မှာပါတဲ့ Student တို့ Father Name, Mother Name စသည်ဖြင့် ပါတဲ့ အရာလေးတွေကို Node လို့ ခေါ်ပါတယ်။
- Father Name, Mother Name, Address, Phone, DOB တို့ဟာ student ကနေ ခွဲထားတာဖြစ်လို့ Student ရဲ့ child တွေ လို့ ခေါ်ပါတယ်။ ပြောင်းပြန် ပြန်ပြောရမယ် ဆိုရင်တော့ Student ကို Father Name, Mother Name, Address, Phone, DOB တို့ရဲ့ parent လို့ ခေါ်ပါတယ်။
- Student ဆိုတာ ထိပ်ဆုံးမှာ ရှိပြီး သူအထက်ဘယ်သူမှ မရှိတော့လို့ တန်ညွှေးအားဖြင့် သူမှာ parent မရှိလို့ Student ကို root node လို့ခေါ်ပါတယ်။

- Father Name, Division, Street, တို့ဟာ child မရှိတော့လို သူတို့ကို leaf node လို ခေါပါတယ်။



ဒီမှာ နောက်ထပ် leaf node တွေ ကျန်ပါသေးတယ်။ လိုက်ရှာကြည့်ကြည့်ပါ။



Chapter အနှစ်ချုပ်

- ဒီ အခန်းမှာ တန်ဖိုးတစ်ခုတည်းသိမ်းဆည်းတဲ့ နည်း နှင့် တန်ဖိုးတွေ စုစည်း သိမ်းဆည်းတဲ့ နည်း အချိုက် ပြောသွားတာဖြစ်ပါတယ်။
- File ဆိုတာ non-volatile ဖြစ်တဲ့ secondary memory ပေါ်မှာ သိမ်းဆည်းတာဖြစ်လို စက်ပိတ်လည်း data တွေကို ဆက်ပြီး သိမ်းထားနိုင်ပါတယ်။
- ကျန်တဲ့ သိမ်းဆည်းနည်းတွေကတော့ စက်ပိတ်လိုက်ရင် program ပိတ်လိုက်ရင် ပျောက်သွားတဲ့ volatile memory ပေါ်မှာ သိမ်းဆည်းတာဖြစ်ပါတယ်။
- Fields or attributes or properties ဆိုတာ column name, အဲဒီအထဲ ထည့်တဲ့ တန်ဖိုးတွေက value, အဲဒီ value တွေကို row လိုက်စုပြီး တစ်ဦးတစ်ယောက်၊ တစ်ခုကို ညွှန်းတာက record, အဲဒီလို records တွေ စုထားတာက File ဖြစ်ပါတယ်။
- Primary key ဆိုတာ unique ဖြစ်တဲ့ attribute ကို ပြောတာဖြစ်ပါတယ်။ အဲဒီ attribute ရဲ value ကို ပြောလိုက်တာနဲ့ ဘယ်သူလဲ ဘာကိုပြောတာလဲ တန်းပြီး သိတာမျိုးဖြစ်ပါတယ်။ primary key ဟာ attribute တစ်ခု ဖြစ်နိုင်သလို၊ တစ်ခုထက်ပိုတဲ့ attributes တွေ ပေါင်းစပ်ထားတာလည်း ဖြစ်နိုင်ပါတယ်။
- One-dimensional array ဆိုတာ index (အခန်းနံပါတ်) တစ်ခုပဲလိုပါတယ်။

- Tow-dimensional array ဆိုတာ index 2 ခုလိုပါတယ်။ index 3 ခုလိုရင် three-dimensional array လိုခေါ်ပါတယ်။ 2 dimensional array နဲ့ အထက်ကို multi-dimensional array လိုခေါ်ပါတယ်။
- One to many relationship (တစ်ခုကနေ အများသို့ ဆက်စပ် ချိတ်ဆက်မှု) အတွက် Linked list ကို သုံးတာ ပြောခဲ့ပါတယ်။ many to many relationship (အများမှ အများသို့ ဆက်စပ် ချိတ်ဆက်မှု) ကိုတော့ နောက်အခန်းတွေမှာ အခြေအနေပေးရင် ဆက်ပြောပါမယ်။
- အဆင့်ဆင့် အထပ်ထပ် ခွဲခွဲသွားတဲ့ ပုံစံနဲ့ သိမ်းချင်တဲ့ အခါမျိုးမှာ tree or hierarchical structure ကို သုံးပါတယ်။
- နောက်ပြီး စုစည်းသိမ်းဆည်းတာ ရဲ့ အစိတ်အပိုင်း တစ်ခုချင်းကို data type တွေ သတ်မှတ်ခိုင်းတာ သတိထားမိမယ် ထင်ပါတယ်။ ဒါဆို primitive data type int ဆိုရင် ကိန်းပြည့် တစ်လုံးပဲ သိမ်းရတယ်၊ primitive data type double ဆိုရင် ဒသေမ ဂဏန်း တစ်လုံးသိမ်းလိုရမယ်၊ primitive data type char ဆိုရင် character တစ်လုံး သိမ်းပဲ သိမ်းလိုရမယ်။ String ဆိုရင် character တွေ အတွဲလိုက်ကြီး သိမ်းလိုရမယ်။ စုစည်းသိမ်းဆည်းတာတစ်ခုမှာ primitives data type တွေရော string လိုမျိုးကော့ အများကြီး စုပြီး သိမ်းဆည်းတယ်ဆိုတာ သဘောပေါက်လောက်ပါပြီ။

အခန်း (၁၁)

String Operations

String ဆိတာဘာလဲ။

String ဆိတာ character တွေ စုထားတာ၊ String တစ်ခုမှာ character တွေ ကြိုက်သလောက်ပါလို့ရတယ်ဆိတာ ပြောခဲ့ပြီးပါပြီ။ နောက်တစ်ခုက တူညီတဲ့ data type တွေ စုထားတာကို array လို ခေါပါတယ်။ ဒါကြောင့်လဲ string ကို character array လိုလည်း ပြောလိုရပါတယ်။ Array အခန်းနံပါတ်သည် ပုံမှန် ဆိုရင် 0 ကနေ စပါတယ်။ ဒါကြောင့် String ဟာလည်း အခန်းနံပါတ် 0 ကနေ စပါတယ်။

“I CAN FLY” ဆိုတဲ့ string လေးကို array ပုံစံမြင်ကြည့်မယ်ဆိုရင်

I		C	A	N		F	L	Y
0	1	2	3	4	5	6	7	8

ဒီလိုပုံစံ တွေမြင်ရမှာ ဖြစ်ပါတယ်။ 0 အခန်းက I, 1 အခန်းမှာက space, 2 အခန်းမှာက C စသည်ဖြင့် နေရာယူသွားပါတယ်။ အခန်းနံပါတ်ဟာ 0 to 8 ဖြစ်လို့ array ရဲ့ length (စုစုပေါင်းအခန်းအရေအတွက်) ဟာ 9 ဖြစ်ပါမယ်။

Length

Length ဆိတာ string မှာ character ဘယ်နှစ်လုံးပါလဲ ပြောတာ သိလောက်ပါပြီ။

LENGTH(“life is a song”) = 14

LENGTH(“mandalay”) = 8

Upper Case and Lower Case

toUpperCase(string) – string ကို အကြီးစာလုံး upper case ကို ပြောင်းတာဖြစ်ပါတယ်။

toLowerCase(string) – string ကို အသေးစာလုံး lower case ကို ပြောင်းတာဖြစ်ပါတယ်။

toUpperCase(“I can fly”) = “I CAN FLY”

toLowerCase(“I Can fly”) = “i can fly”

Substring

Substring ဆိုတာ String ထဲကနေ String အပိုင်းအစလေးကို ဖြတ်ယူလိုက်တာ ဖြစ်ပါတယ်။

SUBSTRING (string, initial, length)

SUBSTRING ဆိုတဲ့ function မှာ parameter 3 ခုလိုက်ပါတယ်။ ပထမ တစ်ခုက မူရင်း string, initial ဆိုတာ စမည့် ခန်းနံပါတ်၊ length ဆိုတာ အလုံးအရေအတွက် ဖြစ်ပါတယ်။

SUBSTRING(“I CAN FLY”, 2, 3) = CAN

SUBSTRING(“I CAN FLY”, 1, 4) = _CAN (spaceCAN ဆိုပြီး ရမှာဖြစ်ပါတယ်။)

တခို့ language တွေမှာတော့ length အစား၊ end ဖြစ်တတ်ပါတယ်။

SUBSTRING (string, initial, end)

SUBSTRING("I CAN FLY", 2, 5) = CAN

initial က 2 ဖြစ်တဲ့အတွက် 2 အခန်းကနေ စတယ်၊ end က 5 ဖြစ်တဲ့အတွက် 5 မရောက်ခင် 4 အခန်းထိဖြတ်သွားတာ ဖြစ်ပါတယ်။ဒါလေးကသိအောင်ပြောသွားတာပါ။

Indexing

INDEX(String, pattern)

INDEX("I am the captain of my soul", " the") = 4

“ the” ကိုလိုက်ရှာပါတယ်။ တွေ့တယ် တွေ့တော့ “ the” ရဲ့ ပထမဆုံး character ဖြစ်တဲ့ space တွေ့တဲ့ အခန်းနံပါတ် 4 ကို return ပြန်ပေးထားဖြစ်ပါတယ်။

INDEX("I am the captain of my soul", "hello") = -1

array အခန်းနံပါတ်ဟာ 0 ကစပြီး ကြီးကြီးသွားတာဖြစ်လို့ မတွေ့ရင် မရှိတဲ့ အခန်းနံပတ် -1 ထွက်တာဖြစ်ပါတယ်။

INDEX("I am a student", "a") = 2

a ကနှစ်နေရာပါပါတယ်၊ ဘယ်နှစ်နေရာပဲပါပါ INDEX function ဟာ အမြဲ 0 ခန်းကနေစပြီးရှာတယ်။ ပြီးရင်ပထမဆုံးတွေ့တဲ့ နေရာကို return ပြန်ပေးမှာဖြစ်ပါတယ်။

INDEX(String, pattern, fromIndex)

Parameter 3 ခဲ့လက်ခံတဲ့ Index Function ဖြစ်ပါတယ်။ ဘယ်အခန်းကနေ စပြီးရှာမလဲဆိုတဲ့ fromIndex တိုးလာတာဖြစ်ပါတယ်။

```
INDEX("I am a student", "a",0) = 2
```

```
INDEX("I am a student", "a", 3) =5
```

LASTINDEX(String, pattern)

LASTINDEX function ဟာ INDEX function နဲ့ အလုပ်လုပ်ပုံ အတူတူပါပဲ။ တစ်နေရာပဲကဲ့ပါတယ်။ အဒါကတော့ ထပ်ခါထပ်ခါ ပါနေတဲ့အချိန်ဆို INDEX က ပထမဆုံးတွေ့တဲ့ နေရာကို return ပြန်ပေးပြီး၊ LASTINDEX ကတော့ နောက်ဆုံးတွေ့တဲ့ နေရာကို return ပြန်ပေးပါတယ်။

```
LASTINDEX("I am a student", "a") = 5
```

a ကနှစ်နေရာပါပါတယ်။ ဘယ်နှစ်နေရာပဲပါပါ LASTINDEX function ဟာ နောက်ဆုံးတွေ့တဲ့ နေရာကို return ပြန်ပေးမှာဖြစ်ပါတယ်။

Concatenation***CONCAT(String1, String2)***

Concatenation ဆိုတာက String နစ်ခုကို ဆက်တာဖြစ်ပါတယ်။

```
CONCAT("Life is beauty" , " admire it.") = "Life is beauty admire it"
```

Insertion

INSERT(String, position, String2)

Insert function မှာ parameter 3 ခုပါပါတယ်။ မူရင်း string ရယ်၊ ထည့်ချင်တဲ့ နေရာရယ်၊ ထည့်ချင်တဲ့ စာသား string ရယ် ဖြစ်ပါတယ်။

```
INSERT("Hello Mandalay", 5, "! welcome to") = "Hello! Welcome to Mandalay"
```

5 ဆိုတဲ့ position နေရာဆိုတော့ Hello ရဲနောက်ကပ်ရက်မှာ “welcome to” ဆိုတဲ့ စာသားကို ထည့်သွားတာဖြစ်ပါတယ်။

```
INSERT("ABCDEFGH", "XYZ",4) = "ABCDXYZEFGH"
```

Deletion

DELETE(String, position, length)

Insert function မှာ parameter 3 ခုပါပါတယ်။ မူရင်း string ရယ်၊ စဖျက်ချင်တဲ့ နေရာရယ်၊ ဖျက်မည့်အလုံးအရေအတွက်ရယ် ဖြစ်ပါတယ်။

```
DELETE("ABCDEFG", 3,2 ) = "ABCfg"
```

“ABCDEFGHIJKLMNO” ထဲက “FGH” ကို ရှာပြီးဖျက်ချင်တယ်ဆိုရင်

“FGH” တွေတဲ့နေရာကနေ စပြီး ဖျက်မှာဖြစ်လို့ position နေရာမှာ INDEX(“ABCDEFGHIJKLMNO”, “FGH”)

“FGH” ဆိုတော့ သူမှာပါတဲ့ အလုံးအရေအတွက် ဖျက်မှာဖြစ်လို့ LENGTH(“FGH”)

ဒါကြောင့်

DELETE("ABCDEFGHIJKLMNO", INDEX("ABCDEFGHIJKLMNO" , "FGH"), LENGTH("FGH"))

လိုပေးလိုပါတယ်။

Trace လိုက်ပြရရင်

INDEX("ABCDEFGHIJKLMNO" , "FGH") = 5

LENGTH("FGH") = 3

DELETE("ABCDEFGHIJKLMNO", 5, 3) = "ABCDEIJKLMNO"

Replacement

REPLACE(String, oldstr, newstr)

String ရဲ့ oldstr တွေနေရာမှာ newstr ကို အစားထိုးမှာ ဖြစ်ပါတယ်။

REPLACE("ABCDABE", "AB", "TTT") = "TTTCDTTTE"

"ABCDABE" ထဲက "AB" တွေ့သမျှနေရာတွေအားလုံးမှာ "TTT" ကို အစားထိုးလိုက်တာ ဖြစ်ပါတယ်။



လေ့ကျင့်ကြည့် ကြရအောင်။

1.text = "Life is beauty, admire it. Life is a dream, realize it. Life is a challenge, meet it."

Str ="Life"

a) Length (text) = ?

b) Length(Str) =?

c) "challenge, meet it." ဆိုတာလေးကို substring ကို အသုံးပြုပြီး text ထဲမှ ဆွဲထုတ်ပါ။

- d) text ထဲက Str ကိုထွေးတဲ့ ပထမဆုံးအခန်းနံပါတ်နဲ့ နောက်ဆုံးအခန်းနံပါတ်ကို သင့်တော်တဲ့ functions များသုံးပြီး ရှာပါ။
- e) INDEX(text, "TO") = ?

2. Word ဆိုတဲ့ စကားလုံး တစ်လုံးရှိမယ်။ အဲဒီထဲကို ထည့်ချင်တဲ့ စာသား ထည့်ထားမယ်။ အဲဒီ word ကို infinitive ပုံစံပြောင်းချင်တယ်။ နမူနာလေးတွေ တစ်ချက်ကြည့်ကြည့်ပါ။

Works => work, Goes => go, Flies => fly, Worked => work, Working => work

ဆိုလိုတာက s, es, ies, ing, ed ဆုံးခဲ့ရင် ဖြုတ်တာဖြစ်ပါတယ်။ ဆုံးခဲ့ရင် ဆိုတော့ conditional statement သုံးဖို့လိုပါတယ်။ ies ဆိုရင် ies ကို ဖြုတ်ပြီး y ထည့်ပေးရပါတယ်။ ပုံမှန်မဟုတ်တဲ့ irregular verb တွေအတွက်တော့ ထည့်မစဉ်းစားပါနဲ့။ အပေါ်မှာပြောခဲ့တဲ့ function တွေကိုသုံးပြီး algorithm တစ်ခုရေးရမှာ ဖြစ်ပါတယ်။

အဲဒီ algorithm ဟာ word ဆိုတဲ့ string ကို parameter အဖြစ် လက်ခံမှာဖြစ်ပြီး ဖြုတ်စရာရှိတာ ဖြုတ်ပြီးသား string ကို return ပြန်ပေးမှာ ဖြစ်ပါတယ်။

Delete Every Occurrence Algorithm

မူရင်း string ထဲက သူရှာချင်တဲ့ string ကို ရှာပြီး တွေ့ရာနေရာ အကုန် ဖျက်မှာဖြစ်ပါတယ်။

Algorithm: Delete Every Occurrence

```
/*delete every occurrence of str in Text*/
```

1. deleteEveryOccurrence(Text, str)
2. I = INDEX(Text, str)
3. Repeat while I!= -1:
4. Text = DELETE(Text, INDEX(Text, str), LENGTH(str))
5. I = INDEX(Text, str)
6. End of Loop
7. Return Text

ထိုးစံအတိုင်း trace လိုက်ပါမယ်။

```
removeText = deleteEveryOccurrence ("ABC DAB E FAB GH", "AB")
```

Line	Text	Str	Index(i)	အလုပ်လုပ်ပုံ
1	ABC DAB E FAB GH	AB		
2			0	I = INDEX (Text,Str) = 0
3				I != -1 => 0 != -1 ဆိုတော့မှန်တယ်၊ line 4 ကို လုပ်မယ်။
4	CDABEFABGH			Text = DELETE(Text, INDEX(Text, str), LENGTH(str)) Text = DELETE("CDABEFABGH", 0, 2) Text= CDABEFABGH

5		2	<p>$I = \text{INDEX}(\text{Text}, \text{str})$</p> <p>$I = \text{INDEX}("CDABEFABGH", "AB")$</p> <p>$I = 2$ ရပါတယ်။ အမှန်က while condition မှာ</p> <p>$\text{index}(i) = 2$ ဖြစ်ပါသည့်အတွက် $I = 2$ ကိုတွေ့တဲ့ index (i) ကို</p> <p>update လုပ်ထားတာဖြစ်ပါတယ်။</p>
6			<p>End of Loop ဆိုတော့ while condition</p> <p>ရေးထားတဲ့ line 3 ကို ပြန်တက်ပါမယ်။</p>
3			<p>$I \neq -1 \Rightarrow I = 2$ ဆိုတော့မှန်တယ်၊ line 4 ကို</p> <p>လုပ်မယ်။</p>
4	CDEFABGH		<p>$\text{Text} = \text{DELETE}(\text{Text}, \text{INDEX}(\text{Text}, \text{str}), \text{LENGTH}(\text{str}))$</p> <p>$\text{Text} = \text{DELETE}("CDEFABGH", 2, 2)$</p> <p>$\text{Text} = \text{CDEFABGH}$</p>
5		4	<p>$I = \text{INDEX}(\text{Text}, \text{str})$</p> <p>$I = \text{INDEX}("CDEFABGH", "AB")$</p> <p>$I = 4$ ရပါတယ်။ အမှန်က while condition မှာ</p> <p>$\text{index}(i) = 4$ ဖြစ်ပါသည့်အတွက် $I = 4$ ကိုတွေ့တဲ့ index (i) ကို</p> <p>update လုပ်ထားတာဖြစ်ပါတယ်။</p>
6			<p>End of Loop ဆိုတော့ while condition</p> <p>ရေးထားတဲ့ line 3 ကို ပြန်တက်ပါမယ်။</p>

3				I != -1 => 4 != -1 ဆိုတော့မှန်တယ်၊ line 4 ကို လုပ်မယ်။
4	CDEFGH			Text = DELETE(Text, INDEX(Text, str), LENGTH(str)) Text = DELETE("CDEFABGH", 4, 2) Text= CDEFGH
5		-1		I = INDEX(Text, str) I = INDEX("CDEFGH", "AB") မတွေ့တော့တဲ့ အတွက် I = -1 ရပါတယ်။
6				End of Loop ဆိုတော့ while condition ရေးထားတဲ့ line 3 ကို ဖြန်တက်ပါမယ်။
3				I != -1 => -1 != -1 ဆိုတော့ condition မှားသွားပြီ ဖြစ်လို့ end of loop အပြင်ဖက် line 7 ကို လုပ်မယ်။
7				Return "text" ဆိုတော့ return "CDEFGH" ဆိုပြီး လုမ်းချေတဲ့ နေရာကို အဖြေဖြန်ပိုလိုက်ပါတယ်။

Pattern Matching

Text ထဲကနေ str ကို လိုက်ရှာရင် ဘယ်လိုရှာလဲ တစ်ချက်ကြည့်ကြည့်ရအောင်။

Text = "abcdefg"

str = "dez"

လို ဆိုခဲ့ရင် ရှာချင်တဲ့ str ဟာ length ၃ ရှိပါတယ်။ သူ့ရဲ့ တိုက်တဲ့ ပုံလေး တစ်ချက်
ကြည့်ကြည့်ရအောင်

Str	Start Index	Text ရဲ substring	ရှင်းလင်းချက်	တိုက်ရတဲ့ အကြိမ်
dez	0	abc	Text ရဲ 0 ခန်းကနေ စပြီးတိုက်မယ်။ dez length အတိုင်း ၃ လုံးဖြတ်ယူတယ်။ ရှုံးဆုံး character 2 လုံးဖြစ်တဲ့ d နဲ့ a ကိုတိုက်တယ်။ မတူတော့ ဆက်မတိုက်တော့ဘူး။ ၁ကြိမ်ပဲ တိုက်လိုက်ရတယ်။	1
dez	1	bcd	Text ရဲ 1 အခန်းကနေစပြီး တိုက်မယ်။ ရှုံးဆုံး character 2 လုံးဖြစ်တဲ့ d နဲ့ b ကိုတိုက်တယ်။ မတူတော့ ဆက်မတိုက်တော့ဘူး။ ၁ကြိမ်ပဲ တိုက်လိုက်ရတယ်။	1
dez	2	cde	Text ရဲ 2 အခန်းကနေစပြီး တိုက်မယ်။ ရှုံးဆုံး character 2 လုံးဖြစ်တဲ့ d နဲ့ c ကိုတိုက်တယ်။ မတူတော့ ဆက်မတိုက်တော့ဘူး။ ၁ကြိမ်ပဲ တိုက်လိုက်ရတယ်။	1
dez	3	def	Text ရဲ 3 အခန်းကနေစပြီး တိုက်မယ်။ ရှုံးဆုံး character 2 လုံးဖြစ်တဲ့ d နဲ့ d ကိုတိုက်တယ်။ တူတော့ ဆက်တိုက်မယ်။ ဒုတိယ character 2 လုံးဖြစ်တဲ့ e နဲ့ e ကိုတိုက်တယ်။ တူတော့ ဆက်တိုက်မယ်။ တတိယ character 2 လုံးဖြစ်တဲ့ z နဲ့ e ကိုတိုက်တယ်။ မတူတော့ မတွေ့ဘူးဖြစ်သွားတယ်။	3

			3 ကြိမ်တိက်လိုက်ရတယ်။	
Dez	4	efg	Text ရဲ့ 4 အခန်းကနေစပီး တိုက်မယ်။ ရှေ့ဆုံး character 2 လုံးဖြစ်တဲ့ d နဲ့ e ကိုတိုက်တယ်။ မတူတော့ ဆက်မတိုက်တော့ဘူး။ ၁ကြိမ်ပဲ တိုက်လိုက်ရတယ်။	1

အခန်း 5 ကစပီး ယူမယ်ဆိုရင် Text မှာ J လုံးပဲ ကျေန်တယ်။ ဒီတော့ length 3 ရှိတဲ့ str နဲ့ တူစရာ အကြောင်းမရှိတော့လို ရပ်လိုက်တယ်။ ဘယ်အခန်းထိ တိုက်မလဲ သိချင်ရင်

$\text{Length}(\text{Text}) - \text{Length}(\text{str}) = 7 - 3 = 4$ အခန်းထိပဲ တိုက်မှာ ဖြစ်ပါတယ်။

စုစုပေါင်း ဘယ်နှစ်ကြိမ်တိုက်ရသလဲ ဆိုတာ trace table ရဲ့ နောက်ဆုံး column က တန်ဖိုးတွေ ပေါင်းကြည့်ရအောင်

$1 + 1 + 1 + 3 + 1 = 7$ ကြိမ်တိုက်ပြီးမှ မတွေ့ဘူး ဆိုတဲ့ အဖြေထွက်ပါတယ်။

နောက်တစ်ခု တွက်ကြည့်ရအောင်။ အရည်ရှင်းမပြတော့ပါဘူး။

Text = “ababaaba.....”

Str = “aaba”

တိုက်တဲ့ အကြိမ် စုစုပေါင်း = $2 + 1 + 2 + 1 + 4 = 10$ ဖြစ်ပါတယ်။ Text ရဲ့ အခန်းနံပါတ် 4 ကနေ စတိုက်တဲ့ အချိန်မှာ တွေ့သွားတာဖြစ်လို ဆက်တိုက်စရာ မလိုတော့ပါဘူး။



တိုက်တဲ့ အကြိမ်အရေအတွက် တွက်ကြည့်ကြရအောင်။

1. Text = $(cd)^{10}$, str = “aaba” ($(cd)^{10}$ ဆိုတာ “cdcdcdcdcdcdcdcdcd” ဆိုပြီး cd ၁၀ ခါရေးတာ ဖြစ်ပါတယ်။)
2. Text = $(cd)^{10}$, str = “cdc” ($(cd)^{10}$ ဆိုတာ “cdcdcdcdcdcdcdcdcd” ဆိုပြီး cd ၁၀ ခါရေးတာ ဖြစ်ပါတယ်။)
3. Text = a^{20} , str = aaaaae

Word Count Algorithm

မူရင်း string Text ထဲက သူရှာချင်တဲ့ string, str ဘယ်နှစ်ကြိမ်ပါလဲ ရေတွက်မှာဖြစ်ပါတယ်။

Algorithm: Word Count

-
- ```
/*Count the occurrence of str in Text*/
1. countOccurrence(Text, str)
2. Set count = 0
3. Text = toUpper(Text) /*can also use toLower(Text)
4. str = toUpper(str)
5. Set I = INDEX(Text, str,0)
6. Repeat while !=-1:
7. count = count + 1
8. I = INDEX(Text, str, I + 1)
9. End of Loop
10. Return count
```
- 

ထုံးစံအတိုင်း trace လိုက်ပါမယ်။

Time = countOccurrence("Life is beauty, admire it. Life is a dream, realize it. Life is a challenge, meet it.", "life")

| Line | အလုပ်လုပ်ပုံ                                                                                                                                                                                                                                                                                                                                                                                                                                                        | Count |
|------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| 1    | Text = "Life is beauty, admire it. Life is a dream, realize it. Life is a challenge, meet it."<br><br>str = "life"                                                                                                                                                                                                                                                                                                                                                  |       |
| 2    | count = 0                                                                                                                                                                                                                                                                                                                                                                                                                                                           | 0     |
| 3    | Text = toUpper(Text)<br><br>Text = "LIFE IS BEAUTY, ADMIRE IT. LIFE IS A DREAM, REALIZE IT. LIFE IS A CHALLENGE, MEET IT."<br><br>ဒီနေရာမှာ toLower(Text) ကိုသုံးလည်းရပါတယ်။ အခိကကတော့ Text နဲ့ str တူသွားဖို့ပဲလိုပါတယ်။ တစ်ခုကို အကြီးပြောင်းရင် ကျန်တစ်ခုလည်း အကြီးပြောင်းပါ။ တစ်ခုကို အသေးပြောင်းရင် ကျန်တစ်ခုကိုလည်း အသေးပြောင်းပါ။ ဒါမှ နှစ်ခု တူမှာပါ။ မဟုတ်ရင် စာလုံးချင်းတော့တူတယ် သို့သော် တစ်ခုက အသေး၊ တစ်ခုက အကြီးဆို မတူဘူး ရှာမတွေ့ဘူး ဖြစ်သွားပါမယ်။ |       |
| 4    | str = toUpper(str)<br><br>str = toUpper("life") = LIFE<br><br>Text ကို အကြီးပြောင်းထားလို့ သူကိုလည်း အကြီးပြောင်းပါတယ်။                                                                                                                                                                                                                                                                                                                                             |       |
| 5    | I = INDEX (Text, str) = 0                                                                                                                                                                                                                                                                                                                                                                                                                                           |       |
| 6    | I != -1 -> 0 != -1 ဆိုတော့ condition မှန်တဲ့အတွက် while ရဲ့ body ကို အလုပ်လုပ်ပါမယ်။                                                                                                                                                                                                                                                                                                                                                                                |       |

|   |                                                                                                                                                                                                                                                                                                                 |  |
|---|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| 7 | <p>Line 6 မှာ index (I) ဟာ -1 နဲ့ မညီလို ဝင်လာတာဖြစ်လို 1</p> <p>တစ်ကြိမ်တွေတဲ့အတွက် count ကို ၁ တိုးပါတယ်။</p> <p>count = count + 1</p>                                                                                                                                                                        |  |
| 8 | <p>I = INDEX (Text, str, I+1)</p> <p>0 ခန်းမှာ တွေ့တာကို count လုပ်ပြီးပြီဖြစ်လို သူရဲ့ နောက်တစ်ခန်း 1 အခန်းကနေ စပြီး ရှာမှာ ဖြစ်ပါတယ်။</p> <p>I = INDEX(Text, str, 1) =27</p> <p>ဒါက while loop ရဲ့ update ဖြစ်ပါတယ်။ ဘာကြောင့်လဲဆိုရင် while loop မှာ စစ်ထားတဲ့   တန်ဖိုး ပြောင်းပေးလိုက်တာ ဖြစ်လိုပါ။</p>    |  |
| 9 | End of loop ဆိုတော့ while ရဲ့ condition ကို ပြန်တက်ပါမယ်။                                                                                                                                                                                                                                                       |  |
| 6 | I != -1 -> 27 != -1 ဆိုတော့ condition မှန်တဲ့အတွက် while ရဲ့ body ကို အလုပ်လုပ်ပါမယ်။                                                                                                                                                                                                                           |  |
| 7 | <p>Line 6 မှာ index (I) ဟာ -1 နဲ့ မညီလို ဝင်လာတာဖြစ်လို 2</p> <p>တစ်ကြိမ်တွေတဲ့အတွက် count ကို ၁ တိုးပါတယ်။</p> <p>count = count + 1</p>                                                                                                                                                                        |  |
| 8 | <p>I = INDEX (Text, str, I+1)</p> <p>27 ခန်းမှာ တွေ့တာကို count လုပ်ပြီးပြီဖြစ်လို သူရဲ့ နောက်တစ်ခန်း 28 အခန်းကနေ စပြီး ရှာမှာ ဖြစ်ပါတယ်။</p> <p>I = INDEX(Text, str, 28) =56</p> <p>ဒါက while loop ရဲ့ update ဖြစ်ပါတယ်။ ဘာကြောင့်လဲဆိုရင် while loop မှာ စစ်ထားတဲ့   တန်ဖိုး ပြောင်းပေးလိုက်တာ ဖြစ်လိုပါ။</p> |  |
| 9 | End of loop ဆိုတော့ while ရဲ့ condition ကို ပြန်တက်ပါမယ်။                                                                                                                                                                                                                                                       |  |

|    |                                                                                                                                                                                                                                                                                                          |  |
|----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| 6  | I != -1 -> 56 != -1 ဆိုတော့ condition မှန်တဲ့အတွက် while ရဲ့ body ကို အလုပ်လုပ်ပါမယ်။                                                                                                                                                                                                                    |  |
| 7  | Line 6 မှာ index (I) ဟာ -1 နဲ့ မညီလို ဝင်လာတာဖြစ်လို 3 တစ်ကြိမ်တွေတဲ့အတွက် count ကို ၁ တိုးပါတယ်။<br><br>count = count + 1                                                                                                                                                                               |  |
| 8  | I = INDEX (Text, str, I+1)<br><br>56 ခန်းမှာ တွေ့တာကို count လုပ်ပြီးပြီဖြစ်လို သူရဲ့ နောက်တစ်ခန်း 57 အခန်းကနေ စပြီး ရှာမှာ ဖြစ်ပါတယ်။<br><br>I = INDEX(Text, str, 57) =-1<br><br>ဒါက while loop ရဲ့ update ဖြစ်ပါတယ်။ ဘာကြောင့်လဲဆိုရင် while loop မှာ စစ်ထားတဲ့ ၅ တန်ဖိုး ပြောင်းပေးလိုက်တာ ဖြစ်လိုပါ။ |  |
| 9  | End of loop ဆိုတော့ while ရဲ့ condition ကို ပြန်တက်ပါမယ်။                                                                                                                                                                                                                                                |  |
| 6  | I != -1 -> -1 != -1 ဆိုတော့ condition မှားသွားတဲ့အတွက် end of loop အပြင်ဖက် line 10 ကိုသွားပါမယ်။                                                                                                                                                                                                        |  |
| 10 | Return count ဆိုတော့ return 3 ဆိုပြီး function call ခေါ်တဲ့ နေရာကို return ပြန်ပြီး function ထဲက ထွက်သွားမှာ ဖြစ်ပါတယ်။                                                                                                                                                                                  |  |



count = countOccurrence("abcdeabcefabcdzabzy", "bc") ကို trace လိုက်ပါ။

ဒီလောက်ဆိုရင် String ကိုနားလည်သောပေါက်လောက်ပြီထင်ပါတယ်။

ဒီမှာပဲ ရပ်လိုက်ပါတော့မယ်။

## အခန်း (၁၂)

### Array အခြေခံ နှင့် Basic Algorithms

#### Array ဆိုတာ

အရင် တုန်းကတော့ array ဆိုတာ တူညီတဲ့ data type တွေ စုထားတာကို array လိုပေါ်ပါတယ်။ ဆိုလိုတာက integer array ဆို integer တွေချည်း တန်းစီပြီး သိမ်းလိုရတယ်။ array က အခန်း 5 ခန်းယူထားရင် integer 5 လုံး သိမ်းလိုရတယ်ပေါ့။ array က အခန်း 100 ယူထားရင် integer အလုံး 100 သိမ်းလိုရတယ်ပေါ့။ ဒီလိုပါပဲ double array ဆို double တွေချည်း သိမ်းလိုရပါတယ်။ character array ဆိုရင် character တွေချည်း တန်းစီပြီး သိမ်းလိုရပါတယ်။ ဒါကြောင့်လည်း string ဟာ character တွေချည်း တန်းစီပြီး သိမ်းတာဖြစ်လို့ string ကို character array လိုလည်း ပြောကြတာဖြစ်ပါတယ်။

သို့သော် ဒီနေ့ခေတ်မှာ တော့ array ဟာ တူညီတဲ့ data type တွေပဲ သိမ်းလိုရတာ မဟုတ်တော့ပဲ မတူညီတဲ့ data type တွေကိုပါ စုစည်း သိမ်းဆည်းလိုရနေပါပြီ။ ဘယ်လိုပဲ ရနေရနေ array ကိုတော့ ဒီနေ့အချိန်ထိ တူညီတဲ့ data type တွေ သိမ်းဆည်းဖို့ အတွက် အသုံးပြုကြတာ များပါတယ်။

ပြီးတော့ Java လိုမျိုး programming language တွေမှာ array အခန်းနံပါတ် index ဟာ auto သတ်မှတ်ပေးတယ်။ 0 ကစတယ်။ သို့သော် တရာ့ဗျား language တွေမှာတော့ အခန်းနံပါတ်ဟာ ကိုယ်ကြိုက်တာ ထားလိုရတယ်ဆိုတာ ရှုံးမှာ ပြောခဲ့ပြီးပါပြီ။ ဒါကြောင့် array ရဲ့ length - array အခန်း ဘယ်နှစ်ခန်းရှိသလဲဆိုတာ တွေက်ချင်ရင်

$$\text{Length} = \text{UB} - \text{LB} + 1$$

UB ဆိုတာ upper bound, LB ဆိုတာ lower bound ပါ။ ဒီပုံသေနည်းဟာ မထူးဆန်းပါဘူး။  
သချို့ခန်းမှာ ကိန်းတစ်ခု နဲ့ တစ်ခုအကြား ကိန်းသယ်နှစ်လုံးရှိသလဲဆိုတာ တွက်တဲ့ ပုံသေနည်းပါပဲ။

|   |         |
|---|---------|
| 0 | Ko Ko   |
| 1 | Ko Let  |
| 2 | Nyi Nyi |

UB = 2, LB = 0

$$\text{Length} = \text{UB} - \text{LB} + 1 = 2 - 0 + 1 = 3$$

စုစုပေါင်း array ခန်း 3 ခန်းရှိပါတယ်။

|     |         |
|-----|---------|
| 5   | Aye Aye |
| 6   | Bo Bo   |
| ... | ....    |
| 87  | Zaw Zaw |

UB = 87, LB = 5

$$\text{Length} = \text{UB} - \text{LB} + 1 = 87 - 5 + 1 = 83$$

စုစုပေါင်း array ခန်း 83 ခန်းရှိပါတယ်။

အခါ နမူနာ ပြထားတဲ့ array J ခုလုံးဟာ စာသားတွေသိမ်းထားတာဖြစ်လို့ တူညီတဲ့ data type တွေ  
သိမ်းတဲ့ String data type array တွေပဲဖြစ်ပါတယ်။

## Representation of Array in Memory

### Linear Array

Linear array ဆိတာ one-dimensional array ကိုပြောတာပါ။ ကျွန်မတိုကာသာ array ကို ယူသုံးရင် 0,1,2 စတဲ့ index လေးနဲ့ အလွယ်တကူ ယူသုံးတာ ဖြစ်ပေမဲ့ နောက်ကွယ်ကနေ ဘယ်လိုတွက်ချက် လုပ်ကိုင်သွားသလဲဆိတာလေး တစ်ချက် ကြည့်ကြည့် ကြရအောင်။

$$LOC(LA[i]) = Base(LA) + w * (i - lower\ bound)$$

**LOC (LA[i])** – location of array LA[i]

**Base (LA)** – အားကစားပြိုင်ပွဲရှိလို အားကစားအဖွဲ့တစ်ဖွဲ့လာမယ်၊ ဟောခန်းကြီးထဲမှာ စုပြီး နေထိုင်ကြရမယ်။ ကိုယ်က ဟောခန်းထဲ ဝင်လာတဲ့ အချိန်မှာ ကိုယ့်ရှုံးမှာ သူငယ်ချင်းအချို့က နေရာယူပြီးသွားပြီဆိုရင် ကိုယ်က သူတို့ ဘေးကနေဆက်ပြီး နေရာယူရပါမယ်။ အဲဒါကို ကိုယ့်ရဲ့ base address လို့ ပြောလို့ရပါတယ်။ Memory ဆိတာလည်း အဲဒီလိုပါပဲ။ ရှုံးမှာ နေရာယူထားတာတွေရှိရင် ရောက်တဲ့နေရာကနေစပြီး ဆက်နေရာယူရပါတယ်။ array တစ်ခု ဆောက်လိုက်တယ်၊ Memory ပေါ်က လွတ်တဲ့နေရာကနေ စပြီး နေရာယူရတယ်၊ အဲဒါ array ရဲ့ base address ပါပဲ။

**W** - ကတော့ number of bytes ပါ။ data type တစ်ခုမှာ ဘယ်လောက် byte နေရာယူမလဲဆိတာ language က ကြိုတင် သတ်မှတ်ထားတာ ရှိပါတယ်။ Java မှာဆိုရင် char က 1 byte, int က 4 bytes နဲ့ double က 4 bytes စီနေရာယူပါတယ်။

**i** – index အခန်းနံပါတ်ဖြစ်ပါတယ်။

**Lower bound** - ကတော့ အပေါ်မှာ ပြောခဲ့သလို စတဲ့ အခန်းနံပါတ်ဖြစ်ပါတယ်။

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

ဒါကတော့ Java နည်းနဲ့ array ဆောက်ပြထားတာဖြစ်ပါတယ်။ int data type ပါ။ array name or variable name ကို arr လို့ ပေးထားပါတယ်။ one-dimensional array ဖြစ်တဲ့အတွက် arr ရဲ့ နောက်မှာ []-ထောင့်ကွင်း တစ်ခုလိုက်ပါတယ်။ Array မှာ {} - တွန်ကွင်းနဲ့ values တွေတန်းစီထည့်ရပါတယ်။ ရှုံးဆုံး value 10 က 0 ခန်း၊ 20 က 1 အခန်း စသည်ဖြင့် အောက်မှာ ပြထားသိလို့ နေရာယူသွားမှာ ဖြစ်ပါတယ်။

|   |    |      |
|---|----|------|
| 0 | 10 | 1000 |
| 1 | 20 | 1004 |
| 2 | 30 | 1008 |
| 3 | 40 | 1012 |
| 4 | 50 | 1016 |

ပုံမှာ 0,1,2,3,4 က ကျွန်မတို့ ယူသုံးမည့် အခန်းနံပါတ် index, 1000,1004 စတာတွေဟာ တက္ကာ memory address တွေဖြစ်ပါတယ်။ 1000 ဟာ base address ဖြစ်ပါမယ်။ int data type ဖြစ်လို့ တစ်ခန်းကို 4 bytes ယူမှာဖြစ်ပါတယ်။

အခန်းနံပါတ် 3 ရဲ့ memory address ကို တွက်ကြည့်ပါမယ်။

$$\text{LOC } (\text{LA}[i]) = \text{Base}(\text{IA}) + w * ( i - \text{lower bound})$$

$$\text{LOC } (\text{arr}[3]) = \text{Base}(\text{arr}) + 4 * ( 3 - 0 ) = 1000 + 4 * 3 = 1012$$

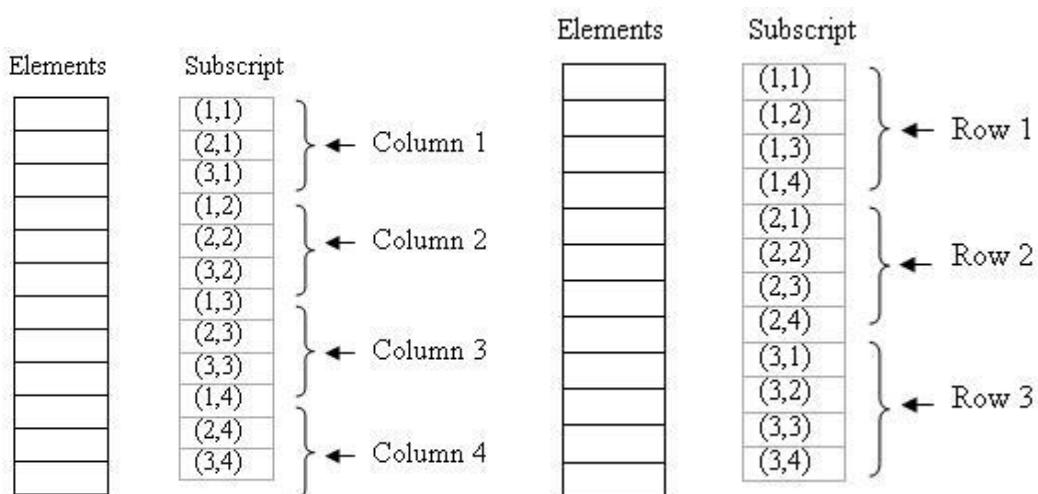
 50 ရဲ့ memory address ကို တွက်ကြည့်ကြည့်ပါ။

## Multi-Dimensional Array

Multi-dimensional array ဆိတာ two-dimensional နဲ့ အထက်ကို ခေါ်တာဖြစ်ပါတယ်။

Multi-dimensional array ထဲကမူ Row နဲ့ Column ဆိတဲ့ Index ၂ခု ရှိတဲ့ two-dimensional array

ရဲ့ နောက်ကွယ်ကနေ memory address ကို ဘယ်လိုတွက်ချက် လုပ်ကိုင်သွားသလဲဆိတာလေး  
တစ်ချက် ကြည့်ကြည့် ကြရအောင်။ အဲဒီမှာ ၂ မျိုး ရှိပါတယ်။ Column ကိုဌီးစားပေးတွက်တဲ့ column-major နဲ့ Row ကိုဌီးစားပေးတွက်တဲ့ row-major တို့ပဲ ဖြစ်ပါတယ်။ Java အပါအဝင် language  
တော်တော်များများကတော့ row ကိုဌီးစားပေးတဲ့ row-major တွေပဲဖြစ်ပါတယ်။



ပထမပုံကတော့ column-major နေရာယူသွားတာပြတဲ့ပဲ ဖြစ်ပြီး၊ ဒုတိယပုံကတော့ row-major နေရာယူသွားတာပြတဲ့ပဲ ဖြစ်ပါတယ်။

### Column-major

$$LOC(A[R,C]) = \text{Base}(A) + w[M * (C-Lc) + (R-Lr)]$$

### Row-major

$$LOC(A[R,C]) = \text{Base}(A) + w[N * (R-Lr) + (C-Lc)]$$

M ဆိုတာ column တစ်ခုမှာရှိတဲ့ row အရေအတွက်ဖြစ်ပါတယ်။ N ဆိုတာကတော့ row တစ်ခုမှာ ရှိတဲ့ column အရေအတွက် ဖြစ်ပါတယ်။ Lc ဆိုတာ column ရဲ့ lower bound, Lr ဆိုတာ row ရဲ့ lower bound ဖြစ်ပါတယ်။

|          | <b>0</b> | <b>1</b> | <b>2</b> | <b>3</b> | <b>4</b> |
|----------|----------|----------|----------|----------|----------|
| <b>0</b> | 10       | 20       | 30       | 40       | 50       |
| <b>1</b> | 60       | 70       | 80       | 90       | 100      |
| <b>2</b> | 110      | 120      | 130      | 140      | 150      |

Row-major နဲ့ ပြောကြည့်ကြရအောင်။ array အမည်ကို A လိုပဲထားပါတော့။ int data type array ဖြစ်တယ်။ int ဖြစ်လို့ တစ်ခန်းကို 4 bytes စီနေရာယူမယ်။ row 3 ခု column 4 ခုဖြစ်လို့ 3\*4 array လို့ ပြောလို့ရပါတယ်။ Base address က 500 ဆိုပါတော့။ ဒါဆိုရင် row-major နဲ့ 140 ဆိုတဲ့ value ရဲ့ memory address ကို တွက်ကြည့်ရအောင်။ 140 ဆိုတော့ A[2][3] ဖြစ်ပါတယ်။

$$\text{LOC}(A[R,C]) = \text{Base}(A) + w [N * (R-Lr) + (C-Lc)]$$

$$\text{LOC}(A[2,3]) = 500 + 4 * (5 * (2-0) + (3-0)) = 500 + 4 * (10+3) = 500 + 4 * 13 = 500 + 52 = 552$$

အဖြေကို စစ်လို့ရအောင် အောက်မှာ memory address ကို ပေါ်လေးလေးနဲ့ ပြပေးထားပါတယ်။

|          | <b>0</b> | <b>1</b> | <b>2</b> | <b>3</b> | <b>4</b> |
|----------|----------|----------|----------|----------|----------|
| <b>0</b> | 500      | 504      | 508      | 512      | 516      |
| <b>1</b> | 520      | 524      | 528      | 532      | 536      |
| <b>2</b> | 540      | 544      | 548      | 552      | 556      |

 အခန်းနံပါတ် 90 ဆိုတဲ့ value ရဲ့ memory address ကို row-major ရော့၊ column-major နဲ့ပါ တွက်ကြည့်ကြည့်ပါ။

## Traversing a Linear Array

Traversing a Linear Array ဆိုတာ array ကို အခန်းပေါက်စွဲ ရောက်အောင် သွားတာဖြစ်ပါတယ်။ ရောက်တဲ့ အချင့်မှာ တော့ output ထုတ်တာဖြစ်ဖြစ်၊ တွက်တာ ဖြစ်ဖြစ် မိမိ နှစ်သက်ရာလုပ်လို့ရပါတယ်။

|   |         |
|---|---------|
| 0 | Ko Ko   |
| 1 | Ko Let  |
| 2 | Nyi Nyi |

|   |     |
|---|-----|
| 3 | CST |
| 4 | CS  |
| 5 | CT  |

နမူနာ ပြထားတဲ့ array 2 ခုကို ကြည့်ကြည့်ပါ။ ပထမ array က အခန်းနံပါတ် 0 ကနေ 2 ထိသွားတယ်။ ဒုတိယ array က အခန်းနံပါတ် 3 ကနေ 5 ထိသွားတယ်။ ဘယ်လိုပဲသွားသွား ခြိုပြီးပြောရရင် lower bound ကနေ upper bound ထိ သွားမယ်ဆိုရင် အခန်းပေါက်စွဲ ရောက်မှာဖြစ်ပါတယ်။

ပထမ array မှာဆို LB(lower bound) က 0, UB(upper bound) က 2, ဒီတော့ LB 0 ကနေစမယ်၊ UB ရောက်တဲ့ အထိသွားမယ်၊ 0 ခန်းပြီးရင် 1 အခန်းဆိုတဲ့အတွက် တစ်ကြိမ်ပြီးတိုင်း အခန်းနံပါတ်ကို 1 တိုးပြီး update လုပ်မယ်။

ပုံစံမ array မှာဆို LB(lower bound) ကဲ 3, UB(upper bound) ကဲ 5, ဒီတော့ LB 3 ကနေစမယ်၊ UB ရောက်တဲ့ အထိသွားမယ်၊ 3 ခန်းပြီးရင် 4 ခန်းဆိုတဲ့အတွက် တစ်ကြိမ်ပြီးတိုင်း အခန်းနံပါတ်ကို 1 တိုးပြီး update လုပ်မယ်။

### ***Algorithm: Traversing a linear array***

1. Set index = LB
2. Repeat while index <= UB:
3.       Do something with LA[index]
4.       Index = index + 1
5. End of Loop

ဒါက while loop နဲ့ ရေးတာဖြစ်ပြီး for loop နဲ့ ထပ်ပြီး ရေးပါမယ်။

### ***Algorithm: Traversing a linear array 2***

1. Repeat for index = LB to UB by 1:
2.       Do something with LA[index]
3. End of Loop

ဒီနေရာထိ ဖတ်လာပြီဆိုတော့ ဒီ traversing algorithm လောက်ကတော့ ကလေးကစားသလောက် ဖြစ်နေလောက်ပါပြီ။ ဒီတော့ trace လိုက်မဖြတော့ပါဘူး။



ဒါလေးတွေ လျှကျင့်ကြည့်ကြရအောင်။

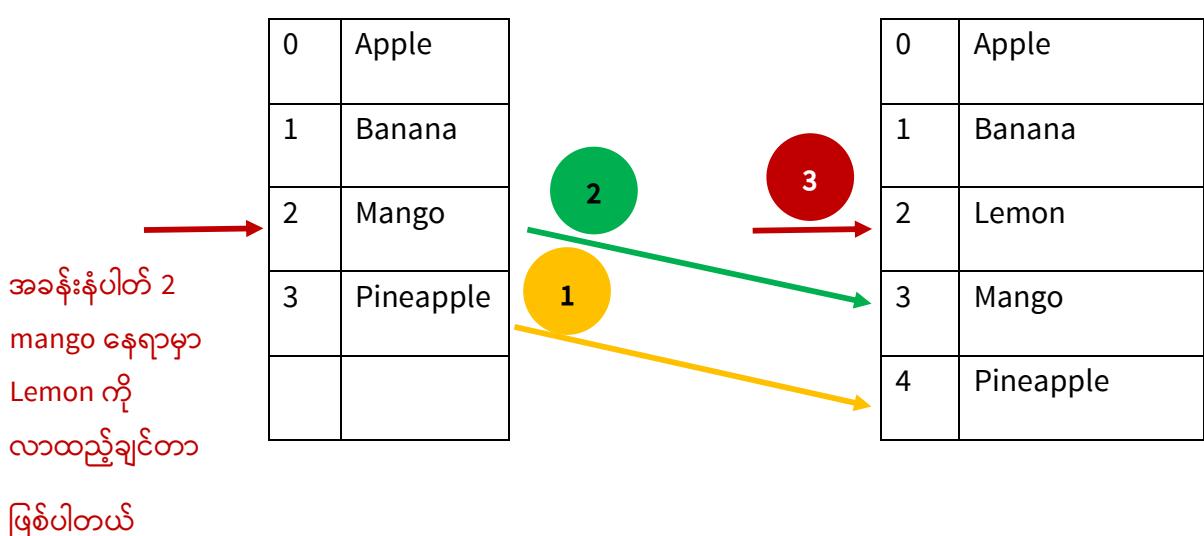
မောင်မောင်ဟာ 2000 ခုနှစ်ကနေပြီး 2021 ခုနှစ်အထိ သူရဲ့ ဝင်ငွေတွေကို one-dimensional array(linear array) နဲ့ သိမ်းထားတယ်။ အခန်းနံပါတ်တွေကို ခုနှစ်တွေပေးထားတာ ဖြစ်လို

အခန်းနံပါတ်က 2000 ကစြမ်း 2021 အထိ သွားပါတယ်။ အထဲမှာတော့ value အနေနဲ့ သူ့ရဲ့ ဝင်ငွေစာရင်းကို သိမ်းဆည်းထားပါတယ်။

1. Array ရဲ့ LB, UB နှင့် length ကိုရှာပါ။
2. အပေါ်က traversing algorithm ကို အသုံးပြုပြီး 2000 ကနေ 2021 အထိ၊ ခုနှစ် နှင့် ဝင်ငွေကို “2000:1000000” ဆိုတဲ့ ခုနှစ်နဲ့ ဝင်ငွေကြား ‘:’ လေးခြားထားတဲ့ ပုံစံနဲ့ output ထုတ်ပြပေးတဲ့ algorithm ကိုရေးပါ။
3. အပေါ်က traversing algorithm နှင့် conditional statement ကို အသုံးပြုပြီး 100 သိန်းကျော် ဝင်ငွေရှိတဲ့ နှစ်တွေကို output ထုတ်ပြပေးတဲ့ algorithm ကိုရေးပါ။
4. အပေါ်က traversing algorithm နှင့် conditional statement ကို အသုံးပြုပြီး 100 သိန်းကျော် ဝင်ငွေရှိတဲ့ နှစ်အရေအတွက်ကို count လုပ်ပေးတဲ့ algorithm ကိုရေးပါ။

### Inserting into Linear Array

Linear Array ရဲ့ ကိုယ်ကြိုက်တဲ့ နေရာမှာ ကိုယ်ထည့်ချင်တဲ့ data လေး သွားထည့်တာကို စဉ်းစားကြည့်ကြရအောင်။



စစချင်း 0 to 3 ဆိုပြီး array ခန်း 4 ခန်းရှိပါတယ်။ အဲဒီထက် 2 အခန်း Mango ရဲ့ နေရာကို Lemon လာပြီး ထည့်ချင်တာ ဖြစ်ပါတယ်။ ဒီတော့ ပေးဝင်လိုဂေါ်အောင် ဝင်ချင်တဲ့အခန်းနဲ့ သူရဲ့ နောက်အခန်းတွေဟာ အောက်ကို တစ်ခန်းစီ ဆင်းပေးရမှာဖြစ်ပါတယ်။ အဲဒီမှာ Mango ကို အရင် ပေးဆင်းခဲ့ရင် pineapple နေရာမှာ mango ဝင်သွားတော့ overwrite ဖြစ်ပြီး pineapple ပျောက်သွားမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် ရွှေ့မယ် ဆိုရင် နောက်ဆုံးအခန်းကနေ စရွှေ့ရပါတယ်။

ယုံမှာပြထားတဲ့အတိုင်းဆို Pineapple အရင်ရွှေ့၊ ပြီးမှ pineapple နေရာကို Mango ရွှေ့ရမှာဖြစ်ပါတယ်။ အခန်းနံပါတ် 2 (Mango) ကိုရွှေ့တာဟာ ထည့်ချင်တဲ့ အခန်းဖြစ်တဲ့အတွက်၊ ထည့်ချင်တဲ့ ခန်းလွတ်သွားပြီဖြစ်လို့ အဲဒီ အခန်းမှာ ထည့်ချင်တဲ့ lemon ကို ထည့်လိုက်ပါတယ်။

ဆိုလိုတာကတော့

1. နောက်ဆုံးအခန်းကနေပြီး ထည့်ချင်တဲ့ အခန်းရောက်သည်အထိ အောက်ကို တစ်ခန်းချင်းစီ ဆင်းပေးရပါမယ်။
2. ပြီးမှ ထည့်ချင်တဲ့ အခန်းထဲကို ထည့်ချင်တဲ့ တန်ဖိုး ထည့်ပေးလိုက်ရမှာ ဖြစ်ပါတယ်။

### **Algorithm: Inserting into linear array**

1. inserAt(LA, UB, II, IITEM)
2. /\*LA – linear array, UB – upper bound, II – index to insert, IITME – Item to insert \*/
3. Set index = UB
4. Repeat while index >= II:
5.     LA[index+1] = LA [index]
6.     Index = index - 1
7. End of Loop
8. LA [II] = IITEM
9. UB = UB + 1

Insert လုပ်ရန် သိမြှိုလိုတာက ထည့်ချင်တဲ့ array ရယ်၊ ထည့်ချင်တဲ့ အခန်းနံပါတ်၊ ထည့်ချင်တဲ့ တန်ဖိုး၊ နောက်ဆုံးအခန်းနံပါတ် တို့ပဲဖြစ်ပါတယ်။

ဒါက while loop နဲ့ ရေးတာဖြစ်ပြီး for loop နဲ့ကတော့ ဖတ်နေသူ အရေးကျင့်ရမှာ ဖြစ်ပါတယ်။

| Anna | Brown | David | Emely | Ford |
|------|-------|-------|-------|------|
| 0    | 1     | 2     | 3     | 4    |

Array LA ကို ပုံမှန်ဖြနေကျ ထောင်လိုက်ကနေ အလျားလိုက် ပြလိုက်လို့ မျက်စိလည် မသွားပါနဲ့အေး။ ဘယ်လို ပြပြ အတူတူပါပဲ။ 2 အခန်းနေရာမှာ Charles ကို ထည့်ချင်တယ် ဆိုပါတော့ trace လိုက်ကြည့်ရအောင်။

Array name က LA ပဲ ဆိုပါတော့၊ နောက်ဆုံးအခန်းနံပါတ် UB = 4, ထည့်ချင်တဲ့ အခန်း II = 2, ထည့်ချင်တဲ့ IITEM = Charles။

| Line | Index | အလုပ်လုပ်ပုံ                                                                                                    | LA                                       |
|------|-------|-----------------------------------------------------------------------------------------------------------------|------------------------------------------|
| 1    |       | UB = 4, II = 2, IITEM = Charles                                                                                 | Anna, Brown, David,<br>Emely, Ford       |
| 2    |       | Comment ဆိုတော့ အလုပ်မလုပ်ဘူး။                                                                                  |                                          |
| 3    | 4     | index = UB<br><br>နောက်ဆုံးအခန်းကနေပြီး စဉ်မှာမို့ index ထဲကို<br>UB ထည့်                                       |                                          |
| 4    |       | index >= II ဆိုတော့ 4 >= 2 ဆိုတော့ while<br><br>condition မှန်တယ်၊ ဒီတော့ while ရဲ့ body ကို<br>အလုပ်လုပ်ပါမယ်။ |                                          |
| 5    |       | LA[index+1] = LA [index]<br><br>LA[5] = LA [4]                                                                  | Anna, Brown, David,<br>Emely, Ford, Ford |

|   |   |                                                                                                                                                                          |                                               |
|---|---|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------|
|   |   | LA[5]=Ford                                                                                                                                                               |                                               |
| 6 | 3 | <p>index = Index -1</p> <p>နောက်ဆုံးအခန်း ရွှေ့ပြီး အပေါ်က အခန်း</p> <p>ဆက်ရွှေ့မှာ ဖြစ်လို့ index ကို 1 လျော့ပြီး</p> <p>condition စစ်ဖို့ update လုပ်တာ ဖြစ်ပါတယ်။</p> |                                               |
| 7 |   | <p>End of loop ဆိုတော့ while condition ဖြစ်တဲ့</p> <p>line 4 ကိုသွားပါမယ်။</p>                                                                                           |                                               |
| 4 |   | <p>index &gt;= 1 ဆိုတော့ 3 &gt;= 2 ဆိုတော့ while</p> <p>condition မှန်တယ်၊ ဒီတော့ while ရဲ့ body ကို</p> <p>အလုပ်လုပ်ပါမယ်။</p>                                          |                                               |
| 5 |   | <p>LA[index+1] = LA [index]</p> <p>LA[4] = LA [3]</p> <p>LA[4]=Emely</p>                                                                                                 | Anna, Brown, David,<br><br>Emely, Emely, Ford |
| 6 | 2 | <p>index = index -1</p> <p>ထည့်ချင်တဲ့ အခန်းရောက်သည်အထိ</p> <p>ဆက်ပြီးရွှေ့မှာ ဖြစ်လို့ index ကို 1 လျော့ပြီး</p> <p>condition စစ်ဖို့ update လုပ်တာ ဖြစ်ပါတယ်။</p>      |                                               |
| 7 |   | <p>End of loop ဆိုတော့ while condition ဖြစ်တဲ့</p> <p>line 4 ကိုသွားပါမယ်။</p>                                                                                           |                                               |
| 4 |   | <p>index &gt;= 1 ဆိုတော့ 2 &gt;= 2 ဆိုတော့ while</p> <p>condition မှန်တယ်၊ ဒီတော့ while ရဲ့ body ကို</p> <p>အလုပ်လုပ်ပါမယ်။</p>                                          |                                               |

|   |   |                                                                                                                                                                                        |                                                 |
|---|---|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------|
| 5 |   | LA[index+1] = LA [index]<br><br>LA[3] = LA [2]<br><br>LA[3]=David                                                                                                                      | Anna, Brown, David, David,<br><br>Emely, Ford   |
| 6 | 1 | index = Index -1<br><br>ထည့်ချင်တဲ့                          အခန်းရောက်သည်အထိ<br><br>ဆက်ပြီးရွှေ့မှာ ဖြစ်လို့ index ကို 1 လျှော့ပြီး<br><br>condition စစ်ဖို့ update လုပ်တာ ဖြစ်ပါတယ်။ |                                                 |
| 7 |   | End of loop ဆိုတော့ while condition ဖြစ်တဲ့<br><br>line 4 ကိုသွားပါမယ်။                                                                                                                |                                                 |
| 4 |   | index >= 1 ဆိုတော့ 1 >= 2 ဆိုတော့ while<br><br>condition မှားသွေးပြီ ဖြစ်လို့ end of loop ရဲ့ အပြင်<br><br>line 8 ကို သွားမှာ ဖြစ်ပါတယ်။                                               |                                                 |
| 8 |   | LA [II] = IITEM<br><br>LA[2]=Charles<br><br>ဘေးက LA အခန်းကို ကြည့်လိုက်ပါ၊<br><br>ထည့်ချင်တဲ့အတိုင်း ထည့်ပြီးသွားပြီ ဖြစ်ပါတယ်။                                                        | Anna, Brown, Charles,<br><br>David, Emely, Ford |

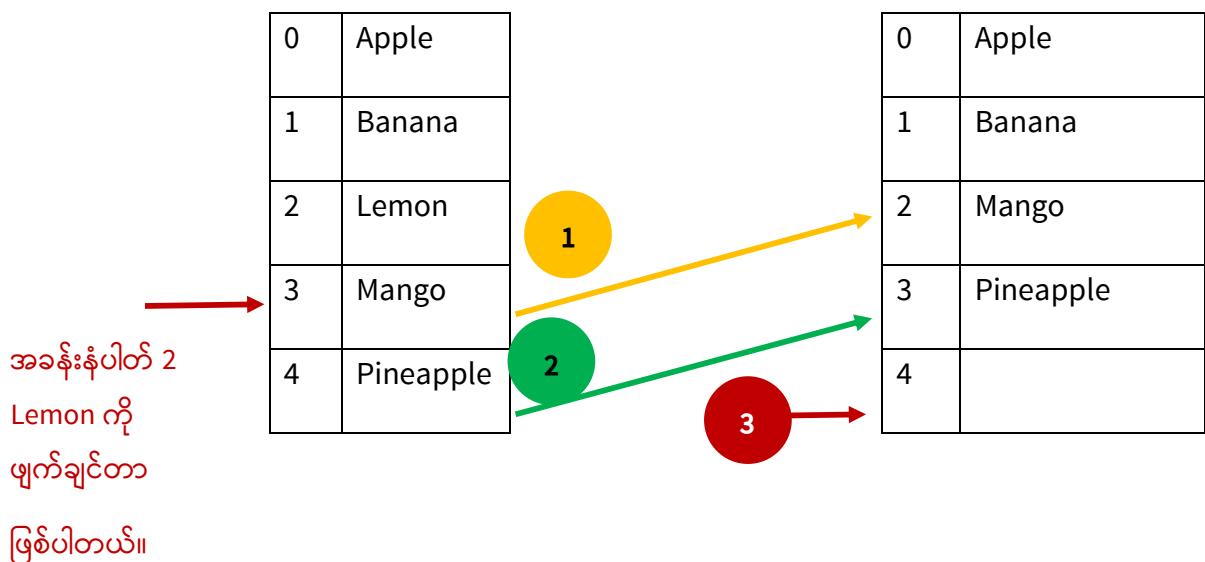


Trace လိုက်ကြည့်ပါ။

LA => {AA, BB, DD, EE, FF, GG} ထဲကို CC ကို ထည့်ချင်ပါတယ်။ ငယ်စဉ်ကြီး လိုက် စီပြီးသား ဖြစ်ချင်တာဖြစ်လို့ ဘယ်အခန်းမှာ CC ကို ထည့်သင့်လဲ ဆုံးဖြတ်ပြီး trace လိုက်ပါ။

## Deleting from a Linear Array

Linear Array ရဲ့ ကိုယ်ကြိုက်တဲ့ နေရာက value ကိုဖျက်တာလေး စဉ်းစားကြည့်ကရအောင်။



စစ်ဆေး 0 to 4 ဆိုပြီး array ခန်း 4 ခန်းရှုပါတယ်။ အဲဒီထဲက 2 အခန်း Lemon ကို ဖျက်ချင်တာ ဖြစ်လို Lemon ဆိုတဲ့ 2 အခန်းနေရာမှာ 3 အခန်းက တန်ဖိုး Mango က အပေါ်တက်ပြီး ဝင်လာရမှာဖြစ်ပါတယ်။

ဒါတော့ 2 အခန်းဖျက်ချင်ရင် 2 နေရာကို (ဖျက်ချင်တဲ့ အခန်းနံပါတ် + 1) 3 ကအစားဝင်လာရမှာ ဖြစ်ပါတယ်။ ဖျက်ချင်တဲ့ အခန်းနံပါတ် + 1 နေရာကို ဖျက်ချင်တဲ့ အခန်းနံပါတ် + 2 က အစားဝင်လာရမှာ ဖြစ်ပါတယ်။ နောက်ဆုံးအခန်းရောက်သည်အထိ အပေါ်တက်ရမှာ ဖြစ်ပါတယ်။ ပုံလေးမှာ နံပါတ်လေးတွေတပ်ပြထားပါတယ်။ ကြည့်ကြည့်ပါ။

ပြောရမယ်ဆိုရင် insert က ထည့်တာဖြစ်လို နေရာလွှတ်အောင် အောက်ဆင်းပေးရတာဖြစ်ပြီး၊ delete ကတော့ ဖျက်တာဖြစ်လို ဖျက်ချင်တဲ့ အခန်းနံပါတ် + ၁ ကနေပြီး နောက်ဆုံးအခန်း(UB) ရောက်သည်အထိ အပေါ်တက်ရတာ ဖြစ်ပါတယ်။ Insert ကတော့ အခန်းတစ်ခန်းတိုးလာတာ (UB+1) ဖြစ်ပြီး၊ delete ကတော့ အခန်းတစ်ခန်းလျော့သွားတာ ဖြစ်လို အလုပ်လုပ်တာပြီးသွားရင် UB ကို ၁ နှတ်ပေးရမှာ ဖြစ်ပါတယ်။

သိမ့် လိုတာတွေက array – LA, ဖျက်ချင်တဲ့ အခန်းနံပါတ် - DI, နောက်ဆုံးအခန်းနံပါတ် UB ရယ် ဖြစ်ပါတယ်။

### ***Algorithm: Deleting Form a Linear Array***

1. deleteAt(LA, UB, DI)
2. /\*LA – linear array, UB – Upper Bound, DI – index to delete \*/
3. Set index = DI
4. Repeat while index < UB:
5.     LA[index] = LA [index+1]
6.     Index = index + 1
7. End of Loop
8. UB = UB - 1

ဒါက while loop နဲ့ ရေးတာဖြစ်ပြီး for loop နဲ့ကတော့ ဖတ်နေသူ အရေးကျင့်ရမှာ ဖြစ်ပါတယ်။

| Anna | Brown | Charles | David | Emely |
|------|-------|---------|-------|-------|
| 0    | 1     | 2       | 3     | 4     |

Charles ကို ဖျက်ချင်တယ် ဆိုပါတော့ trace လိုက်ကြည့်ရအောင်။

Array name က LA ပဲ ဆိုပါတော့၊ နောက်ဆုံးအခန်းနံပါတ် UB = 4, ဖျက်ချင်တဲ့ အခန်း DI = 2

| Line | Index | အလုပ်လုပ်ပုံ                   | LA                                 |
|------|-------|--------------------------------|------------------------------------|
| 1    |       | UB = 4, DI = 2                 | Anna, Brown, Charles, David, Emely |
| 2    |       | Comment ဆိုတော့ အလုပ်မလုပ်ဘူး။ |                                    |
| 3    | 2     | index = DI = 2                 |                                    |
| 4    |       | index < UB                     |                                    |

|   |   |                                                                                                 |                                  |
|---|---|-------------------------------------------------------------------------------------------------|----------------------------------|
|   |   | 2 < 4 ဆိုတော့ while condition မှန်လို့<br>while ရဲ့ body ကို အလုပ်လုပ်မယ်။                      |                                  |
| 5 |   | LA[index] = LA [index+1]<br><br>LA[2]= LA[3]<br><br>LA[2]=David                                 | Anna, Brown, David, David, Emely |
| 6 | 3 | index = index + 1<br><br>While loop ရဲ့ update ဖြစ်ပါတယ်။                                       |                                  |
| 7 |   | End of loop ဆိုတော့ while condition<br><br>line 4 ကို ပြန်တက်ပါမယ်။                             |                                  |
| 4 |   | index< UB<br><br>3 < 4 ဆိုတော့ while condition မှန်လို့<br><br>while ရဲ့ body ကို အလုပ်လုပ်မယ်။ |                                  |
| 5 |   | LA[index] = LA [index+1]<br><br>LA[3]= LA[4]<br><br>LA[3] = Emely                               | Anna, Brown, David, Emely, Emely |
| 6 | 4 | index = index + 1<br><br>While loop ရဲ့ update ဖြစ်ပါတယ်။                                       |                                  |
| 7 |   | End of loop ဆိုတော့ while condition<br><br>line 4 ကို ပြန်တက်ပါမယ်။                             |                                  |
| 4 |   | index< UB<br><br>4 < 4 ဆိုတော့ while condition<br><br>မှာသွားပြီဖြစ်လို့ end of loop ပြီး       |                                  |

|   |  |                                                                                        |                           |
|---|--|----------------------------------------------------------------------------------------|---------------------------|
|   |  | နောက်တစ်ကြောင်း line 8 ကို<br>သွားပါမယ်။                                               |                           |
| 8 |  | UB = UB - 1 ဆိုတာ တစ်ခန်းလျှော့<br>လိုက်တာဖြစ်လို့ LA အဖြေကို LA<br>column မှာကြည့်ပါ။ | Anna, Brown, David, Emely |

ဒီနေရာမှာ နည်းနည်းလေးထပ်ရှင်းချင် ပါတယ်။ ပထမဆုံး လုပ်ရမည့်အလုပ်က DI + 1 အခန်းကို DI ထဲ  
ချွေရမှာဖြစ်ပါတယ်။ အဲဒီမှာ Set index = DI လို့ ရေးလိုက်တော့ Index ဟာ အလုပ်လုပ်ရမည့် အခန်း  
2 ခုအနက် ရှေ့အခန်းကို ယူလိုက်တဲ့ သဘောဖြစ်ပါတယ်။ ဒီတော့ သူ့နောက်အခန်းဖြစ်တဲ့ Index + 1  
နဲ့ အလုပ်လုပ်ရတော့မှာ ဖြစ်ပါတယ်။ Index ဟာ နောက်ဆုံးအခန်းဖြစ်တဲ့ UB ထိသွားမယ်ဆိုရင် Index  
+ 1 ဟာ သွားစရာ နေရာမရှိတော့ပါဘူး။ ဒါကြောင့် Index ဟာ တစ်ခန်းလျှော့ပြီး <UB (UB  
ထက်ငယ်တဲ့အထိ) ဆိုပြီး လုပ်ထားတာ ဖြစ်ပါတယ်။ ဒါမှ Index + 1 ဟာ နောက်ဆုံးအခန်းဖြစ်မှာ  
ဖြစ်ပါတယ်။

Repeat while **index < UB**:

LA[index] = **LA [index+1]**

Index = index + 1

End of Loop

တကယ်လို့ Index ဟာ အလုပ်လုပ်ရမည့် အခန်း ၂ ခုအနက် နောက်အခန်းကိုယူခဲ့မယ် ဆိုရင်  
သူဟာ သူရဲ့ အရှေ့တစ်ခန်းထဲကို ထည့်ပေးရမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် ဖျက်မည့်အခန်းရဲ့  
နောက်တစ်ခန်းကို ဖျက်မည့်အခန်းထဲ ထည့်ချင်တာဖြစ်လို့

Set Index = DI + 1

ဖြစ်ပါမယ်။ ပြီးတော့ Index ဟာ သူရှုံးအခန်းနဲ့ အလုပ်လုပ်မှာ ဖြစ်တဲ့အတွက် သူဟာ နောက်ဆုံးအခန်းထိ သွားလိုက်မှာ ဖြစ်ပါတယ်။ ဒါကြောင့်

Repeat while index <= UB:

LA[index - 1] = LA [index]

Index = index + 1

End of Loop

ဆိုပြီး ဖြစ်သွားမှာ ဖြစ်ပါတယ်။ ပြောချင်တာက ယခုရေးလိုက်တာနဲ့ algorithm ထဲမှာ ရေးထားတာနဲ့ အတူတူပဲ ဖြစ်ပါတယ်။ Index ကို အလုပ်လုပ်မည့် အခန်း ၂ ခုထဲက ရှုံးအခန်းယူတာနဲ့ နောက်အခန်းယူတာလေး ကွဲသွားတာ ဖြစ်ပါတယ်။ ဒါကြောင့် problem တစ်ခုမှာ ဖြေရှင်းနည်း မျိုးစုံရှိတယ်ပြောတာဖြစ်ပါတယ်။



အောက်ပါတွေကို လေ့ကျင့်ကြည့်ပါ။

1. Index ကို အလုပ်လုပ်မည့်အခန်းထဲက နောက်အခန်းကို ယူတဲ့ နည်းနဲ့ algorithm ကို အစအဆုံး ပြန်ရေးပေးပါ။
2. နံပါတ် ၁ မှာ ပြန်ရေးထားတဲ့ algorithm ဖြင့်  $LA \Rightarrow \{AA, BB, CC, DD, EE, FF, GG\}$  ထဲမှ CC ဖျက်တာကို trace လိုက်ပါ။

## Linear Search Algorithm

Linear Search Algorithm ဆိုတာ array ထဲမှ ကိုယ်ရှာချင်တဲ့ တန်ဖိုးကို 0 ခန်း၊ 1 အခန်း၊ 2 အခန်း စသည်ဖြင့် တစ်ခန်းပြီးတစ်ခန်း အစဉ်လိုက် လိုက်ပြီးရှာတာဖြစ်ပါတယ်။ ကိုယ်ရှာချင်တဲ့ တန်ဖိုးနှင့် ရောက်နေတဲ့ အခန်းထဲကတန်ဖိုး ညီခဲ့တယ်ဆိုရင် တွေ့တာဖြစ်လို့ အဲဒါ

တွေ့တဲ့အခန်းနံပါတ်ကို return ပြန်ပေးမှာ ဖြစ်ပါတယ်။ နောက်ဆုံးအခန်းထိ ရှာမတွေ့ခဲ့ဘူး ဆိုရင်တော့ မရှိတဲ့ အခန်းနံပါတ်ဖြစ်တဲ့ -1 ကို return ပြန်မှာ ဖြစ်ပါတယ်။ အခန်းနံပါတ်ဟာ 0 က စတာကြောင့် အနှုတ်အခန်းနံပါတ်မရှိတာဖြစ်ပါတယ်။

### **Algorithm: Linear Search**

1. linearSearch(LA, N, x)
2. /\*LA – linear array, N – number of elements in array, x – value to search \*/
3. Set index = 0
4. Set found = -1
5. Repeat while index < N:
6.     If LA[index] == x then:
7.         Found = index
8.         Index = index + 1
9.     End of Loop
10.  Return found

ဒါ algorithm ဟာ အရင် algorithm တွေနဲ့ မတူတာက UB ကို လက်မခံပဲ number of elements လို့ခေါ်တဲ့ array အခန်းအရေအတွက်ကို လက်ခံထားတာဖြစ်ပါတယ်။ UB(uppe bound) ဆိုတာ နောက်ဆုံးအခန်းနံပါတ်ဖြစ်ပါတယ်။ Number of elements က 10 ဆိုရင် array ခန်း 10 ခန်းရှိတယ်လို ဆိုလိုတာဖြစ်လို အခန်းနံပါတ်တွေကတော့ 0 to 9 ဖြစ်မယ်။ ဒီတော့ Number of elements(N) က 10 ဆိုရင် UB - နောက်ဆုံးအခန်းနံပါတ်ဟာ 9 ဖြစ်မှာ ဖြစ်ပါတယ်။ ဒါကြောင့် index < N ဆိုပြီ နောက်ဆုံးအခန်းနံပါတ်ထိ ပိတ်ထားတာဖြစ်ပါတယ်။

UB လက်ခံလည်းရပါတယ်။ ဘာကြောင့် UB လက်မခံပဲ N လက်ခံသလဲ မစဉ်းစားနေပါနဲ့။

**Algorithm ရေးသူ စိတ်ကြိုက်လုပ်လိုရပါတယ်။**

Trace လိုက်ပြပါမယ်။

LA => {10, 20, 30, 40, 50, 60}

X = 10, N = 6

| Line    | Index |    | အလုပ်လုပ်ပုံ                                                                                                                                                                                                                                                                                                                                               |
|---------|-------|----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1,2,3,4 | 0     | -1 | <p>Line 1 =&gt; LA =&gt; {10, 20, 30, 40, 50, 60}, X = 10, N = 6</p> <p>Line 2 =&gt; comment ဆိုတော့ အလုပ်မလုပ်</p> <p>Line 3 and 4 =&gt; index =0, found = -1</p>                                                                                                                                                                                         |
| 5       |       |    | <p>index &lt; N ဆိုတော့ 0 &lt; 6 ဆိုတော့မှန်တယ်။ while condition မှန်တော့</p> <p>while body ကို အလုပ်လုပ်ပါမယ်။</p>                                                                                                                                                                                                                                        |
| 6,7     |       | 0  | <p>If နဲ့ တွေ့သလားဆိုပြီး LA[index] == x ဆိုပြီး စစ်ပါတယ်။ LA[0]== 10, 10 == 10 ဆိုတော့ မှန်တယ်။ မှန်တော့ if အောက်က စာကြောင်းကို အလုပ်လုပ်မယ်။ found ထဲကို index ထည့်တယ်။ found = 0 ဖြစ်မယ်။ဆိုလိုတာက 0 ခန်းမှာ အခန်းထဲက တန်ဖိုးနဲ့ ရှာချင်တဲ့ တန်ဖိုး x ညီသွားတာ ဖြစ်လို့ တွေ့တယ်။ တွေ့လို့ တွေ့တဲ့ အခန်းနံပါတ် 0 ကို found ထဲ သိမ်းလိုက်တာဖြစ်ပါတယ်။</p> |
| 8       | 1     |    | <p>index = index + 1</p> <p>index = 0 + 1 = 1</p> <p>0 ခန်းပြီး 1 အခန်းရှာမှာ ဖြစ်လို့ အခန်းနံပါတ်ကို 1 တိုးတာ update လုပ်တာဖြစ်ပါတယ်။</p>                                                                                                                                                                                                                 |
| 9       |       |    | End of loop ဆိုတော့ condition ဆီ ပြန်တက်မယ်။                                                                                                                                                                                                                                                                                                               |
| 5       |       |    | <p>index &lt; N ဆိုတော့ 1 &lt; 6 ဆိုတော့မှန်တယ်။ while condition မှန်တော့</p> <p>while body ကို အလုပ်လုပ်ပါမယ်။</p>                                                                                                                                                                                                                                        |

|     |   |                                                                                                                                                      |
|-----|---|------------------------------------------------------------------------------------------------------------------------------------------------------|
| 6,7 |   | If နဲ့ တွေ့သလားဆိုပြီး LA[index] == x ဆိုပြီး စစ်ပါတယ်။ LA[1]== 10, 20 == 10 ဆိုတော့ မှားတယ်။ မှားတော့ if အောက်က line 7 ကို မလုပ်တော့ပဲ ကျဉ်သွားမယ်။ |
| 8   | 2 | $\text{index} = \text{index} + 1$<br>$\text{index} = 1 + 1 = 2$<br>1 ခန်းပြီး 2 အခန်းရှာမှာ ဖြစ်လို့ အခန်းနံပါတ်ကို 1 တိုးတာ update လုပ်တာဖြစ်ပါတယ်။ |
| 9   |   | End of loop ဆိုတော့ condition ဆီ ပြန်တက်မယ်။                                                                                                         |
| 5   |   | $\text{index} < N$ ဆိုတော့ 2 < 6 ဆိုတော့မှန်တယ်။ while condition မှန်တော့ while body ကို အလုပ်လုပ်ပါမယ်။                                             |
| 6,7 |   | If နဲ့ တွေ့သလားဆိုပြီး LA[index] == x ဆိုပြီး စစ်ပါတယ်။ LA[2]== 10, 30 == 10 ဆိုတော့ မှားတယ်။ မှားတော့ if အောက်က line 7 ကို မလုပ်တော့ပဲ ကျဉ်သွားမယ်။ |
| 8   | 3 | $\text{index} = \text{index} + 1$<br>$\text{index} = 2 + 1 = 3$<br>2 ခန်းပြီး 3 အခန်းရှာမှာ ဖြစ်လို့ အခန်းနံပါတ်ကို 1 တိုးတာ update လုပ်တာဖြစ်ပါတယ်။ |
| 9   |   | End of loop ဆိုတော့ condition ဆီ ပြန်တက်မယ်။                                                                                                         |
| 5   |   | $\text{index} < N$ ဆိုတော့ 3 < 6 ဆိုတော့မှန်တယ်။ while condition မှန်တော့ while body ကို အလုပ်လုပ်ပါမယ်။                                             |

|     |   |                                                                                                                                                             |
|-----|---|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 6,7 |   | If နဲ့ တွေ့သလားဆိုပြီး LA[index] == x ဆိုပြီး စစ်ပါတယ်။ LA[3]== 10,<br>40 == 10 ဆိုတော့ မှားတယ်။ မှားတော့ if အောက်က line 7 ကို<br>မလုပ်တော့ပဲ ကျော်သွားမယ်။ |
| 8   | 4 | index = index + 1<br><br>index = 3 + 1 = 4<br><br>3 ခန်းပြီး 4 အခန်းရှာမှာ ဖြစ်လို့ အခန်းနံပါတ်ကို 1 တိုးတာ update<br>လုပ်တာဖြစ်ပါတယ်။                      |
| 9   |   | End of loop ဆိုတော့ condition ဆီ ပြန်တက်မယ်။                                                                                                                |
| 5   |   | index < N ဆိုတော့ 4 < 6 ဆိုတော့မှန်တယ်။ while condition မှန်တော့<br>while body ကို အလုပ်လုပ်ပါမယ်။                                                          |
| 6,7 |   | If နဲ့ တွေ့သလားဆိုပြီး LA[index] == x ဆိုပြီး စစ်ပါတယ်။ LA[4]== 10,<br>50 == 10 ဆိုတော့ မှားတယ်။ မှားတော့ if အောက်က line 7 ကို<br>မလုပ်တော့ပဲ ကျော်သွားမယ်။ |
| 8   | 4 | index = index + 1<br><br>index = 4 + 1 = 5<br><br>4 ခန်းပြီး 5 အခန်းရှာမှာ ဖြစ်လို့ အခန်းနံပါတ်ကို 1 တိုးတာ update<br>လုပ်တာဖြစ်ပါတယ်။                      |
| 9   |   | End of loop ဆိုတော့ condition ဆီ ပြန်တက်မယ်။                                                                                                                |
| 5   |   | index < N ဆိုတော့ 5 < 6 ဆိုတော့မှန်တယ်။ while condition မှန်တော့<br>while body ကို အလုပ်လုပ်ပါမယ်။                                                          |

|     |   |                                                                                                                                                                                                            |
|-----|---|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 6,7 |   | If နဲ့ တွေ့သလားဆိုပြီး LA[index] == x ဆိုပြီး စစ်ပါတယ်။ LA[5]== 10,<br>60 == 10 ဆိုတော့ မှားတယ်။ မှားတော့ if အောက်က line 7 ကို<br>မလုပ်တော့ပဲ ကျော်သွားမယ်။                                                |
| 8   | 6 | index = index + 1<br><br>index = 5 + 1 = 6<br><br>5 ခန်းပြီး 6 အခန်းရှာမှာ ဖြစ်လို့ အခန်းနံပါတ်ကို 1 တိုးတာ update<br>လုပ်တာဖြစ်ပါတယ်။                                                                     |
| 9   |   | End of loop ဆိုတော့ condition ဆီ ပြန်တက်မယ်။                                                                                                                                                               |
| 5   |   | index < N ဆိုတော့ 6 < 6 ဆိုတော့ condition မှားသွားပြီဖြစ်လို့ looping<br>ထဲက ထွက်မှာဖြစ်လို့ end of loop ရဲနောက် တစ်ကြောင်း line 10 ကို<br>သွားအလုပ်လုပ်မှာ ဖြစ်ပါတယ်။                                     |
| 10  |   | Return found ဆိုတဲ့အတွက် return 0 ဆိုပြီး တွေ့တဲ့ အခန်းနံပါတ်ကို<br>0 ကို return ပြန်ပေးသွားမှာ ဖြစ်ပါတယ်။ တကယ်လို့ နောက်ဆုံး<br>အခန်းထိ မတွေ့ခဲ့ဘူး ဆိုရင်တော့ -1 ကို return ပြန်ပေးသွားမှာ<br>ဖြစ်ပါတယ်။ |

ဒီ algorithm ရဲ့ မကောင်းတဲ့ အချက်ကတော့

LA => {10, 20, 30, 40, 50, 60} ထဲက X = 10 ကိုရှာတာ ပထမဆုံးအခန်း (0 အခန်း) မှာ တင် တွေ့ရဲ့ နဲ့  
နောက်ဆုံးအခန်းအထိ အချိန်ကုန်ခံပြီးလိုက် စစ်နေတာဖြစ်ပါတယ်။ တကယ်က တွေ့ရင် ဆက်ပြီး  
စစ်စရာ မလိုအပ်တော့ပါဘူး။ ဒီတော့ အချိန်ကုန်သက်သာရအောင် တွေ့တာနဲ့ ဆက်မစစ်ပဲ တန်းပြီး

ထွက်ချင်တော့ တွေ့ရင် return ပြန်မယ်လို့ ရေးလိုက်ရင်ရပါပြီ။ ဒါတော့ မှတ်စေချင်တာက အခါန်ကုန်သက်သာစေဖို့ အဖြေထွက်ပြီဆိုတာနဲ့ တန်း ထွက်ပါ တန်ည်းပြောရရင် တန်းပြီး return ပြန်ပါ။ ပြန်ရေးပြရရင်

### **Algorithm: Linear Search (2)**

1. linearSearch(LA, N, x)
2. Set index = 0
3. Repeat while index < N:
4.     If LA[index] == x then:
5.         Return index
6.     Index = index + 1
7. End of Loop



အသိရေးပြတဲ့ algorithm(2) ကို အသုံးပြုပြီး trace လိုက်ကြည့် ကြည့်ပါ။

1. LA => {10, 20, 30, 40, 50, 60}, X = 10
2. LA => {10, 20, 30, 40, 50, 60}, X = 5

### **Binary Search Algorithm**

ငယ်စဉ်ကြီးလိုက် သို့မဟုတ် ကြီးစဉ်ငယ်လိုက် sorting စီပြီးသား array မှာ မှ Binary Search Algorithm ကို အသုံးပြုပြီး ရှာလိုရမှာ ဖြစ်ပါတယ်။

Binary search ဟာ အလယ်ကနေ ဖြတ်ပြီး တိုက်စစ်ပါတယ်။ ဒါကြောင့် အလယ်အန်းနံပါတ် Mid ကို Mid = (LB + UB)/2 ဆိုပြီး ထွက်ပေးရပါတယ်။

ရှာချင်တဲ့ x နှင့် Mid အခန်းထဲက တန်ဖိုးတွေကို တိုက်စစ်ပါမယ်။ ဒီနေရာမှာ ပြောချင်တာက တစ်ခုနဲ့ တစ်ခု တိုက်စစ်ခဲ့ရင် ဖြစ်နိုင်ခြာ ၃ ခုပဲရှိပါတယ်။ ငယ်တာရယ်၊ ညီတာရယ် နဲ့ ကြီးတာရယ် ဖြစ်ပါတယ်။

- ညီခဲ့ရင် Mid အခန်းမှာ တွေ့တာ ဖြစ်လို့ တွေ့တဲ့ အခန်းနံပါတ်ဖြစ်တဲ့ Mid ကို return ပြန်ပါမယ်။

| AA | BB | CC | DD | JJ  | KK | LL | XX | YY | ZZ |
|----|----|----|----|-----|----|----|----|----|----|
| 0  | 1  | 2  | 3  | 4   | 5  | 6  | 7  | 8  | 9  |
| LB |    |    |    | Mid |    |    |    |    | UB |

- ရှာချင်တဲ့ x တန်ဖိုးက Mid အခန်းထဲက တန်ဖိုးထက် ငယ်ခဲ့ရင် array ဟာ ငယ်စဉ်ကြီးလိုက် စီထားတာဖြစ်လို့ Mid ရဲ့ ငယ်တဲ့ ရှုံးတစ်ခြမ်းကိုပဲ ရှာဖို့လိုပါတယ်။ ပြထားတဲ့ table ကို ကြည့်ကြည်ပါ။ ရှုံးတစ်ခြမ်းမှာ LB က ရှိပြီးသားဖြစ်တဲ့ အတွက် LB ကို ဘာမှ လုပ်ဖို့မလိုပါဘူး။ ဒါကြောင့် ရှုံးတစ်ခြမ်းမှာ မပါတဲ့ UB ကို Mid ရဲ့ ရှုံးတစ်ခန်း Mid-1 ကို ရွှေ့ပါမယ်။
- ရှာချင်တဲ့ x တန်ဖိုးက Mid အခန်းထဲက တန်ဖိုးထက် ကြီးခဲ့ရင် array ဟာ ငယ်စဉ်ကြီးလိုက် စီထားတာဖြစ်လို့ Mid ရဲ့ ကြီးတဲ့ အနောက်တစ်ခြမ်းကိုပဲ ရှာဖို့လိုပါတယ်။ ပြထားတဲ့ table ကို ကြည့်ကြည်ပါ။ အနောက်တစ်ခြမ်းမှာ UB က ရှိပြီးသားဖြစ်တဲ့ အတွက် UB ကို ဘာမှ လုပ်ဖို့မလိုပါဘူး။ ဒါကြောင့် နောက်တစ်ခြမ်းမှာ မပါတဲ့ LB ကို Mid ရဲ့ နောက်တစ်ခန်း Mid+1 ကို ရွှေ့ပါမယ်။

ပထမ စစ်ချင်းဆိုရင် LB က UB ထက်ငယ်ပါတယ်။ ရှာချင်တဲ့ ကောင်က ငယ်ခဲ့ရင် UB = Mid - 1 ဆိုတာ UB ကို ရှုံးတိုးလိုက်တာ၊ UB လျော့လာတာ ဖြစ်ပါတယ်။ ရှာတဲ့ ကောင်က ကြီးခဲ့ရင် LB = Mid + 1 ဆိုတာ LB ကို နောက်ရွှေ့လာတာ၊ LB ကို တိုးလာတာ ဖြစ်ပါတယ်။ ဒီတော့ လျော့လာတဲ့ UB နဲ့

တိုးလာတဲ့ LB တို့ ဆုံမိရင် (ညီကြပြီဆိုရင်) ရှာတာ တစ်ပတ်ပြည့်တော့တာပါ။ ဆက်လုပ်လို့ UB က LB ထက်ငယ်သွားရင် ရှာတာ တစ်ပတ်ပြည့်ပြီးလို့ မတွေ့တော့တဲ့ အခြေအနေဖြစ်ပါတယ်။

| AA | BB | CC | DD | JJ | KK | LL | XX | YY | ZZ |
|----|----|----|----|----|----|----|----|----|----|
| 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  |

Array အမည်ကို LA လိုပဲဆိုကြပါစို့။ အထဲမှာ ရှိတဲ့ တန်ဖိုးတွေကို A to Z ငယ်စဉ်ကြီးလိုက် sorting စီပြီးသားဖြစ်လို့ Binary Search ကိုသုံးလို့ရပါတယ်။ x= CC ကိုရှာမည် ဆိုပါတော့။

LB =0, UB = 9



Mid =  $(LB+UB)/2 = (0+9)/2 = 4$  (ကိန်းပြည့်ကို ကိန်းပြည့်နဲ့စား ကိန်းပြည့်ရပါတယ်)

ဒီတော့ LA[Mid], LA[4] နဲ့ x ကို တိုက်စစ်တယ်။ JJ နဲ့ CC ဆိုတော့ user ရှာချင်တဲ့ CC က ငယ်နေတယ်။  
ဒီတော့ ရှုံးတစ်ခြမ်းကို ရှာမယ်။ ရှုံးတစ်ခြမ်းဆိုတော့ UB ကို Mid -1 ကို ရွှေ့မယ်။

UB = Mid – 1 = 3



LB(0) ဟာ UB(3) ထက်ငယ်ပြီး ညီနေသေးတယ်၊ ဒီတော့ရှာတာ တစ်ပတ်မပြည့်သေးဘူး။

Mid =  $(LB+UB)/2 = (0+3)/2 = 1$

ဒီတော့ LA[Mid], LA[1] နဲ့ x ကို တိုက်စစ်တယ်။ BB နဲ့ CC ဆိုတော့ ရှာချင်တဲ့ CC က ကြီးနေတယ်။  
ဒီတော့ LB ကို Mid + 1 ကို ရွှေ့မယ်။

LB = Mid + 1 = 1+ 1 =2



LB(2) ဟာ UB(3) ထက်ယော်ပြီး ညီနေသေးတယ်၊ ဒီတော့ရှာတာ တစ်ပတ်မပြည့်သေးဘူး။

$$\text{Mid} = (\text{LB} + \text{UB})/2 = (2+3)/2 = 2$$

ဒီတော့  $\text{LA}[\text{Mid}]$ ,  $\text{LA}[2]$  နဲ့  $x$  ကို တိုက်စစ်တယ်။ CC နဲ့ CC ဆိုတော့ ညီသွားပြီ၊ တွေ့သွားပြီ ဖြစ်လို တွေ့တဲ့ အခန်းနံပါတ်  $2(\text{Mid})$  ကို return ပြန်ပေးလိုက်ပါတယ်။ မတွေ့ရင် -1 ကို ပြန်ပေးမှာ ဖြစ်ပါတယ်။ ကဲ algorithm ကို ကြည့်ကြည့်လိုက်ရအောင်။

### **Algorithm: Binary Search**

1. `binarySearch(LA, LB, UB, x)`
2.     Repeat while  $\text{LB} \leq \text{UB}$ :
3.          $\text{Mid} = (\text{LB} + \text{UB})/2$
4.         If  $x == \text{LA}[\text{Mid}]$  then:
5.             Return  $\text{Mid}$
6.         Else if  $x < \text{LA}[\text{Mid}]$  then:
7.              $\text{UB} = \text{Mid} - 1$
8.         Else
9.              $\text{LB} = \text{Mid} + 1$
10.       End of Loop
11.       Return -1



trace လိုက်ကြည့် ကြည့်ပါ။

1.  $\text{LA} \Rightarrow \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 122, 170\}$ ,  $X = 20$
2.  $\text{LA} \Rightarrow \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 122, 170\}$ ,  $X = 500$
3. ပြထားပေးတဲ့ Binary Search Algorithm ဟာ ငယ်စဉ်ကြီးလိုက်စီထားတဲ့ array အတွက် ရေးသားပေးထားတာဖြစ်ပါတယ်။ ဒါကြောင့် အဲဒီ algorithm ကို ကြီးစဉ်ငယ်လိုက်စီထားတဲ့ array အတွက် ပြောင်းရေးပေးပါ။

4. ပြောင်းရေးထားတဲ့ algorithm ဖြင့် LA => {555, 444, 333, 222, 111, 99, 88, 77, 66, 55, 44, 33, 22, 11}, X = 22 ကို traceလိုက်ပါ။

## Maximum and Minimum Searching Algorithms

### Minimum

Array အခန်းထဲက အငယ်ဆုံးကို ရှာချင်တယ်ဆိုရင် ရှုံးခံးအခန်း(0 အခန်း) ကို အငယ်ဆုံးလို ယူဆပါ။ ရှုံးခံးအခန်းကို အငယ်ဆုံးလို ယူဆ ပြီးပြီ ဖြစ်လို့ ရှုံးခံးအခန်းရဲ့ နောက်တစ်ခန်း (အခန်းနံပါတ် ၁) ကနေ နောက်ဆုံးအခန်းထိ တစ်ခန်းပြီး တစ်ခန်း min တန်ဖိုးနဲ့ တိုက်စစ်ပါ။ min ဟာ ငယ်ရမှာ ဖြစ်ပါတယ်။ min က ကြီးနေရင် ငယ်တဲ့ ကောင်နဲ့ ပြောင်းထည့်ပါ။

#### *Algorithm: Find Minimum*

1. findMin(LA, UB)
2. Set min = LA [0]
3. Repeat for index = 1 to UB by 1:
4.     If min > LA[index] then:
5.         min = LA[index]
6.     End of Loop
7.     Return min

LA = { 20, 15, 30, 5}, UB = 3

| Line | Min | index | အလုပ်လုပ်ပုံ                                   |
|------|-----|-------|------------------------------------------------|
| 2    | 10  |       | min = LA [0] - ရှုံးခံးခန်းကို အငယ်ဆုံးလိုယူဆ။ |
| 3    |     | 1     | For initialization: index =1                   |

|          |   |  |                                                                                                                                                                                                                                                                        |
|----------|---|--|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|          |   |  | Condition: to UB ဆိတာ <=UB လိုပြောတာဖြစ်ပါတယ်။<br><br>Index<= UB → 1 <= 3 ဆိတော့ true ထွက်လို line 4 ကို လုပ်မယ်။                                                                                                                                                      |
| <b>4</b> |   |  | min > LA[index]<br><br>min > LA [1] → 10 > 15 ဆိတော့ false ထွက်လို line 5 ကို မလုပ်တော့ဘဲ line 6 ကိုသွားမယ်။                                                                                                                                                           |
| <b>6</b> |   |  | End of loop ဆိုတော့ line 3 ပြန်တက်။                                                                                                                                                                                                                                    |
| <b>3</b> | 2 |  | For looping မှာ အောက်ကနေ အပေါ်ကို ပြန်တက်လာရင် update လုပ်၊ ပြီးရင် condition စစ်ရပါတယ်။<br><br>Update : by ဆိတာ index = index + 1 = 1+ 1 = 2<br><br>Condition: to UB ဆိတာ <=UB လိုပြောတာဖြစ်ပါတယ်။<br><br>Index<= UB → 2 <= 3 ဆိတော့ true ထွက်လို line 4 ကို လုပ်မယ်။ |
| <b>4</b> |   |  | min > LA[index]<br><br>min > LA [2] → 10 > 30 ဆိတော့ false ထွက်လို line 5 ကို မလုပ်တော့ဘဲ line 6 ကိုသွားမယ်။                                                                                                                                                           |
| <b>6</b> |   |  | End of loop ဆိုတော့ line 3 ပြန်တက်။                                                                                                                                                                                                                                    |
| <b>3</b> | 3 |  | For looping မှာ အောက်ကနေ အပေါ်ကို ပြန်တက်လာရင် update လုပ်၊ ပြီးရင် condition စစ်ရပါတယ်။<br><br>Update : by ဆိတာ index = index + 1 = 2+ 1 = 3<br><br>Condition: to UB ဆိတာ <=UB လိုပြောတာဖြစ်ပါတယ်။<br><br>Index<= UB → 3 <= 3 ဆိတော့ true ထွက်လို line 4 ကို လုပ်မယ်။ |
| <b>4</b> | 5 |  | min > LA[index]<br><br>min > LA [3] → 10 > 5 ဆိတော့ true ထွက်လို line 5 ကိုလုပ်မယ်။                                                                                                                                                                                    |

|   |  |   |                                                                                                                                                                                                                                                                                                      |
|---|--|---|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|   |  |   | $\min = \text{LA}[\text{index}] = \text{LA}[3] = 5$                                                                                                                                                                                                                                                  |
| 6 |  |   | End of loop ဆိုတော့ line 3 ပြန်တက်။                                                                                                                                                                                                                                                                  |
| 3 |  | 4 | <p>Update : by ဆိုတာ <math>\text{index} = \text{index} + 1 = 3 + 1 = 4</math></p> <p>Condition: to UB ဆိုတာ <math>\leq \text{UB}</math> လိုပြောတာဖြစ်ပါတယ်။</p> <p><math>\text{Index} \leq \text{UB} \rightarrow 4 \leq 3</math> ဆိုတော့ false ထွက်လို့ end of loop အပြင်ဖက် line 7 ကို လုပ်မယ်။</p> |
| 7 |  |   | Return $\min$ ဆိုတော့ return 5 ဆိုပြီး ခေါ်တဲ့ နေရာဆီကို return ပြန်ပေးသွားမှာ ဖြစ်ပါတယ်။                                                                                                                                                                                                            |

Min ဆိုတော့ ငယ်ရမှာ၊ ဒါကြောင့်  $\min$  ကြိုးနေခဲ့ရင် ပြောင်းထည့်တာဖြစ်ပါတယ်။ Maximum မှာ ကျန်တာ အတူတူပါပဲ။ Maximum ဆိုတော့ ကြိုးရမှာဖြစ်လို့ ငယ်ရင် ပြောင်းထည့်ရမှာ ဖြစ်ပါတယ်။ အဲဒီ တစ်နေရာလေးပဲ ကွဲတာ ဖြစ်ပါတယ်။



### လောက့် ကြည့်ပါ။

1. `findMaximum(LA, UB)` algorithm ကိုရေးပါ။
2.  $\text{LA} \Rightarrow \{10, 70, 35, 77, 6, 600, 4\}$  ထဲမှ Maximum ကို Question 1 ရဲ့ `findMaximum` algorithm ဖြင့် trace လိုက်ပါ။

“ပထမနစ်တက်တုန်းက

စာမေးပွဲနားနီးတော့မှ

သူငယ်ချင်းက ဟဲ့ ငါ data structure မေးအံးမယ် ဆိုတော့

ရှင်းပြတော့မယ်ပေါ့၊

မေးလိုက်တာက

‘Trace လိုက်တယ် ဆိုတာ ဘာတုန်းတဲ့’

အတွေးက အဲဒီလိုလွန်ကြတာ၊

အဲဒီလို ဖြစ်နေအံးမယ်။”

## အခန်း (၁၃)

### Stack, Queue and Hashing

#### Stack

Stack ရဲ့ နာမည်ကြီးတဲ့ ဥပမာတွေကတော့ Pancake ထပ်တာ နဲ့ ပန်းကန် ထပ်တာ တို့ပဲဖြစ်ပါတယ်။



ပုံလေးတွေကို ကြည့်ကြည့်ပါ။ Pancake လုပ်သူဟာ ပန်းကန်ပြားထဲကို pancake တစ်ခုပေါ် နောက်တစ်ခုကို ထပ်ထပ်ပြီး တင်သွားတာဖြစ်ပါတယ်။ ဒီတော့ သူနောက်ဆုံးတင်ခဲ့တဲ့ pancake ဟာ အပေါ်ဆုံးမှာ ရှိနေမှာဖြစ်ပါတယ်။ စားသုံးမည့်သူရဲ့ နေရာကနေ ဝင်ပြီး စဉ်းစားကြည့်ပါ။ အောက်ဆုံးတစ်ခုကနေ ဖြစ်ဖြစ်၊ အလယ်တစ်ခုကနေဖြစ်ဖြစ် ယူစားမယ်ဆိုရင် သေချာပေါက်ပြီကျမှာဖြစ်ပါတယ်။ စားမည့်သူကလည်း အဲဒီလို ခက်ခက်ခဲဲ ဘယ်သူမှ ယူမစားပါဘူး။ အပေါ်ဆုံးတစ်ခုကနေ ယူပြီးစားမှာ ဖြစ်ပါတယ်။

ဒါကြောင့် pancake လုပ်သူနောက်ဆုံးလုပ်ထားတဲ့ မုန်က အပေါ်ဆုံးမှာ ရှိတယ်။ စားတဲ့သူက အပေါ်ဆုံးကဟာဘာကို အရင် ယူစားလိုက်တယ်။ ဒီတော့ နောက်ဆုံးလုပ်ထားတဲ့မုန်ကို အရင်စားတယ်တနည်းအားဖြင့် နောက်ဆုံးထည့်တာ အရင်ဆုံးထွက်တာဖြစ်လို့ LAST IN FIRST OUT (LIFO) – STACK လို့ ခေါ်ပါတယ်။

ဒီလိပါပဲ ပန်းကန်ပြားဆေးတဲ့သူဟာ ဆေးဆေးပြီး ပန်းကန်ပြားကို အပေါ်က ထပ်ထပ်သွားမယ်။ ထမင်းစားမယ်ဆိုပြီးယူရင် အပေါ်ကနေပဲယူတယ်။ ဒီတော့ နောက်ဆုံးထည့်တဲ့ ပန်းကန်ကို အရင်ဆုံးပြန်သုံးတာဖြစ်လို့ ပန်းကန်ပြားဆေးတာကလည်း Stack ပုံစံ အလုပ်လုပ်တာပဲဖြစ်ပါတယ်။

ထပ်စဉ်းစားကြည့်မယ်ဆိုရင် ထည့်တယ်၊ အပေါ်ကနေ ထည့်ထည့်သွားတယ်။ ယူတယ် အပေါ်ကနေပဲ ယူယူသွားတယ်။ ဒါကြောင့် အပေါ်ဆိုတဲ့ တစ်ဖက်ကနေပဲ အလုပ်လုပ်သွားတာဖြစ်လို့ ထောက်ဖိုရာ တစ်ခုပဲလိပါတယ်။ အဲဒါကို top လို့ ခေါ်ပါတယ်။



နှမူနာလေး တွက်ကြည့်ကြရအောင်။

LA ဆိုတဲ့ array မှာ Length က 5 ဖြစ်တယ်ဆိုပါတော့။ Array ခန်း 5 ခန်းဆိုတော့ အခန်းနံပါတ်က 0 to 4 ဖြစ်ပါမယ်။ ထည့်ရင် 0 ခန်းကနေစပြီး ထည့်ပါမယ်။ ဒါကြောင့် top = 0 ဆိုပြီး top ထဲ 0 ထည့်ထားလိုက်ပါမယ်။ အဲဒီထဲကို AA,BB,CC,DD, EE,FF ဆိုပြီးထည့်ကြည့်ကြပါမယ်။



စစ်ဆေး တစ်ခန်းမှ မထည့်ရသေးတော့ top = 0

|   |   |   |   |   |
|---|---|---|---|---|
|   |   |   |   |   |
| 0 | 1 | 2 | 3 | 4 |

ဆိုပြီးဖြစ်နေပါမယ်။ ဒီတော့ top က 0 ဖြစ်နေတယ် ဆိုတာ တစ်ခန်းမှ မရှိလို့ဖြစ်ပါတယ်။ ဒါကြောင့် Stack ဟာ empty ဖြစ်နေသလား(တစ်ခန်းမှ မရှိဘူးလား) စစ်ဆေးရင် top က 0 နဲ့ ညီသလား စစ်လိုက်ရင်ရပါတယ်။

*isEmpty()*

*Return top == 0*

ဆိုပြီး ရေးလိုက်ရင်ရပါပြီ။



AA ထည့်မယ်။ ထည့်တာကို stack မှာ push လုပ်တယ်လို့ ခေါ်ပါတယ်။ top တန်ဖိုး 0 ဖြစ်နေတဲ့ အတွက် တစ်ခန်းမှ မရှိသေးလို့ 0 ခန်းကနေ စပြီးထည့်ပါမယ်။ ထည့်ပြီးတာနဲ့ top ကို ၁ တိုးပါမယ်။

|    |   |   |   |   |
|----|---|---|---|---|
| AA |   |   |   |   |
| 0  | 1 | 2 | 3 | 4 |

Top ကို ၁ တိုးလိုက်တော့ top = 1 ဖြစ်သွားပါမယ်။



BB ထည့်မယ်။ ၀ အခန်းထည့်ပြီးတာဖြစ်လို့ ဒီတစ်ခါ ၁ အခန်းထဲကို ထည့်ပါမယ်။ စဉ်းစားကြည့်လိုက်တော့ top 0 ဖြစ်နေတူန်းက ၀ အခန်းထဲ ထည့်ပြီး၊ top 1 ဖြစ်နေတော့ ၁ အခန်းထဲ ထည့်တယ်။ ဒီတော့ top အခန်းထဲ ထည့်တယ် လို့ မှတ်ရင်ရပါပြီ။ ထည့်ပြီးတာနဲ့ top ကို ၁ တိုးပါမယ်။

|    |    |   |   |   |
|----|----|---|---|---|
| AA | BB |   |   |   |
| 0  | 1  | 2 | 3 | 4 |

top က ၂ ဖြစ်သွားပါမယ်။



CC ထည့်မယ်။ top က ၂ ဖြစ်နေတော့ ၂ အခန်းထဲ CC ထည့်မယ်။

|    |    |    |   |   |
|----|----|----|---|---|
| AA | BB | CC |   |   |
| 0  | 1  | 2  | 3 | 4 |

ထည့်ပြီးတာနဲ့ top ကို ၁ တိုးတော့ top က ၃ ဖြစ်သွားပါမယ်။



DD ထည့်မယ်။ top က 3 ဖြစ်နေတော့ 3 အခန်းထဲ DD ထည့်မယ်။

|    |    |    |    |   |
|----|----|----|----|---|
| AA | BB | CC | DD |   |
| 0  | 1  | 2  | 3  | 4 |

ထည့်ပြီးတာနဲ့ top ကို 1 တိုးတော့ top က 4 ဖြစ်သွားပါမယ်။



EE ထည့်မယ်။ top က 4 ဖြစ်နေတော့ 4 အခန်းထဲ EE ထည့်မယ်။

|    |    |    |    |    |
|----|----|----|----|----|
| AA | BB | CC | DD | EE |
| 0  | 1  | 2  | 3  | 4  |

ထည့်ပြီးတာနဲ့ top ကို 1 တိုးတော့ top က 5 ဖြစ်သွားပါမယ်။



FF ထည့်မယ်ဆိုတော့ EE ထည့်ပြီးကတည်းက stack array က အခန်းပြည့်သွားပြီ ဖြစ်ပါတယ်။  
ထပ်ထည့်လို့မရတော့ပါဘူး။ ဒါကြောင့် stack ထဲကို ထည့်တော့မယ် ဆိုရင် အခန်းပြည့်  
နေပြီလား စစ်ဖို့လိုပါတယ်။ မပြည့်သေးမှ ထပ်ထည့်လို့ရမှာဖြစ်ပါတယ်။ ကြည့်ကြည့်လိုက်တော့ top  
တန်ဖိုး 5 မှာ ဆက်ထည့်လို့မရတော့ဘူး။ ဘာကြောင့်လဲဆိုရင် array ရဲ့ length ဟာ 5 ဖြစ်နေတော့ 4  
အခန်းထိပ်ရှိလိုပါ။ ဒါကြောင့် top နဲ့ length ညီရင် stack array ပြည့်သွားပြီ ပြောလို့ရပါတယ်။

*isFull()*

*Return top == N*

ဒီမှာ သုံးထားတဲ့ N ဆိုတာ array ရဲ့ length တန်ဖိုးပဲ ဖြစ်ပါတယ်။ ဒီတော့ ထည့်ရင် မပြည့်သေးမှ  
ထည့်လို့ရမှာ ဖြစ်လို့ ပြည့်မပြည့် စစ်ပေးရမှာ ဖြစ်ပါတယ်။

```
push(Item)
```

*if isFull() then:*

*return false*

*Else:*

*Stack[top] = Item*

*top = top + 1*

*return true*

ထည့်တဲ့ push function ထဲမှာ isFull ဆိုတဲ့ function ကို ခေါ်ပြီး စစ်လိုက်တယ်။ Full ဖြစ်နေရင် ထည့်မရတော့တဲ့ အကြောင်း false ပြန်ပေးလိုက်တယ်။ မဆုံးတော့တာကို ထပ်ထည့်တာကို OVERFLOW လို့ခေါ်ပါတယ်။ မှတ်မိအောင်ပြောရရင် ကိုယ့်ခေါင်းက လက်မခံနိုင်တော့ဘူး၊ ဆရာမက အတင်းစာဆက်သင်နေရင် ‘ခေါင်းရဲ့မှတ်နိုင်စွမ်းက overflow ဖြစ်နေပြီ’လို့ ပြောလို့ရပါတယ်။ ဆက်ပြောရရင် full မဖြစ်သေးဘူး ဆိုရင် top အခန်းထဲကို ထည့်တယ်။ top ကို 1 တိုးတယ်။ ပြီးရင် ထည့်တာ အောင်မြင်ကြောင်း true return ပြန်ပေးလိုက်တယ်။



ပြန်ဆဲထုတ်ပျေမယ်။ ထုတ်တာကို Stack မှာ pop လုပ်တယ်လို့ ခေါ်ပါတယ်။ stack ဟာ နောက်ဆုံးဝင်တာ အရင်ထုတ်မှာ ဖြစ်တာကြောင့် top ကနေ ဆဲထုတ်မှာဖြစ်ပါတယ်။

|    |    |    |    |    |
|----|----|----|----|----|
| AA | BB | CC | DD | EE |
| 0  | 1  | 2  | 3  | 4  |

top တန်ဖိုး 5 ဖြစ်နေတယ်။ ထုတ်မယ်ဆို EE အခန်း 4 ကို ထုတ်ရမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် top 5 ဆိုရင် 4 အခန်းထုတ်ချင်တာဖြစ်လို့ top ကို အရင် 1 လျော့ပြီးမှ top အခန်း ဆဲထုတ်ရမှာ ဖြစ်ပါတယ်။ top ကို ၁ လျော့လိုက်တော့ top တန်ဖိုးက 4 ဖြစ်မယ်။

top အခန်း return ပြန်တော့ return Stack[4] ဆိုတော့ EE ဆိုပြီး နောက်ဆုံးထည့်ခဲ့တဲ့ အခန်းပြန်ရမယ်။ stack ကတော့ ဒီလို သဘော ဖြစ်သွားမယ်။

|    |    |    |    |   |
|----|----|----|----|---|
| AA | BB | CC | DD |   |
| 0  | 1  | 2  | 3  | 4 |

ဒါကြောင့် pop ကို ရေးကြည့်မယ်ဆိုရင် array ခန်းက empty ဖြစ်နေရင် ဆွဲထုတ်လို့မရတော့ဘူး။ မရှိတာကို ဆွဲထုတ်တာကို “UNDERFLOW” ဖြစ်တယ်လို့ ခေါ်ပါတယ်။

---

*pop()*

---

*If isEmpty() then:*

*Return NULL*

*Else:*

*top = top - 1*

*Retrun Stack[top]*

---

ဒီမှာ အပေါ်မှာ ရေးခဲ့တဲ့ isEmpty ဆိုတဲ့ function ကို ပြန်ခေါ်သုံးပြီး စစ်လိုက်တာဖြစ်ပါတယ်။



Empty မဖြစ်မချင်း pop လုပ်ကြည့်ရအောင်။ top က 4

|    |    |    |    |   |
|----|----|----|----|---|
| AA | BB | CC | DD |   |
| 0  | 1  | 2  | 3  | 4 |

top 1 လျော့တော့ top =3, ဒီတော့ return Stack[3] → retrun DD

|    |    |    |   |   |
|----|----|----|---|---|
| AA | BB | CC |   |   |
| 0  | 1  | 2  | 3 | 4 |

Empty မဖြစ်သေးတော့ top 1 လျှော့တော့ top =2, ဒီတော့ return Stack[2] → retrun CC

|    |    |   |   |   |
|----|----|---|---|---|
| AA | BB |   |   |   |
| 0  | 1  | 2 | 3 | 4 |

Empty မဖြစ်သေးတော့ top 1 လျှော့တော့ top =1, ဒီတော့ return Stack[1] → retrun BB

|    |   |   |   |   |
|----|---|---|---|---|
| AA |   |   |   |   |
| 0  | 1 | 2 | 3 | 4 |

Empty မဖြစ်သေးတော့ top 1 လျှော့တော့ top =0, ဒီတော့ return Stack[0] → retrun AA

|   |   |   |   |   |
|---|---|---|---|---|
|   |   |   |   |   |
| 0 | 1 | 2 | 3 | 4 |

နောက်တစ်ကြိမ် စစ်တော့ top က 0 ဆိုတော့ empty ဖြစ်သွားပြီဆိုပြီးရပ်သွားပါမယ်။

ထည့်လိုက်တာက AA, BB, CC, DD, EE

Pop လုပ်ခဲ့လို့ retrun ပြန်တာလေးတွေ ကြည့်ကြည့်ပါ။ ပြန်ရတာက EE, DD, CC, BB, AA ဖြစ်ပါတယ်။

ဒါကြောင့် Stack ဟာ နောက်ဆုံးဝင်တာ အရင်ထွက်တာဖြစ်ပါတယ်။ ဒီမှာ algorithm လေးတွေကို ကြားညွှန်ညွှန်ပြီး ရေးပြခဲ့ပါတယ်။ သေချာလေးပြန်ကြည့်ပါတယ်။ နားလည်ပြီဆိုမှ အောက်က တွက်တာကို ဆက်ပြီးလုပ်ပါ။



နမူနာ တစ်ပုဒ်တွက်ပြုမယ်။ ကျန်တာတွေကတော့ လေ့ကျင့်ကြည့်ကြပါ။

1. Stack: 11, 22, 33, 44, 55, ---, ---, ----

- a) Top တန်ဖိုး ဘယ်လောက်ဖြစ်မလဲဆိုတာ Stack ကိုကြည့်ပြီး စဉ်းစားပါ။
- b) Pop()
- c) Pop()
- d) Push(66)
- e) Push(77)
- f) Pop()
- g) Push(88)
- h) Push(99)
- i) Pop()

2. Stack : A, E, I, O, U, -----

- a) Push("BB")
- b) Pop()
- c) Push("CC")
- d) Push("DD")
- e) Pop()
- f) Push("EE")

3. Stack is empty and find the output. N is 6.

- a) start = 5
- b) end = 100
- c) Push(start)
- d) Push(end)
- e) Push(summation of start to end)
- f) Push(count of numbers from start to end)
- g) Push(average of start to end)
- h) Repeat while !isEmpty():

Write pop (), “ ”

**Answer of No1:**

Stack: 11, 22, 33, 44, 55, ---, ---, ----

a) top = 5

b) Pop()

top = 4 and Stack: 11, 22, 33, 44, ----, ---, ----, ---- and return 55

c) Pop()

top = 3 and Stack: 11, 22, 33, ----, ----, ---, ----, ---- and return 44

d) Push(66)

Stack: 11, 22, 33, 66, ----, ---, ---- and top = 4

e) Push(77)

Stack: 11, 22, 33, 66, 77, ---, ----, ---- and top = 5

f) Pop()

top= 4 and Stack: 11, 22, 33, 66, ---, ----, ---- and return 77

g) Push(88)

Stack: 11, 22, 33, 66, 88, ---, ----, ---- and top = 5

h) Push(99)

Stack: 11, 22, 33, 66, 88, 99, ----, ---- and top = 6

i) Pop()

top= 5 and Stack: 11, 22, 33, 66, 88, ---, ----, ---- and return 99

## Example of Stack

ကွင်းစ ကွင်းပိတ် မှန်မှန် စစ်တာလေး ကို စဉ်းစားကြည့်ကြရအောင်ပါ။

$[a * (2 + b)] / 2$

ဒီ equation မှာ ကွင်းစ ကွင်းပိတ်တွေ မှန်သလား ဆိုတော့ မှန်ပါတယ်။ ဒီမှာ [ - ထောင့်ကွင်းအစ အရင်လာပါတယ်။ ပြီးတော့ ( - လက်သည်းကွင်းစ လာပါတယ်။ ဟော ပိတ်တော့ လက်သည်းကွင်းက အရင်ပိတ်ရတယ်။ ပြီးမှ ထောင့်ကွင်းက ပိတ်ရတယ်။

ဒီတော့ နောက်ဆုံး ထည့်ခဲ့တဲ့ ကွင်းစရဲ့ အပိတ်က အရင်လာရတာဖြစ်လို့ stack ကို သုံးပြီး စစ်လို့ရပါတယ်။ တွန့်ကွင်း {, ထောင့်ကွင်း [, လက်သည်းကွင်း ( ဘယ်ကွင်းစ လာလာ Stack ပေါ်တင်ပါမယ်။

- ဝင်လာတဲ့ equation ထဲက character တစ်လုံးချင်း ယူပါမယ်။

- Character ဟာ ကွင်းစဖြစ်ခဲ့ရင် ဘာကွင်းစ ဖြစ်ဖြစ် Stack ပေါ်တင်ပါမယ်။ နောက်ဆုံးတင်တဲ့ ကွင်းစ ကို အရင်စစ်ရမှာ ကြောင့် Stack ကို အသုံးပြုတာ ဖြစ်ပါတယ်။
- ကွင်းပိတ်လာပြီဆိုတာနဲ့ Stack ပေါ်တင်ထားတဲ့ ကွင်းစ တစ်ခုကို pop လုပ်ပြီး လက်သည်းကွင်း လက်သည်းကွင်းချင်း၊ ထောင့်ကွင်း ထောင့်ကွင်းချင်း၊ တွန်ကွင်း တွန်ကွင်းချင်း တူးမတူး စစ်ပါမယ်။ မတူးရင် ကွင်းစ ကွင်းပိတ်လွှဲနေတာ ဖြစ်လို့ မှားပါတယ်။ (Unmatch)
- တကယ်လို့ ကွင်းပိတ်က ဝင်လာပြီး stack က empty ဖြစ်သွားပြီဆိုရင် ကွင်းပိတ်ပိုတဲ့ error ဖြစ်ပါတယ်။ (Extra close bracket)
- တကယ်လို့ ဝင်လာစရာကွင်းပိတ်က မရှိတော့ဘူး၊ stack ပေါ်မှာက ကျန်နေသေးတယ် ဆိုရင် ကွင်းစ ပိုတဲ့ error ဖြစ်ပါတယ်။ (Extra Open Bracket)
- အပေါ်ကပြောတဲ့ အချက်တွေ တစ်ခုမှ မဖြစ်ခဲ့ဘူးဆိုရင်တော့ ကွင်းစ ကွင်းပိတ် မှန်လို့ ဖြစ်ပါတယ်။ (Correct)



$[a * (2 + b) } / 2$

| ဝင်လာတဲ့ Character | အလုပ်လုပ်ပုံ                                  | Stack |
|--------------------|-----------------------------------------------|-------|
| [                  | ကွင်းစဆိုတော့ Stack ပေါ်တင်မယ်။               | [     |
| A                  | ကွင်းစ ကွင်းပိတ် မဟုတ်တော့ ကျော်ပါမယ်။ (Skip) |       |
| *                  | Skip                                          |       |
| (                  | ကွင်းစဆိုတော့ Stack ပေါ်တင်မယ်။               | [, (  |
| 2                  | Skip                                          |       |

|   |                                                                                                                                                                                                                                                                                              |   |
|---|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|
| + | Skip                                                                                                                                                                                                                                                                                         |   |
| B | Skip                                                                                                                                                                                                                                                                                         |   |
| ) | ကွင်းပိတ်ဆိုတော့ stack ပေါ်က pop လုပ်လိုက်တယ်။ ( ရတယ်။ stack<br>ပေါ် တစ်ခု လျော့သွားမယ်။ ရလာတဲ့ ( - လက်သည်းကွင်းစနဲ့ ဝင်လာတဲ့<br>) - လက်သည်းကွင်း အပိတ်နဲ့ တိုက်စစ်တော့ လက်သည်းကွင်းချင်း<br>တူတော့ မှန်သေးတော့ ဆက်တိုက်မယ်။                                                                 | [ |
| } | ကွင်းပိတ်ဆိုတော့ stack ပေါ်က pop လုပ်လိုက်တယ်။ [ ရတယ်။ stack<br>ပေါ် တစ်ခု လျော့သွားမယ်။ ရလာတဲ့ [ - ထောင့်ကွင်းစနဲ့ ဝင်လာတဲ့ } -<br>တွန်ကွင်း အပိတ်နဲ့ တိုက်စစ်တော့ လက်သည်းကွင်းနဲ့ တွန်ကွင်း<br>မတူတော့ လို့ “Unmatch” ဆိုပြီး မတူကြောင်းထုတ်ပြမယ်။ မှားသွားလို့<br>ဆက်တိုက်စရာမလိုတော့ဘူး။ |   |



{e + [a \* (2 + b)] / 2

| ဝင်လာတဲ့ character | အလုပ်လုပ်ပုံ                 | Stack |
|--------------------|------------------------------|-------|
| {                  | ကွင်းစဆိုတော့ Stack ပေါ်တင်။ | {     |
| E                  | Skip                         | {     |
| +                  | Skip                         | {     |
| [                  | ကွင်းစဆိုတော့ Stack ပေါ်တင်။ | {, [  |
| A                  | Skip                         | {, [  |
| *                  | Skip                         | {, [  |

|   |                                                                                                                                                                                                           |         |
|---|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------|
| ( | ကွင်းစဆိုတော့ Stack ပေါ်တင်။                                                                                                                                                                              | {, [, ( |
| 2 | Skip                                                                                                                                                                                                      | {, [, ( |
| + | Skip                                                                                                                                                                                                      | {, [, ( |
| B | Skip                                                                                                                                                                                                      | {, [, ( |
| ) | ကွင်းပိတ်ဆိုတော့ Stack ပေါ်က တစ်ခု pop လုပ်မယ်။ (- လက်သည်းကွင်းအပိတ်ဖြစ်တဲ့ အတွက် တူတယ်။ ဆက်တိုက်မယ်။                                                                                                     | {, [    |
| ] | ကွင်းပိတ်ဆိုတော့ Stack ပေါ်က တစ်ခု pop လုပ်မယ်။ [- ထောင့်ကွင်းစရာတယ်။ ဝင်လာ တာဟာ )-လက်သည်းကွင်းအပိတ်ဆိုတော့ တူတယ်။ ဆက်ပြီး တိုက်မယ်။                                                                      | {       |
| / | Skip                                                                                                                                                                                                      | {       |
| 2 | Skip                                                                                                                                                                                                      | {       |
|   | Equation ထဲ က character တစ်ခုချင်း ဖြတ်ဝင်လာတာတွေ ကုန်သွားပြီဆိုတော့ Stack ကို စစ်တယ်။ Stack က empty မဖြစ်ဘေးပဲ တစ်ခုကုန်နေတာဖြစ်လို့ ကွင်းစပိုတာ ဖြစ်တဲ့ အတွက် “Extra open bracket” ဆိုပြီး အဖြေထွက်မယ်။ | {       |



လေ့ကျင့် ကြည့်ပါ။

- [a + {(b - 3)-(4 - e)} / 2] ကို လေ့ကျင့်ကြည့်ပါ။
- [a + {(b - 3)-(4 - e)} / 2 ကို လေ့ကျင့်ကြည့်ပါ။
- [a + {(b - 3)-(4 - e)} / 2 ] ကို လေ့ကျင့်ကြည့်ပါ။

## Bracket Checker Algorithm ကို ရေး ပြရရင်တော့

bracketChecker (equation, stack)

1. Set index = 0
2. Set length= LENGTH(equation)
3. Repeat while index < length:
  4. Ch = equation [index]
  5. If Ch == '(' or Ch == '[' or Ch == '{' then:
    6. stack.push(Ch)
  7. Else if Ch == ')' or Ch == ']' or Ch == '}' then:
    8. If stack.isEmpty() then:
      9. Return "Extra Close Bracket", Ch
    10. popCh = stack.pop()
    11. if (popCh == '(' and Ch != ')') or (popCh== '[' and Ch != ']')
      12. or popCh=='{' and Ch != '}' then:
        13. Return "UNMATCH"
      14. index = index + 1
  15. End of Loop
  16. If !stack.isEmpty() then:
    17. Return "Extra Open Bracket", stack.pop()

## Queue

Queue ဆိတာကတော့ ကျွန်မတို့ အားလုံး ရင်းနှီးပြီးသားဖြစ်ပါတယ်။ Counter မှာ ငွေရှင်းဖို့ တန်းစီတယ်၊ ရုပ်ရှင်လက်မှတ် တန်းစီတယ်။ တိုးဂိတ်မှာ ကားတွေတန်းစီတယ်။ ဒါတွေအားလုံးဟာ Queue လဲ ဥပမာတွေ ဖြစ်ပါတယ်။ ဝင်ပြီးတန်းစီရင် နောက်ကနေ ဝင်ပြီးတန်းစီရတယ်။ ပြီးသွားလို့

ထွက်တော့ ရှုံးကလူက အရင်ထွက်သွားတယ်။ ဒီတော့ ဝင်ရင် နောက်ကနေဝင်၊ ထွက်ရင် ရှုံးက ထွက်တာဖြစ်လို့ နောက်ကို ထောက်ဖို့ Rear နှင့် ရှုံးကို ထောက်ဖို့ Front ဆိုပြီး ထောက်ဖို့ နှစ်ခုလိုပါတယ်။ ဒါကြောင့် ဝင်ရင် Rear က ဝင်၊ ထွက်ရင် Front ကနေထွက် ရမှာဖြစ်ပါတယ်။

Counter မှာ ငွေရှင်းရင် ဘယ်သူအရင်ရှင်းပြီး ထွက်သွားခွင့်ရသလဲဆိုရင် အရင်ရောက်တဲ့ သူက အရင်ရှင်းပြီး ထွက်သွားရတာဖြစ်ပါတယ်။ အရင် ရောက်သူ အရင်ထွက်ရတာဖြစ်လို့ Queue ကို First In First Out (FIFO) လို့ ခေါ်ပါတယ်။

Queue Size : N, Number of Elements in Queue : E

- E ဆိုတာ array ခန်းထဲ ထည့်ထားတဲ့ element အရေအတွက်ကို ပြောတာဖြစ်လို့ E တန်ဖိုး 0 ဆိုတာ empty ဖြစ်နေတာ ဖြစ်ပါတယ်။
- ထည့်ရင် Number of Element (E) ကို 1 တိုးမယ်၊ ဖျက်ရင် E ကို 1 လျော့မယ်။
- E နဲ့ N ညီရင် full ဖြစ်သွားတဲ့ သဘောဖြစ်ပါတယ်။ ထပ်ထည့်လို့ မရတော့ပါဘူး။

| Operation | Front, Rear<br>and E                   | QUEUE                   | ရှင်းလင်းချက်                                                                                                        |
|-----------|----------------------------------------|-------------------------|----------------------------------------------------------------------------------------------------------------------|
|           | Front : 0<br><br>Rear : 0<br><br>E : 0 | ---, ---, ---, ---, --- | Queue Size(N) : 5<br><br>Number of Elements(E) : 0                                                                   |
| Insert A  | Front : 0<br><br>Rear : 1<br><br>E : 1 | A, ---, ---, ---, ---   | ထည့်တော့ Rear ဘက်ကထည့်၊ Rear က 0 ဆိုတော့ 0 ခန်းထည့်၊ ပြီးတာနဲ့ Rear ၁ တိုး၊ တစ်ခန်းထည့်လိုက်တာဖြစ်လို့ E ကို 1 တိုး။ |
| Insert B  | Front : 0<br><br>Rear : 2              | A, B, ---, ---, ---     |                                                                                                                      |

|                 |           |                       |                                                                                                                                                       |
|-----------------|-----------|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
|                 | E: 2      |                       |                                                                                                                                                       |
| <b>Insert C</b> | Front : 0 | A, B, C, ---, ---     |                                                                                                                                                       |
|                 | Rear : 3  |                       |                                                                                                                                                       |
|                 | E : 3     |                       |                                                                                                                                                       |
| <b>Delete</b>   | Front : 1 | ---, B, C, ---, ----  | ထုတ်မှာဖြစ်လို Front ဖက်က နေ ထုတ်।                                                                                                                    |
|                 | Rear : 3  |                       | Front 0 ဖြစ်နေလို 0 ခန်းက A ကို ထုတ်။                                                                                                                 |
|                 | E : 2     |                       | ထုတ်ပြီးတာနဲ့ Front ၁တိုး၊ E ၁ လျော့။<br><br>Return : A                                                                                               |
| <b>Delete</b>   | Front : 2 | ---, ---, C, ---, --- | ထုတ်မှာဖြစ်လို Front ဖက်က နေ ထုတ်။                                                                                                                    |
|                 | Rear : 3  |                       | Front 1 ဖြစ်နေလို 1 ခန်းက B ကို ထုတ်။                                                                                                                 |
|                 | E : 1     |                       | ထုတ်ပြီးတာနဲ့ Front ၁တိုး၊ E ၁ လျော့။<br><br>Return : B                                                                                               |
| <b>Insert D</b> | Front : 2 | ---, ---, C, D, ---   | ထည့်တော့ Rear ကထည့်၊ Rear ၂ ၃                                                                                                                         |
|                 | Rear : 4  |                       | ဆိုတော့ ၃ အခန်းထဲထည့်။ ပြီးရင် Rear ၃                                                                                                                 |
|                 | E : 2     |                       | တိုး၊ E ၁တိုး။                                                                                                                                        |
| <b>Insert E</b> | Front : 2 | ---, ---, C, D, E     | ထည့်တော့ Rear ကထည့်၊ Rear ၂ ၄                                                                                                                         |
|                 | Rear : 0  |                       | ဆိုတော့ ၄ အခန်းထဲထည့်။ ပြီးရင် Rear ၄                                                                                                                 |
|                 | E : 3     |                       | တိုး၊ E ၁တိုး မယ်။ Rear ကို ၁ တိုးမယ်<br>ဆိုတဲ့အချိန်မှာ Rear ဟာ UB(array size<br>(N) -1) နဲ့ ညီနေတယ်။ ဒီတော့ ထည့်ရင်<br>အစက ပြန်စရမှာမို့ Rear ကို ၁ |

|                                   |                                        |                     |                                                                                                                                                                                  |
|-----------------------------------|----------------------------------------|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                   |                                        |                     | မတိုးတော့ပြု Rear ထဲကို 0 ပြန်ထည့်။ E<br>ကတော့ 1 တိုး။                                                                                                                           |
| <b>Insert F</b>                   | Front : 2<br><br>Rear : 1<br><br>E: 4  | F, ---, C, D, E     | ထည့်တော့ Rear ကထည့်၊ Rear က 0<br>ဆိုတော့ 0 အခန်းထဲထည့်။ ပြီးရင် Rear ၁<br>တိုး၊ E သတိုး။                                                                                         |
| <b>Insert G</b>                   | Front : 2<br><br>Rear : 2<br><br>E : 5 | F, G, C, D, E       | ထည့်တော့ Rear ကထည့်၊ Rear က 1<br>ဆိုတော့ 1 အခန်းထဲထည့်။ ပြီးရင် Rear ၁<br>တိုး၊ E သတိုး။                                                                                         |
| <b>Insert H</b>                   |                                        |                     | Number of Element(E) ဟာ array size<br>(N) နဲ့ ညီသွားပြီဆိုတော့ ထပ်ထည့်လို့<br>မရတော့ပါဘူး (Overflow) ဖြစ်ပါမယ်။                                                                  |
| <b>Empty မဖြစ်<br/>မချင်းထုတ်</b> | Front : 3<br><br>Rear : 2<br><br>E : 4 | F,G,---, D, E       | Front က 2 ဆိုတော့ 2 အခန်း C ကိုထုတ်။<br><br>Front ၁ တိုး၊ E တစ်လျှော့။<br><br><b>Return : C</b>                                                                                  |
|                                   | Front : 4<br><br>Rear : 2<br><br>E : 3 | F, G, ---, ---, E   | Front က 3 ဆိုတော့ 3 အခန်း D ကိုထုတ်။<br><br>Front ၁ တိုး၊ E တစ်လျှော့။<br><br><b>Return : D</b>                                                                                  |
|                                   | Front : 0<br><br>Rear : 2<br><br>E : 2 | F, G, ---, ---, --- | Front က 4 ဆိုတော့ 4 အခန်း E ကိုထုတ်။<br><br>Front ၁ တိုး၊ E တစ်လျှော့ ရမယ်။ သို့သော်<br>Front ဟာ LB(N-1) နဲ့ညီသွားလို့ ၁<br>တိုးမည့်အစား 0 က ပြန်စရမယ်။<br><br><b>Return : E</b> |

|  |                                |                         |                                                                                         |
|--|--------------------------------|-------------------------|-----------------------------------------------------------------------------------------|
|  | Front : 1<br>Rear : 2<br>E : 1 | ---, G, ---, ---, ---   | Front က 0 ဆိုတော့ 0 အခန်း F ကိုထုတ်။<br>Front ၁ တိုး၊ E တစ်လျှော့။<br><b>Return : F</b> |
|  | Front : 2<br>Rear : 2<br>E : 0 | ---, ---, ---, ---, --- | Front က 2 ဆိုတော့ 2 အခန်း G ကိုထုတ်။<br>Front ၁ တိုး၊ E တစ်လျှော့။<br><b>Return : G</b> |

Number of Elements (E) က 0 နဲ့ ညီသွားပြီဖြစ်လို့ ထပ်ထုတ်စရာ မရှိတော့ပါဘူး။ ထပ် ထုတ်ရင် မရှိတာကိုထုတ်တဲ့ “UNDERFLOW” ဖြစ်မှာ ဖြစ်ပါတယ်။

A, B, C, D, E, F, G ဆိုပြီး ထည့်ခဲ့တာ Delete မှာ Retrun ပြန်ထားတာလေးတွေကို အစဉ်လိုက် ကြည့်ကြည့်ပါ။ A, B, C, D, E, F, G ဆိုပြီး အစဉ်လိုက်ပြန်ရပါတယ်။ အရင်ထည့်တဲ့ A အရင်ပြန်ရတဲ့ သဘောပါ။ ဒုတိယ ထည့်တဲ့ B ဒုတိယ ဆိုပြီး FIFO ပုံစံနဲ့ ပြန်ရတာ ဖြစ်ပါတယ်။

Algorithm တွေ ရေးကြည့်မယ်ဆိုရင်

$Front = 0, Rear = 0,$

$E$  is Number of Elements in Queue,

$N$  is length of Queue

`isEmpty()`

`Return E == 0`

`isFull()`

`return E == N`

---

*Insert (Item)*

---

*If  $isFull()$  then:*

*Return false*

*Queue[ $Rear$ ] = Item*

*If  $Rear == N - 1$  then:*

*$Rear = 0$*

*Else:*

*$Rear = Rear + 1$*

*$E = E + 1$*

---

---

*Delete ()*

---

*If  $isEmpty()$  then:*

*Write "Underflow"*

*Return NULL*

*Value = Queue[ $Front$ ]*

*If  $Front == N - 1$  then:*

*$Front = 0$*

*Else:*

*$Front = Front + 1$*

*$E = E - 1$*

*Return Value*

---

အနည်းငယ် ထပ်ရှင်းချင်တာက insert Algorithm မှာ

*If  $isFull()$  then:*

*Return false*

ဆိတဲ့ နေရာပါ။အခန်းပြည့်သလား စစ်တယ်။ ပြည့်ရင် return false ဆိတဲ့ အတွက် ယခုလုပ်နေတဲ့ function ထဲကနေ function call ခေါ်တဲ့ဆီကို ထွက်သွားမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် IF မှာ စစ်ထားတဲ့ isFull မှန်ခဲ့ရင် Queue[Rear] = Item ကနေစပီး သူရဲ့အောက်က စာကြောင်းတွေကို ရောက်လာစရာ အကြောင်းမရှိပါဘူး။ ရောက်လာတယ်ဆိုရင် if နောက်က condition ဖြစ်တဲ့ isFull မှားလို့ ရောက်လာတာဖြစ်ပါတယ်။ ဒါကြောင့် Queue[Rear] = Item အပေါ်မှာ Else: ကို ဖျောက်ထားခဲ့တာ ဖြစ်ပါတယ်။ Else ထည့်ရေးလည်း အတူတူပဲဖြစ်ပါတယ်။ မြင်သာအောင် ပြန်ရေးပြရရင်

***Insert (Item)***

*If isFull() then:*

*Return false*

*Else:*

*Queue[Rear] = Item*

*If Rear == N - 1 then:*

*Rear = 0*

*Else:*

*Rear = Rear + 1*

*E = E + 1*

ဒီပုံစံ ဖြစ်ပါတယ်။ အခု အသစ်ရေးပြတာနဲ့ အပေါ်က ရေးပြခဲ့တဲ့ insert ဟာ အတူတူပဲဖြစ်ပါတယ်။ Else လေးဖျောက်တာနဲ့ မဖျောက်တာပဲကွာပါတယ်။ Delete မှာ လည်း ထိနည်းတူ ပြန်ရေးလို့ ရပါတယ်။



## လေ့ကျင့်ကြည့်ကြပါ။

အပေါ်မှာ တွက်ပြတဲ့ ရုံးရိုးနည်းနဲ့ တစ်ခေါက်၊ algorithm ကို trace လိုက်တာ တစ်ခေါက်နဲ့ တစ်ပုဒ်ကို 2 ခါတွက်ပေးပါ။

1. Queue : ---, ---, 30, 40, ---, -----

1. Insert 50

2. Insert 60

3. Insert 70

4. Insert 80

5. Empty ဖြစ်သည့်တိုင်အောင် delete လုပ်ပါ။

6. Delete လုပ်လိုက်လို့ Return ပြန်တာတွေကို တန်းစီရေးပြပါ။

2. Queue is Empty. N = 10

1. Insert 10, 20, 30, 40, 50, 60, 70

2. Delete five values

3. Queue ပြည့်သည်အထိ odd numbers များကို 1 ကစပြီး Insert လုပ်ပါ။

4. Empty မဖြစ်မချင်းပြန်ထုတ်ပါ။

5. Delete လုပ်လိုက်လို့ Return ပြန်တာတွေကို တန်းစီရေးပြပါ။

## Hashing

အခန်းပေါင်း 10000 ရှိတဲ့ array ထဲကို တွက်လို့ရလာတဲ့ value တွေ ထည့်တယ်ဆိုပါတော့။

ပထမဆုံး တွက်လို့ရလာတဲ့ တန်ဖိုးကို 0 ခန်း၊ ဒုတိယ တွက်လို့ရလာတဲ့ တန်ဖိုးကို 1 အခန်း စသည်ဖြင့်

array ထဲ သိမ်းသွားတယ်။ တွက်လို့ရလာတဲ့ တန်ဖိုးဆိုတာ ကြီးစဉ်ပေါ်လိုက်တွေ၊ ငယ်စဉ်ကြီးလိုက်တွေ မဟုတ်ဘူး။ သူတွက်ချင်တာ ထွက်တယ်။ ဒီတော့ အဲဒီ array ထဲ က တစ်ခုခုကို ရှာချင်တယ် ဆိုရင် တန်ဖိုးတွေဟာ sorting စီထားတာမဟုတ်လို့ Binary search သုံးလို့မရဘူး၊ Linear search ကိုပဲ သုံးပြီး 0 ခန်း၊ 1 အခန်းစသည်ဖြင့် အခန်းနံပါတ် ၁ တိုးတိုးပြီး အခန်းအစဉ်လိုက် ရှာသွားမယ်။ တကယ်လို့ ကိုယ်ရှာချင်တဲ့ တန်ဖိုးက နောက်ဆုံးအခန်းနံပါတ် ၁၉၉၉၅ မှာ ရှိတယ်ဆိုရင် အကြိမ် ၁၀၀၀၀ တို့က်စစ်လိုက်ရတာ ဖြစ်လို့ အချိန် အတော်လေးယူသွားမှာ ဖြစ်ပါတယ်။

အဲဒီလို အခန်းနံပါတ် အစဉ်လိုက်ရှာလို့ ကြာတဲ့ပြဿနာကို ဖြေရှင်းဖို့ Hashing ဆိုတာကို အသုံးပြုလာကြပါတယ်။ Hashing ဟာ ပုံမှန်သိမ်းနည်းအတိုင်း အခန်းအစဉ်လိုက် ထည့်သွားတာမျိုး မလုပ်တော့ဘဲ ထည့်ချင်တဲ့တန်ဖိုးပေါ်မှုတည်ပြီး အခန်းနံပါတ် တွက်ပါမယ်။ တွက်လို့ရလာတဲ့ အခန်းထဲကို ထည့်ချင်တဲ့ တန်ဖိုး သွားသိမ်းမှာ ဖြစ်ပါတယ်။ ဒီတော့ ရှာတဲ့ အခါမှာလည်း Linear search လိုမျိုး အခန်းပေါက်စွေး အခန်းနံပါတ်အစဉ်လိုက် ရှာစရာမလိုတော့ဘဲ ရှာချင်တဲ့ တန်ဖိုးပေါ် အခန်းနံပါတ် တွက်ပြီး ရလာတဲ့ အခန်းထဲ သွားရှာရမှာဖြစ်ပါတယ်။

**တကယ်လို့ ဝင်လာတာက စာသားဆိုရင် Number အရင်ပြောင်းရပါမယ်။**

Cat ဆိုရင်

$C = 3$  (A-Z မှာ သုံးနေရာမြောက်မှာရှိလို့ ဖြစ်ပါတယ်။)

$a = 1$  (A-Z မှာ ပထမနေရာမှာရှိလို့ ဖြစ်ပါတယ်။)

$t = 20$

$$\text{Cat} = 3 + 1 + 20 = 24$$

တကယ်လို့ blank ပါခဲ့ရင် blank တန်ဖိုးက ၀ ဖြစ်ပါတယ်။ ဒါကတော့ စာသားကို ဂဏေန်းပြောင်းတဲ့ အလွယ်ဆုံးနည်းဖြစ်ပါတယ်။ တစ်ခြားတွက်နည်းတွေ ရှိပါသေးတယ်။

### **Array Index Calculation in Hashing**

arrayIndex = value % arraySize(N) ဆိုပြီး တွက်ပေးရပါတယ်။

N = 60

 Insert 360

arrayIndex = 360 % 60 = 0, ဒါကြောင့် 360 ကို array ရဲ့ 0 ခန်းမှာ သွားပြီး ထည့်မှာဖြစ်ပါတယ်။

 Insert 70

arrayIndex = 70 % 60 = 10, ဒါကြောင့် 70 ကို array ရဲ့ 10 ခန်းမှာ သွားပြီး ထည့်မှာဖြစ်ပါတယ်။

 Search 365

arrayIndex = 365 % 60 = 5 ဆိုတော့ 5 အခန်းကို တန်းသွားပြီး 365 ရှိမရှိ စစ်မှာဖြစ်ပါတယ်။

### **Collisions**

သို့သော်လည်း Hashing နည်းက collisions ပြဿနာရှိနေပါတယ်။ Collision ကို နမူနာပြရရင်

365 => arrayIndex = 365 % 60 = 5, ဒီတော့ 365 ကို 5 အခန်းမှာ သွားထည့်မှာဖြစ်ပါတယ်။

125 => arrayIndex = 125 % 60 = 5 ထွက်လာတယ်။ သို့သော် 5 အခန်းက 365 ထည့်ပြီးသားဖြစ်လို့ အခန်းမအားတော့ပါဘူး။ အဲဒါ Collision ပါပဲ။

## အဲဒီပြဿနာကို ဖြေရှင်းစွဲ အတွက်

1. **Linear probing** - ထည့်ရမည့်ခန်းက မအားတာဖြစ်ဖြစ်၊ ရှာတဲ့အခန်းမှာ မတွေ့တာဖြစ်ဖြစ် အခြေနေရှိမျိုးမှာ တွက်လိုရလာတဲ့ array index ကို  $1^1$  ပေါင်းတယ်၊ ဒါမှ အဆင်မပြေသေးဘူးဆိုရင်  $2^1$  ပေါင်းတယ်၊ ဒါမှ အဆင်မပြေသေးဘူးဆိုရင်  $3^1$  ပေါင်းတယ်၊ စသည်ဖြင့် အဆင်မပြေသေးသရွှေ့ 4,5,6 .... ၁ထပ် တွေ့နဲ့ ဆက်တွက် သွားမှာဖြစ်ပါတယ်။

 365 =>  $\text{arrayIndex} = 365 \% 60 = 5$ , ဒီတော့ 365 ကို 5 အခန်းမှာ သွားထည့်မှာဖြစ်ပါတယ်။

 125 =>  $\text{arrayIndex} = 125 \% 60 = 5$  ထွက်လာတယ်။ သို့သော် 5 အခန်းက 365 ထည့်ပြီးသား ဖြစ်လို့ အခန်းမအားတော့ပါဘူး။ ဒါဆို  $1^1$  ပေါင်းမယ်  $5+1 = 6$  အခန်းထဲထည့်မယ်။ တကယ်လို့ 6 အခန်းမအားသေးရင်  $2^1$  ပေါင်းမယ်  $5+2 = 7$  အခန်းထဲထည့်မယ်။ 7 ခန်းမအားသေးရင်  $3^1$  ပေါင်းတွက်မယ်၊ စသည်ဖြင့် တွက် သွားမှာ ဖြစ်ပါတယ်။

2. **Quadratic probing** - ထည့်ရမည့်ခန်းက မအားတာဖြစ်ဖြစ်၊ ရှာတဲ့အခန်းမှာ မတွေ့တာဖြစ်ဖြစ် အခြေနေရှိမျိုးမှာ တွက်လိုရလာတဲ့ array index ကို Quadratic ဆိုတဲ့အတိုင်း ၂ ထပ်တွေ ပေါင်းတွက်မှာဖြစ်ပါတယ်။  $1^2$  ပေါင်းတယ်၊ ဒါမှ အဆင်မပြေသေးဘူးဆိုရင်  $2^2$  ပေါင်းတယ်၊ ဒါမှ အဆင်မပြေသေးဘူးဆိုရင်  $3^2$  ပေါင်းတယ်၊ စသည်ဖြင့် အဆင်မပြေသေးသရွှေ့ 4,5,6 .... ၂ထပ်တွေနဲ့ ဆက်တွက်သွားမှာဖြစ်ပါတယ်။ Linear Probing မှာ ထည့်တာ နမူနာပြုပြီးပြီဖြစ်တာကြောင့် Quadratic probing မှာ ရှာတာကို နမူနာပြပါမယ်။

$N = 15$ 

|     |     |
|-----|-----|
| ... |     |
| 7   | 67  |
| 8   | 128 |
| 9   | 24  |
| 10  |     |
| 11  | 41  |
| 12  |     |
| 13  |     |
| 14  |     |
| 15  |     |
| 16  | 307 |
| ... |     |

📍 **Searching 307**

$$\text{arrayIndex} = 307 \% 15 = 7$$

7 အခန်းမှာ မတွေ့တော့  $1^2$  ပေါင်းမယ်။

$7 + 1^2 = 8$  မှာလည်း မတွေ့ဘူး။

$2^2$  ပေါင်းမယ်။

$7 + 2^2 = 11$  မှာလည်း မတွေ့ဘူး။

$3^2$  ပေါင်းမယ်။

$7 + 3^2 = 16$  မှာ တွေ့ပါတယ်။

### 3. Double Hashing

$$\text{arrayIndex} = \text{value \% N}$$

$$\text{StepSize} = \text{constant} - (\text{value \% constant})$$

$$\text{arrayIndex} = (\text{arrayIndex} + \text{StepSize}) \% N$$

 $N = 10$ 

📍 **Insert 12**

$$\text{newArrayIndex} = \text{value \% N} = 12 \% 10 = 2, 2 အခန်းမှာ 12 ထည့်မယ်။$$

 Insert 2

$\text{arrayIndex} = \text{value \% N} = 2 \% 10 = 2$ , 2 အခန်း မလွတ်တော့လို့ StepSize တွက်မယ်။ StepSize တွက်ဖို့ constant က ကြိုက်တာထားလို့ရပါတယ်၊ 5 ထားလိုက်ပါမယ်။

$$\text{StepSize} = 5 - (2 \% 5) = 5 - 2 = 3$$

$\text{arrayIndex} = (2 + 3) \% 10 = 5$ , 2 ကို 5 အခန်းထဲ ထည့်မှာဖြစ်ပါတယ်။

 Insert 102

$\text{arrayIndex} = \text{value \% N} = 102 \% 10 = 2$ , 2 အခန်း မလွတ်တော့လို့ StepSize တွက်မယ်။ StepSize တွက်ဖို့ constant က ကြိုက်တာထားလို့ရပါတယ်၊ 5 ထားလိုက်ပါမယ်။

$$\text{StepSize} = 5 - (102 \% 5) = 5 - 2 = 3$$

$\text{arrayIndex} = (2 + 3) \% 10 = 5$ , 5 အခန်းက ခုဏာက 2 ထည့်ထားတော့ မအားတော့ဘူး၊ ဒီတော့ ထပ်တွက်မယ်။

$\text{arrayIndex} = (5 + 3) \% 10 = 8$ , 8 အခန်းထဲ 102 ကို ထည့်မှာဖြစ်ပါတယ်။

Hashing အကြောင်းပြောရမယ်ဆိုရင် ဒီထက်မက အများကြီး ပြောစရာကျန်ပါသေးတယ်။ သို့သော် Hashign ဆိုတာ ရှာတဲ့အချိန်(ဆဲထဲတဲ့ အချိန်) အချိန်ကုန် သက်သာအောင် သိမ်းကတည်းက အခန်းနံပါတ် တွက်ပြီး သိမ်းပါတယ်။ ရှာတော့လဲ အခန်းနံပါတ်တွက်ပြီးရှာပါတယ်။

အခန်းအရေတွက် များတာအတွက်ဆိုရင် hashing ကို သုံးသင့်ပါတယ်။ Hash ဆိုတာ သိရအောင် တွက်နည်း အခြေခံလေးပဲ ပြောသွားတာဖြစ်ပါတယ်။ တစ်ခြားတွက်နည်းတွေလည်း ရှိပါသေးတယ်။



### လေ့ကျင့်ကြည့်ကြပါ။

1.  $N = 15$  ဖြစ်သော Array ထဲကို 2, 32, 92, 17, 18, 33, 48, 153, 150 တို့ကို linear probing သုံးပြီး ထည့်ပါ။
2. ထည့်ပြီးသား array ထဲမှ 153 ကို linear probingဖြင့် ဖြန်ရှာပါ။ ဘယ်နှစ်ခါ တိုက်စစ်လိုက်ရသလဲ ဖြေပါ။
3.  $N = 30$  ဖြစ်သော Array ထဲကို 13, 2, 32, 92, 17, 18, 33, 63, 153, 150 တို့ကို Quadratic probing သုံးပြီး ထည့်ပါ။
4. ထည့်ပြီးသား array ထဲမှ 153 ကို Quadratic probing ဖြင့် ဖြန်ရှာပါ။ ဘယ်နှစ်ခါ တိုက်စစ်လိုက်ရသလဲ ဖြေပါ။
5.  $N = 23$ , constant 5, 1, 38, 37, 16, 20, 3, 11, 24, 5, 16, 10, 31, 18, 12, 30, 1, 19, 36, 41, 15, 25 တို့ကို double hashing ဖြင့် ထည့်ပါ။
6. ထည့်ပြီးသား array ထဲမှ 25 ကို double hashing ဖြင့် ဖြန်ရှာပါ။ ဘယ်နှစ်ခါ တိုက်စစ်လိုက်ရသလဲ ဖြေပါ။
7. ဒီမှာ stack တို့ Queue တို့ အပြင် sorted array ဆိုတာ ရှိပါသေးတယ်။ ဥ ပမာ  $N=10$  ဆို 10 ခန်းမပြည့်မချင်း ထည့်လိုက်ရမယ်။ ပထမ 20 ဝင်လာတယ်ဆိုပါစို့။ ပထမဆုံးဆိုတော့ 0 အခန်းမှာ ထည့်မယ်။ နောက် 10 ဝင်လာတယ်ဆိုရင် 10 ကငယ်တော့ 0 အခန်းဝင်မယ်၊ ဒီတော့ 0 ခန်းက 20 က နောက်အခန်းကို ဈွှေရမယ်။ 15 ဝင်လာတယ်ဆိုရင် 10 ထက်ကြီးတယ်၊ 20 ထက်ငယ်တယ်ဆိုတော့ 10 နဲ့ 20 ကြား ထည့်ရမှာဖြစ်ပါတယ်။ Sorted array လေးအတွက် လိုအပ်တဲ့ algorithms ရေးကြည့်ပြီး လေ့ကျင့်စေချင်ပါတယ်။



## Chapter အနှစ်ချုပ်

- ဒီအခန်းကတော့ Array ကို လိုအပ်ချက်ပေါ်မူတည်ပြီး ပုံစံမျိုးစုံနဲ့ အသုံးပြုလို ရတယ်ဆိုတာကို ပြောချင်တာဖြစ်ပါတယ်။
- LIFO - နောက်ဆုံးဝင်တဲ့ကောင် အရင်ဆုံးထွက်ချင်ရင် Stack ကိုသုံးပါတယ်။ နောက်ဆုံးဝင် အရင်ထွက်ဆိုပြီး တစ်ဖက်ထဲမှာ အလုပ်လုပ်တာဖြစ်လို ထောက်ဖို့ရာ တစ်ခုပဲလိုပါတယ်။ top လို ခေါ်ပါတယ်။
- FIFO - လက်မှတ်တန်းစီသလိုမျိုး အရင်ဝင်တဲ့ကောင် အရင်ထွက် လုပ်ချင်ရင်တော့ Queue ကိုသုံးပါတယ်။ ဝင်မယ်ဆိုရင် နောက်ကနေဝင်၊ အရင်ရလိုထွက်မှာက ရှုံးက ဖြစ်ပါတယ်။ ဒါကြောင့် ထောက်ဖို့နှစ်ခုလိုပါတယ်။ ဝင်မည့် နောက်ကို ထောက်မှာက Rear ဖြစ်ပြီး၊ ထွက်မည့် ရှုံးကိုထောက်တာက Front ဖြစ်ပါတယ်။
- Hashing ကတော့ Array ထဲ value ထည့်တဲ့ နေရာမှာပဲဖြစ်ဖြစ်၊ array ထဲက value ရှာတဲ့ အခါမှာပဲ ဖြစ်ဖြစ် value ပေါ်မှာ အခန်းနံပါတ်တွက်ချက်ပြီး အလုပ်လုပ်တာဖြစ်ပါတယ်။ အခန်းအရေတွက်များတာတွေအတွက်ဆိုရင် Hashing က ရှာရဖွေရတာအတွက် အချိန်ကုန် သက်သာစေသော်လည်း အခန်းအရေတွက်နည်းရင်တော့ array index တွက်ရတာ အချိန်ကုန်တယ် ဆိုပြီး ပြောလိုရပါတယ်။

“ပထမနှစ်တုန်းက code တွေဆိုတာ

ကျောက်သင်ပုန်းပေါ်သင်ရတာ၊

နားလည်ရခက်တော့ နားလည်တာလိုလို၊ မလည်တာလိုလိုနဲ့။

ဒါနဲ့ သချို့ဘွာက်သလိုကို ခဏ ခဏ ပြန်တွက်၊ ပြန်ပြန် trace လိုက်တာ

ကြောတော့ အလွတ်တွေကို ရလို့။

ဒုတိယနှစ်ရောက်တော့ ကိုယ့်ဟာကိုယ် ရေးတတ်လာပြီ။

အဲဒါ ပထမနှစ်က အပြန်ပြန် အလှန်လှန် လုပ်ခဲ့တဲ့ အကျိုးတွေပါ။”

## အခန်း (၁၄)

### Sorting Algorithms

#### Swap

Sorting algorithm တွေအကြောင်း မပြောခင်မှာ sorting စီတဲ့နေရာမှာ သိကိုသိဖို့လိုအပ်တဲ့ swap လုပ်တာလေးကို အရင်ပြောပါမယ်။



အဲဒီ ပန်းကန်ပြား နှစ်ခုထဲမှာ ရှိတဲ့ အစားအသောက်တွေကို နေရာလဲချင်ပါတယ်။ ဆိုလိုတာက ဒီဘက်ပန်းကန်ထဲက အသီးတွေကို ဟိုဖက်ပန်းကန်ထဲ ထည့်ပြီး၊ ဟိုဘက်ပန်းကန်ထဲက အစားအသောက်တွေကို အသီးပန်းကန်ထဲ လာထည့်ချင်တာဖြစ်ပါတယ်။ အဲတော့ ဘယ်လိုလုပ်ကြမလဲ။ ပန်းကန်တစ်ချပ် အပိုယူရပါမယ်။



- ယူလိုက်တဲ့ ပန်းကန်အလွတ်ထဲကို အသီးတွေထည့်လိုက်ပါမယ်၊ (တကယ်က ကြိုက်တာအရင်ထည့်လိုရပါတယ်။)
- ဒါဆို အသီးပန်းကန်လွတ်သွားပြီဆိုတော့ အသီးပန်းကန်ထဲကို အာလူးကြွှုစတဲ့အစားအစာတွေ လာထည့်ပါမယ်။

- အာလူးကြော်စတဲ့အစားအစာပန်းကန်က အခုလွတ်သွားပြီ ဆိုတော့မှ ပန်းကန်အလွတ်ထဲ ထည့်ထားတဲ့ အသီးတွေ လာထည့်မှာ ဖြစ်ပါတယ်။

ဘယ်နှစ်ခါ လုပ်လိုက်ရသလဲဆိုရင် ၃ ခါ လုပ်လိုက်ရပါတယ်။ အဲဒီလိုပါပဲ LA<sub>i</sub>  $\leftarrow$  j အခန်းနဲ့ k အခန်း ထဲက တန်ဖိုးတွေကို swap လုပ်ချင်တယ်ဆိုရင်

### Swap (j, k)

- Temp = LA [ j ]
- LA [ j ] = LA [ k ]
- LA [ k ] = Temp

ဒီမှာ Temp ဆိုတာက အပေါ်မှာပြောခဲ့တဲ့ ပန်းကန်လွတ်နဲ့ဆင်တူပါတယ်။ သူ့ထဲကို LA [ j ] ကို ထည့်လိုက်တော့ j အခန်းလွတ်သွားပါတယ်။ ဒီတော့မှ j အခန်းထဲကို k အခန်းထဲက တန်ဖိုး ထည့်ပါတယ်။ နောက်ဆုံးမှာတော့ လွတ်သွားတဲ့ k အခန်းထဲကို Temp ထဲကဟာ ပြန်ထည့်လိုက်တာကြောင့် Temp ထဲ ယာယီထည့်ထားတဲ့ j အခန်း တန်ဖိုးဟာ k အခန်းထဲ ရောက်သွားတာ ဖြစ်ပါတယ်။

|    |    |    |    |    |
|----|----|----|----|----|
| 10 | 20 | 30 | 40 | 50 |
| 0  | 1  | 2  | 3  | 4  |

Let j = 1, k=3

Line 1: Temp = LA [ j ] = LA [ 1 ] = 20

Line 2: LA [ j ] = LA [ k ]

LA [ 1 ] = LA [3]

LA[1] = 40

Line 3: LA [ k ] = Temp

LA [3] = 20

|    |    |    |    |    |
|----|----|----|----|----|
| 10 | 40 | 30 | 20 | 50 |
| 0  | 1  | 2  | 3  | 4  |

## Bubble Sort

Sorting တွေမှာ ငယ်စဉ်ကြီးလိုက်စီချင်ရင် ရွှေအခန်းက ကြိုးနေရင် လဲမယ်(swap မယ်)။ ကြိုးစဉ်ငယ်လိုက် စီချင်ရင် ရွှေအခန်းက ငယ်နေရင် swap လုပ်မှာဖြစ်ပါတယ်။

Bubble sort ရဲ့ အလုပ်လုပ်ပုံက ကပ်ရက်နှစ်ခန်းကို တိုက်စစ်မှာဖြစ်ပါတယ်။ ပထမဆုံး 0 အခန်း နှင့် 1 အခန်းကို တိုက်စစ်မှာ ဖြစ်ပါတယ်။ 0 ဆို 1, 1 ဆို 2, in ဆိုရင် in + 1 အခန်းနဲ့ တိုက်စစ်တာဖြစ်လို့ နောက်ဆုံးအခန်းထိသွားလို့ မရပါဘူး။ နောက်ဆုံးအခန်းထိ သွားခဲ့ ရင် သူရဲ့နောက်မှာ တိုက်စရာ မရှိတော့လို့ ဖြစ်ပါတယ်။ ဒါကြောင့် ပထမအကြိမ်မှာ တစ်ခန်း လျော့တိုက်စစ်မှာ ဖြစ်ပါတယ်။

44, 33, 11, 55, 77, 90, 10 ကို ငယ်စဉ်ကြီးလိုက် စီကြည့်လိုက်ရအောင်ပါ။

### No. 1 - နောက်အခန်းနဲ့ တိုက်မှာမို့ တစ်ခန်းလျော့တိုက်မယ်။

|                            |                                                                              |
|----------------------------|------------------------------------------------------------------------------|
| 44, 33, 11, 55, 77, 90, 10 | 0 အခန်း နဲ့ 1 အခန်းတိုက်စစ်တယ်။ ရွှေအခန်းက ကြိုးတော့ swap လုပ်မယ်။           |
| 33, 44, 11, 55, 77, 90, 10 | 1 အခန်း နဲ့ 2 အခန်းတိုက်စစ်တယ်။ ရွှေအခန်းက ကြိုးတော့ swap လုပ်မယ်။           |
| 33, 11, 44, 55, 77, 90, 10 | 2 အခန်း နဲ့ 3 အခန်းတိုက်စစ်တယ်။ ရွှေအခန်းက ငယ်နေတော့ swap လုပ်စရာ မလိုပါဘူး။ |
| 33, 11, 44, 55, 77, 90, 10 | 3 အခန်း နဲ့ 4 အခန်းတိုက်စစ်တယ်။ ရွှေအခန်းက ငယ်နေတော့ swap လုပ်စရာ မလိုပါဘူး။ |
| 33, 11, 44, 55, 77, 90, 10 | 4 အခန်း နဲ့ 5 အခန်းတိုက်စစ်တယ်။ ရွှေအခန်းက ငယ်နေတော့ swap လုပ်စရာ မလိုပါဘူး။ |

|                            |                                                                                                                                                                                                                                                               |
|----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 33, 11, 44, 55, 77, 90, 10 | <p>5 အခန်း နဲ့ 6 အခန်းတိုက်စစ်တယ်။ ရှုံးအခန်းက ကြီးနေတော့ swap လုပ်မှာ ဖြစ်ပါတယ်။</p> <p>တစ်ခန်းလျော့တိုက်တာကြောင့် 90 အထိပဲ သွားတာဖြစ်ပါတယ်။</p> <p>ဒီတော့မှ သူ့ရဲ့ နောက်တစ်ခန်း 10 နဲ့ တိုက်လို့ရမှာပါ။ 10 ထိသွားခဲ့ရင် နောက်မှာ တိုက်စရာမရှိတော့ပါဘူး။</p> |
|----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

33, 11, 44, 55, 77, 10, 90

ဒါဟာ ပထမအကြိမ် တိုက်စစ်လိုပြီးသွားတာ ဖြစ်ပါတယ်၊ ပထမ အကြိမ်လဲ ပြီးသွားရော အကြီးဆုံးကိန်း တစ်လုံးဖြစ်တဲ့ 90 ဟာ နောက်ဆုံးကို ရောက်သွားပြီ ဖြစ်ပါတယ်။ ဒါကြောင့် ခုတိယ အကြိမ်မှာ နောက်ဆုံး တစ်ခန်းကို ထည့်တိုက်စရာမလိုတော့ပါဘူး။ ဒီတော့ ခုတိယ အကြိမ်မှာ နှစ်ခန်း လျော့ပြီး တိုက်စစ်မှာဖြစ်ပါတယ်။

### No. 2 - အကြီးဆုံး 1 လုံးနောက်ဆုံးရောက်သွားလို့ 2 ခန်းလျော့တိုက်မယ်။

|                            |                                                                               |
|----------------------------|-------------------------------------------------------------------------------|
| 33, 11, 44, 55, 77, 10, 90 | 0 အခန်း နဲ့ 1 အခန်းတိုက်စစ်တယ်။ ရှုံးအခန်းက ကြီးတော့ swap လုပ်မယ်။            |
| 11, 33, 44, 55, 77, 10, 90 | 1 အခန်း နဲ့ 2 အခန်းတိုက်စစ်တယ်။ ရှုံးအခန်းက ငယ်တော့ swap မလုပ်တော့ပါဘူး။      |
| 11, 33, 44, 55, 77, 10, 90 | 2 အခန်း နဲ့ 3 အခန်းတိုက်စစ်တယ်။ ရှုံးအခန်းက ငယ်နေတော့ swap လုပ်စရာ မလိုပါဘူး။ |
| 11, 33, 44, 55, 77, 10, 90 | 3 အခန်း နဲ့ 4 အခန်းတိုက်စစ်တယ်။ ရှုံးအခန်းက ငယ်နေတော့ swap လုပ်စရာ မလိုပါဘူး။ |

|                            |                                                                                                                         |
|----------------------------|-------------------------------------------------------------------------------------------------------------------------|
| 11, 33, 44, 55, 77, 10, 90 | 4 အခန်း နဲ့ 5 အခန်းတိုက်စစ်တယ်။ ရှုံးအခန်းက ကြီးတော့ swap လုပ်မယ်။<br><br>နှစ်ခန်းလျော့တိုက်လို့ 77 ထိ သွားတာဖြစ်ပါတယ်။ |
|----------------------------|-------------------------------------------------------------------------------------------------------------------------|

11, 33, 44, 55, 10, 77, 90

ဒုတိယအကြိမ်တိုက်လဲပြီးသွားရော အကြီးဆုံးကိန်း 2 လုံးနောက်ဆုံးရောက်သွားပြီ ဖြစ်ပါတယ်။ ဒါကြောင့် တတိယ အကြိမ်မှာ 3 ခန်း လျော့တိုက်မှာ ဖြစ်ပါတယ်။

No. 3 - အကြီးဆုံး 2 လုံးနောက်ဆုံးရောက်သွားလို့ 3 ခန်းလျော့တိုက်မယ်။

|                            |                                                                                                                        |
|----------------------------|------------------------------------------------------------------------------------------------------------------------|
| 11, 33, 44, 55, 10, 77, 90 | 0 အခန်း နဲ့ 1 အခန်းတိုက်စစ်တယ်။ ရှုံးအခန်းက ငယ်တော့ swap မလုပ်တော့ပါဘူး။                                               |
| 11, 33, 44, 55, 10, 77, 90 | 1 အခန်း နဲ့ 2 အခန်းတိုက်စစ်တယ်။ ရှုံးအခန်းက ငယ်တော့ swap မလုပ်တော့ပါဘူး။                                               |
| 11, 33, 44, 55, 10, 77, 90 | 2 အခန်း နဲ့ 3 အခန်းတိုက်စစ်တယ်။ ရှုံးအခန်းက ငယ်နေတော့ swap လုပ်စရာ မလိုပါဘူး။                                          |
| 11, 33, 44, 55, 10, 77, 90 | 3 အခန်း နဲ့ 4 အခန်းတိုက်စစ်တယ်။ ရှုံးအခန်းက ကြီးတော့ swap လုပ်မယ်။<br><br>3 ခန်းလျော့တိုက်တာမို့ 55 ထိသွားတာဖြစ်ပါတယ်။ |

11, 33, 44, 10, 55, 77, 90

တတိယအကြိမ်တိုက်လဲပြီးသွားရော အကြီးဆုံးကိန်း 3 လုံးနောက်ဆုံးရောက်သွားပြီ ဖြစ်ပါတယ်။ ဒါကြောင့် နောက်အကြိမ်မှာ 4 ခန်း လျော့တိုက်မှာ ဖြစ်ပါတယ်။

**No. 4 - အကြီးဆုံး 3 လုံးနောက်ဆုံးရောက်သွားလို့ 4 ခန်းလျှော့တိုက်မယ်။**

|                            |                                                                           |
|----------------------------|---------------------------------------------------------------------------|
| 11, 33, 44, 10, 55, 77, 90 | 0 အခန်း နဲ့ 1 အခန်းတို့က်စစ်တယ်။ ရှုံးအခန်းက ငယ်တော့ swap မလုပ်တော့ပါဘူး။ |
| 11, 33, 44, 10, 55, 77, 90 | 1 အခန်း နဲ့ 2 အခန်းတို့က်စစ်တယ်။ ရှုံးအခန်းက ငယ်တော့ swap မလုပ်တော့ပါဘူး။ |
| 11, 33, 44, 10, 55, 77, 90 | 2 အခန်း နဲ့ 3 အခန်းတို့က်စစ်တယ်။ ရှုံးအခန်းက ကြီးနေတော့ swap လုပ်မယ်။     |

11, 33, 10, 44, 55, 40, 90

တတိယအကြိမ်တိုက်လဲပြီးသွားရော အကြီးဆုံးကိန်း 4 လုံးနောက်ဆုံးရောက်သွားပြီ ဖြစ်ပါတယ်။ ဒါကြောင့် နောက်အကြိမ်မှာ 5 ခန်း လျှော့တိုက်မှာ ဖြစ်ပါတယ်။

**No. 5 - အကြီးဆုံး 4 လုံးနောက်ဆုံးရောက်သွားလို့ 5 ခန်းလျှော့တိုက်မယ်။**

|                            |                                                                           |
|----------------------------|---------------------------------------------------------------------------|
| 11, 33, 10, 44, 55, 40, 90 | 0 အခန်း နဲ့ 1 အခန်းတို့က်စစ်တယ်။ ရှုံးအခန်းက ငယ်တော့ swap မလုပ်တော့ပါဘူး။ |
| 11, 33, 10, 44, 55, 40, 90 | 1 အခန်း နဲ့ 2 အခန်းတို့က်စစ်တယ်။ ရှုံးအခန်းက ကြီးတော့ swap လုပ်ပါမယ်။     |

11, 10, 33, 44, 55, 40, 90

ငါးကြိမ်တိုက်လဲပြီးသွားရော အကြီးဆုံးကိန်း 5 လုံးနောက်ဆုံးရောက်သွားပြီ ဖြစ်ပါတယ်။ ဒါကြောင့် နောက်အကြိမ်မှာ 6 ခန်း လျှော့တိုက်မှာ ဖြစ်ပါတယ်။

**No. 6 - အကြီးဆုံး 5 လုံးနောက်ဆုံးရောက်သွားလို့ 6 ခန်းလျှော့တိုက်မယ်။**

|                                   |                                                                       |
|-----------------------------------|-----------------------------------------------------------------------|
| <b>11, 10, 33, 44, 55, 40, 90</b> | 0 အခန်း နဲ့ 1 အခန်းတို့က်စစ်တယ်။ ရှုံးအခန်းက ကြီးတော့ swap လုပ်ပါမယ်။ |
|-----------------------------------|-----------------------------------------------------------------------|

10, 11, 33, 44, 55, 40, 90

အကြီးဆုံးကိန်း ၆ လုံးနောက်ဆုံးရောက်သွားပြီ ဖြစ်ပါတယ်။ ရှုံးမှာ တစ်လုံးပဲ ကျွန်တော့တာကြောင့် ဆက်ပြီး တိုက်စစ်စရာမလိုတော့ပါဘူး။

**Bubble Sort Algorithm**

အပေါ်က တွက်ပြထားတာကို ပြန်ကြည့်ကြည့်ပါမယ်။ 44, 33, 11, 55, 77, 90, 10 ဆိုပြီး ကိန်း 7 လုံးကို ငယ်စဉ်ကြီးလိုက် စီပြထားတာဖြစ်ပါတယ်။ ( $N = 7$ )

- No တပ်ထားတာလေးတွေပြန်ကြည့်လိုက်တော့ 6 ကြိမ်တိတိ table ဆွဲပြထားတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။ 7 လုံးကို 6 ကြိမ်ဆိုတော့, N လုံးဆိုရင် N - 1 ကြိမ် တိုက်စစ်မှာဖြစ်ပါတယ်။
- No တပ်ထားတာ တစ်ကြိမ်မှာ ယေားထဲ အများကြီး တိုက်စစ်ရပါသေးတယ်။ တစ်ကြိမ်မှာ အများကြီး ဆိုရင် nested loop လို့ ပြောခဲ့ပါတယ်။ ဒီတော့ တစ်ကြိမ်လုပ်တဲ့ No တပ်ပြထားတာက outer loop ဖြစ်ပြီး၊ ယေားထဲမှာ ကပ်ရက်နှစ်ခန်း အကြိမ်ကြိမ်တိုက်တာက inner loop ဖြစ်မှာ ဖြစ်ပါတယ်။ No တပ်ပြထားတဲ့ outer loop အတွက် variable out လို့ထားပြီး၊ ကပ်ရက်အခန်းတိုက်တဲ့ inner loop အတွက် variable in လို့ ထားလိုက်ပါမယ်။
- ပြီးတော့ No 1 မှာ တစ်ခန်းလျှော့တိုက်တယ်၊ No 2 အကြိမ်မှာ နှစ်ခန်းလျှော့တိုက်တာဖြစ်လို့ No ကို out ထားထားတော့ out အကြိမ် လျှော့တိုက်မှာဖြစ်ပါတယ်။

---

bubbleSort(LA, N)

---

1. Set UB = N - 1
  2. Repeat for **out = 1** to **UB** by **1**:
  3.     Repeat for **in =0** to **UB - out** by **1**:
  4.         If **LA[in] > LA[in+1]** then:
  5.             Swap ( **in, in+1**)
  6.     End of Loop
  7. End of loop
- 

- ရှိတဲ့အခန်းအရေအတွက် N က 5 ဆိုရင် အခန်းနံပါတ်က 0 to 4 ဖြစ်မှာ ဖြစ်လို့ UB က 4 ဖြစ်မှာ ဖြစ်ပါတယ်။ ဒါကြောင့် line 1 မှာ UB = N - 1 ဆိုပြီး တွက်ထားတာပါ။
- အပေါ်မှာ ကိန်း 7 လုံးစီတော့ No 0 တပ်ပြထားတာကြည့်ပါ။ 1 to 6 လုပ်ခဲ့ရတာဖြစ်လို့ Line 2 က out ဟာ 1 to UB လုပ်ထားတာဖြစ်ပါတယ်။
- ကပ်ရက် နှစ်ခန်းတိုက်စစ်တဲ့ inner loop ဟာ 0 ခန်းက စတယ်၊ No 1 မှာ ရှိတဲ့ အခန်းနံပါတ်ထဲက 1 ခန်းလျော့၊ No 2 မှာ ရှိတဲ့ အခန်းနံပါတ်ထဲက 2 ခန်းလျော့ စသည်ဖြင့်သွားလို့ No ကို ကိုယ်စားပြုတာဟာ out ဖြစ်တဲ့အတွက် UB-out ထိ သွားထားတာဖြစ်ပါတယ်။

စပြီး လေ့လာတဲ့အချိန်မှာ algorithm ကို ကောင်းကောင်း နာလည်သဘောပေါက် စေတဲ့ နည်းကတော့ Trace လိုက်တာပဲ ဖြစ်ပါတယ်။ ဒီတော့ trace လိုက်ကြည့်ကြရအောင်။

LA = { 44, 11, 22, 33}, N = 4

UB = N – 1 = 4 – 1 = 3

-  Out ဟာ 1 ကနေ UB အထိ သွားမှာဖြစ်လို့ table ရဲ့ out column မှာ 1,2,3 ဆိုပြီးဖြည့်လိုက်တယ်။

| out | In | အလုပ်လုပ်ပုံ | LA |
|-----|----|--------------|----|
| 1   |    |              |    |
| 2   |    |              |    |
| 3   |    |              |    |

-  Out တန်ဖိုး 1 မှာ in က 0 to UB - out =  $3 - 1 = 2$  ထိသွားမှာဖြစ်လို့ out 1 အတွက် in column ကို 0,1,2 လို့ဖြည့်လိုက်တယ်။

| out | In | အလုပ်လုပ်ပုံ | LA             |
|-----|----|--------------|----------------|
| 1   | 0  |              | 44, 11, 22, 33 |
|     | 1  |              |                |
|     | 2  |              |                |

-  Out တန်ဖိုး 2 ဆိုရင် in က 0 to UB - out =  $3 - 2 = 1$  (0,1)

Out တန်ဖိုး 3 ဆိုရင် in က 0 to UB - out =  $3 - 3 = 0$  (0)

| out | In | အလုပ်လုပ်ပုံ | LA             |
|-----|----|--------------|----------------|
| 1   | 0  |              | 44, 11, 22, 33 |
|     | 1  |              |                |
|     | 2  |              |                |

|          |   |  |  |
|----------|---|--|--|
| <b>2</b> | 0 |  |  |
|          | 1 |  |  |
| <b>3</b> | 0 |  |  |

အခုခိုရင် outer loop နဲ့ inner loop ကို table ထဲ ဖြည့်ပြီဖြစ်လို့ တစ်ကြိမ်ချင်းစီမှာ

4. If  $LA[in] > LA[in+1]$  then:

5.              Swap ( in, in+1)

ကို လုပ်ရတော့မှာဖြစ်ပါတယ်။

| out      | In | LA[in] > LA[in+1]                                                                                                                                          | LA                                                                       |
|----------|----|------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------|
| <b>1</b> | 0  | LA [0] > LA[1] $\rightarrow$ 44 > 11 ဆိုတော့မှန်တော့<br>Swap(in, in+1) $\rightarrow$ Swap(0,1) ကိုလုပ်မယ်။                                                 | 44, 11, 22, 33                                                           |
|          | 1  | LA [1] > LA[2] $\rightarrow$ 44 > 22 ဆိုတော့မှန်တော့ <span style="color:red">→ 11, 44, 22, 33</span><br>Swap(in, in+1) $\rightarrow$ Swap(1,2) ကိုလုပ်မယ်။ |                                                                          |
|          | 2  | LA [2] > LA[3] $\rightarrow$ 44 > 33 ဆိုတော့မှန်တော့<br>Swap(in, in+1) $\rightarrow$ Swap(2,3) ကိုလုပ်မယ်။                                                 | 11, 22, 44, 33                                                           |
| <b>2</b> | 0  | LA [0] > LA[1] $\rightarrow$ 11 > 22 ဆိုတော့မှားတယ်၊ ဒီတော့<br>ဘာမှ လုပ်စရာမလိုတော့ဘူး။                                                                    | <span style="border: 2px solid red; padding: 2px;">11, 22, 33, 44</span> |
|          | 1  | LA [1] > LA[2] $\rightarrow$ 22 > 33 ဆိုတော့မှားတယ်၊ ဒီတော့<br>ဘာမှ လုပ်စရာမလိုတော့ဘူး။                                                                    | 11, 22, 33, <span style="color:red">44</span>                            |
| <b>3</b> | 0  | LA [0] > LA[1] $\rightarrow$ 11 > 22 ဆိုတော့မှားတယ်၊ ဒီတော့<br>ဘာမှ လုပ်စရာမလိုတော့ဘူး။                                                                    | 11, 22, <span style="color:red">33, 44</span>                            |

11, 22, 33, 44 ဆိုပြီး အဖြေထွက်သွားပါတယ်။ ဒီမှာ တစ်ချက်သတိထားစေခဲင်တာ out တန်ဖိုး

1 ပြီးလို့ ထွက်လာတဲ့ အဖြေကို တစ်ချက်ကြည့်ကြည့်ပါ။ table မှာ မြင်သာအောင် နှိမ်ပြုရောင် လေးထောင့်လေးနဲ့ ပိုင်းပြထားပါတယ်။ out တန်ဖိုး 1 ပြီးရုံနဲ့ အားလုံး sorting စီပြီးသား ဖြစ်သွားတာကို တွေ့ရမှာဖြစ်ပါတယ်။ ဒီလို့ sorting စီပြီးသား ဖြစ်မဖြစ်ဆိုတာကတော့ ဝင်လာတဲ့ ကိန်းတွေပေါ်မှာမူတည်တာပေါ့လေ။ sorting စီပြီးသား ဖြစ်တာကို ထပ် စစ်နေတော့ အချိန်ကုန်တာပဲ ရှိမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် စီပြီးသား ဖြစ်သွားရင်၊ တနည်းအားဖြင့် out တစ်ကြိမ်စာအတွက် inner loop တိုက်စစ်တာ မှာ တစ်ကြိမ်မှ မလဲရတော့ရင် (swap) မလုပ်ရတော့ရင် ဆက်မတိုက်နဲ့ ဆိုတာမျိုး ရေးလို့ရပါတယ်။

---

### bubbleSort(LA, N)

---

1. Set UB = N – 1
  2. Set out = 1
  3. Set isSwap = true
  4. Repeat while **out<= UB and isSwap:**
  5.     isSwap = false
  6.     Repeat for **in =0 to UB - out by 1:**
  7.         If LA[in] > LA[in+1] then:
  8.             Swap ( in, in+1)
  9.             isSwap = true
  10.    End of Loop
  11.    out = out + 1
  12. End of loop
-

outer loop ရဲ့ while မှာ and isSwap ဆိတာ isSwap ကလည်း true ဖြစ်နေမှု ဆက်လုပ်မယ်လို့ ဆိုလိုတာ ဖြစ်ပါတယ်။ ဒီမှာ out (outer) loop တစ်ကြိမ်စတာနဲ့ isSwap ထဲကို false ထည့်ထားလိုက်တယ်။ မလဲရသေးဘူးလို့ ဆိုလိုတာဖြစ်ပါတယ်။ အထဲက in(inner) loop မှာ လဲလိုက်တာနဲ့ isSwap ထဲကို true ကောက်ထည့်လိုက်တယ်။ ဒီတော့ inner ထဲမှာ တစ်ကြိမ်မှ မလဲခဲ့ရဘူးဆိုရင် isSwap ဟာ false အတိုင်းပဲရှိနေမှာ ဖြစ်တယ်။ ဒါဆို စောင့်မှု မှာ isSwap(true) ဖြစ်မှု လုပ်မယ်ပြောထားလို့ out ကနေထွက်သွားမှာဖြစ်လို့ ဆက်ပြီးပတ်စကားမလိုတော့ဘူး။ လဲခဲ့ရင် isSwap ဟာ true ဖြစ်နေမယ်။ ဒါဆို ဆက်ပြီး looping ပတ်ဖို့လိုတယ်။



### လေ့ကျင့်ကြည့်ကြပါ။

- LA = 44, 33, 11, 55, 77, 90, 40, 60, 99, 22, 88, 66 ကို Bubble sort သော algorithm ဖြင့် sorting စီပါ။
- LA = 44, 33, 11, 55, 77, 90, 40, 60, 99, 22, 88, 66 ကို Bubble sort ဒုတိယ algorithm ဖြင့် sorting စီပါ။ Trace လိုက်ရမည့် table ပုံစံ ဆွဲပြထားပါတယ်။

| Line | Out | In | isSwap | အလုပ်လုပ်ပုံ | LA |
|------|-----|----|--------|--------------|----|
|------|-----|----|--------|--------------|----|

- ပထာမ Bubble sort algorithm ကို ကြိုးစဉ်ပေါ်လိုက်စီစွဲ ပြန်ရေးပေးပါ။
- ဒုတိယ Bubble sort algorithm ကို ကြိုးစဉ်ပေါ်လိုက်စီစွဲ ပြန်ရေးပေးပါ။

## Selection Sort

Bubble sort က 0 အခန်းဆိုရင် 1 အခန်းနဲ့ တိုက်စစ်တယ်၊ in ဆိုရင် in+1 နဲ့ တိုက်စစ်တယ်။ ဆိုလိုတာက နောက်တစ်ခန်းနဲ့ တိုက်စစ်တယ်။ ဒါကြောင့် နောက်ဆုံးအခန်းထိသွားလို့မရဘူး။ နောက်ဆုံးရဲ့ ရှေ့တစ်ခန်းထိပဲ သွားလို့ရပါတယ်။

Selection Sorting က ပြောင်းပြန်၊ 2 ဆိုရင် 1, 1 ဆိုရင် 0 အခန်း၊ J ဆိုရင် J-1 ဆိုပြီး ရှေ့အခန်းနဲ့ တိုက်တာဖြစ်ပါတယ်။ ဒါကြောင့် အခန်းနံပါတ် 0 ကနေ စလိုမပါဘူး။ 0 ကသာ စခဲ့မယ်ဆိုရင် သူ့မှာ တိုက်စစ်စရာ အရှေ့ခန်းမရှိတော့လို့ ဖြစ်ပါတယ်။ ဒါကြောင့် 1 ခန်းက စပါမယ်။ အပေါ်မှာက နမူနာ တွက်ပြပြီးမှ algorithm ရှင်းတာ ဆိုတော့၊ ဒီတစ်ခါ algorithm ပြပြီးမှ trace လိုက်ကြရအောင်။

---

`selectionSort(LA, N)`

---

1. Set UB = N -1
  2. Repeat for out =0 to UB-1 by 1
  3.     Set min = out
  4.     Repeat for in = out + 1 to UB by 1
  5.         If(LA[min] > LA[in]) then:
  6.             min = in;
  7.     End of Loop
  8.     Swap (out, min)
  9. End of Loop
- 

44, 33, 11, 55, 77, 90, 10 ကို C ယ်စဉ်ကြီးလိုက် SelectionSort နဲ့ စီကြည့်လိုက်ရအောင်ပါ။

N = 7

| Line | out | in | Min | LA [ min ] > LA [in]                       | LA                            | အလုပ်လုပ်ပုံ                                                                                                                                                                                                                                      |
|------|-----|----|-----|--------------------------------------------|-------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1    |     |    |     |                                            | 44, 33, 11,<br>55, 77, 90, 10 | UB = N - 1 = 7 - 1 = 6                                                                                                                                                                                                                            |
| 2    | 0   |    |     |                                            |                               |                                                                                                                                                                                                                                                   |
| 3    |     |    | 0   |                                            |                               | min = out = 0                                                                                                                                                                                                                                     |
| 4    |     | 1  |     |                                            |                               | in = out + 1 = 0 + 1                                                                                                                                                                                                                              |
| 5    |     |    | 1   | LA[0] > LA[1]<br><br>44 > 33 -> true       |                               | Line 6 : min = in                                                                                                                                                                                                                                 |
| 7    |     |    |     |                                            |                               | Inner loop အပိတ် ဖြစ်လို့ line<br>4ကို ပြန် တက်မယ်။                                                                                                                                                                                               |
| 4    |     | 2  |     |                                            |                               | For looping မှာက အပေါ်<br>ပြန်တက်လာတာနဲ့ update လုပ်<br>ပါတယ်။ ပြီးရင် condition စစ်<br>ပါတယ်။<br><br>By 1 ဆိုတဲ့အတွက် in = in + 1 လုပ်<br>in = 1 + 1 = 2<br><br>to UB လို့ဆိုတဲ့အတွက်<br>in <= UB ထိ လုပ်မှာဖြစ်ပါတယ်။<br>2 <= 6 ဆိုတော့မျန်တယ်။ |
| 5    |     |    | 2   | LA [ 1 ] > LA [ 2 ]<br><br>33 > 11 -> true |                               | Line 6 : min = in                                                                                                                                                                                                                                 |
| 7    |     |    |     |                                            |                               | Go to line 4                                                                                                                                                                                                                                      |

|          |  |   |                                          |  |                                                               |
|----------|--|---|------------------------------------------|--|---------------------------------------------------------------|
| <b>4</b> |  | 3 |                                          |  | Update : in = in +1 = 2+1 = 3<br><br>in <= UB, 3 <= 6 -> ture |
| <b>5</b> |  |   | LA [ 2] > LA [3]<br><br>11 > 55 -> false |  | မှားတော့ line 6 ကို မလုပ်<br>တော့ဘူး                          |
| <b>7</b> |  |   |                                          |  | Go to Line 4                                                  |
| <b>4</b> |  | 4 |                                          |  | Update : in = in +1 = 3+1 = 4<br><br>in <= UB, 4 <= 6 -> ture |
| <b>5</b> |  |   | LA [ 2] > LA [4]<br><br>11 > 77 -> false |  | မှားတော့ line 6 ကို မလုပ်<br>တော့ဘူး                          |
| <b>7</b> |  |   |                                          |  | Go to Line 4                                                  |
| <b>4</b> |  | 5 |                                          |  | Update : in = in +1 = 4+1 = 5<br><br>in <= UB, 5 <= 6 -> ture |
| <b>5</b> |  |   | LA [ 2] > LA [5]<br><br>11 > 90 -> false |  | မှားတော့ line 6 ကို မလုပ်<br>တော့ဘူး                          |
| <b>7</b> |  |   |                                          |  | Go to Line 4                                                  |
| <b>4</b> |  | 6 |                                          |  | Update : in = in +1 = 5+1 = 6<br><br>in <= UB, 6 <= 6 -> ture |
| <b>5</b> |  | 6 | LA [ 2] > LA [4]<br><br>11 > 10 -> true  |  | Line 6: min = in                                              |
| <b>7</b> |  |   |                                          |  | Go to Line 4                                                  |
| <b>4</b> |  | 7 |                                          |  | Update : in = in +1 = 6+1 = 7                                 |

|          |   |   |  |                               |                                                                                                                      |
|----------|---|---|--|-------------------------------|----------------------------------------------------------------------------------------------------------------------|
|          |   |   |  |                               | in <= UB, 6 <= 6 -> false ဆိုတော့<br>inner loop ကုတ် line 8 ကို<br>သွား                                              |
| <b>8</b> |   |   |  | 10, 33, 11,<br>55, 77, 90, 44 | Swap (out, min)<br><br>Swap (0, 6)<br><br>Inner loop ပြီးတာနဲ့ ငယ်ဆုံး<br>တစ်လုံး ရွှေ့ကို ရောက်သွားတာ<br>ဖြစ်ပါတယ်။ |
| <b>9</b> |   |   |  |                               | Outer loop အပိတ်ဖြစ်လို့ Line 2<br>ပြန်တက်။                                                                          |
| <b>2</b> | 1 |   |  |                               | Update : out = out +1 =0+1 =1<br><br>Condition : out <= UB<br><br>1 <= 6 -> true                                     |
| <b>3</b> |   | 1 |  |                               | min = out                                                                                                            |
| <b>4</b> |   | 2 |  |                               | Initialization: in = out + 1<br><br>Condition : in <= UB<br><br>2 <= 6 → true                                        |
| <b>5</b> |   |   |  |                               |                                                                                                                      |

ကဲ့ ဒီလောက်နမူနာ trace လိုက်ပြထားရင် ဆက်လိုက်တတ်မယ်လို့ ထင်ပါတယ်။ Algorithm တစ်ခုကို ကောင်းကောင်းနားလည်ဖို့ ဆိုရင် trace များများလိုက်ရမှာဖြစ်ပါတယ်။ ဒါကြောင့် ဒါလေးကို

အဖြေထွက်သည်အထိ Trace လိုက်ဖြစ်အောင်လိုက်ပါ။ Trace ပြီးအောင် လိုက်ပြီးမှ အောက်က စာတွေကို ဆက်ဖတ်ပါ။

အလုပ်လုပ်ပုံကို အကျဉ်းချုပ်ပြီး ပြောရမယ်ဆိုရင်

- ကိန်းအလုံးအရေအတွက်အတိုင်း အကြိမ်အရေအတွက်လုပ်ရမှာဖြစ်လို့ out သည် 0 to UB ထိ လုပ်တယ်။
- ရှေ့တစ်ခန်းနဲ့ တိုက်မှာဖြစ်လို့ တိုက်စရာရှေ့တစ်ခန်းရှိအောင် in သည် 0 ခန်းထိသွားလို့မရဘူး။ out 0 မှာ in က 1 အခန်းက စမယ်။ 0 (out 0 မှာ 0 ခန်း) ခန်းနဲ့ တိုက်ချင်တာဖြစ်လို့ 1 ကစတာဖြစ်ပါတယ်။

Out 1 မှာ in က 2 က စမယ်။ out 0 ပြီးရင် အငယ်ဆုံးတစ်ခန်း ထွက်သွားပြီဖြစ်ပါတယ်။ ဒါကြောင့် 0 ခန်းနဲ့ တိုက်စရာမလိုတော့ပဲ out 1 မှာ အခန်း နဲ့ တိုက်ချင်လို့ in က 2 က စတာဖြစ်ပါတယ်။

ဒီတော့ in သည် out + 1 ကစပြီး နောက်ဆုံးအခန်းထိသွားမယ်။ ရှေ့အခန်းနဲ့ တိုက်စစ်မယ်။ တန်ည်းအားဖြင့် out အခန်းနဲ့ တိုက်စစ်မယ်။ ဒါကြောင့် out အခန်းကို အငယ်ဆုံးလို့ ယူဆ လိုက်တယ်။ ပြီးရင် ငယ်တဲ့ အခန်း ပြောင်းထည့်မယ်။

အလွယ်နည်းနဲ့ Trace လိုက်ပါမယ်။

44, 33, 11, 55, 77, 90, 10

$N = 7$ ,  $UB = 7 - 1 = 6$

Out : 0 to UB → 0 to 6

**Out 0:** min = 0

in 1: 44, 33, 11, 55, 77, 90, 10, min = 1

in 2: 44, 33, 11, 55, 77, 90, 10, min = 2

in 3: 44, 33, 11, 55, 77, 90, 10, min = 2

in 4: 44, 33, 11, 55, 77, 90, 10, min = 2

in 5: 44, 33, 11, 55, 77, 90, 10, min = 2

in 6: 44, 33, 11, 55, 77, 90, 10, min = 6

အစိမ်းရောင်အဂိုင်းက in ဖြစ်ပြီး၊ အနီးရောင် အဂိုင်းက min ဖြစ်ပါတယ်။ ပြီးတာနဲ့ out အခန်းနှင့် min အခန်း swap လုပ်မယ်။

Swap(0,6) ဆိုတော့ 10, 33, 11, 55, 77, 90, 44 ဆိုပြီးရမယ်။ အငယ်ဆုံး ကိန်းဖြစ်တဲ့ 10 ဟာ ရှေ့ဆုံးရောက်သွားပြီဖြစ်ပါတယ်။ Bubble sort ဟာ တစ်ကြိမ်ပြီးတိုင်း အကြီးဆုံးတစ်လုံး နောက်ရောက်တယ်။ Selection sort ကတော့ တစ်ကြိမ်ပြီးတိုင်း အငယ်ဆုံး တစ်ခန်း ရှေ့ရောက်ပါတယ်။ ဒါကြောင့် တစ်ကြိမ်ပြီးရင် ရှေ့တစ်ခန်း လျှော့လျှော့ပြီး တွက်မှာ ဖြစ်ပါတယ်။

**Out 1:** min = 1

in 2 : 10, 33, 11, 55, 77, 90, 44 , min = 2

in 3 : 10, 33, 11, 55, 77, 90, 44 , min = 2

in 4 : 10, 33, 11, 55, 77, 90, 44 , min = 2

in 5 : 10, 33, 11, 55, 77, 90, 44 , min = 2

in 6 : 10, 33, 11, 55, 77, 90, 44 , min = 2

ပြီးတာနဲ့ out အခန်းနှင့် min အခန်း swap လုပ်မယ်။

Swap(1,2) ဆိုတော့ 10, 11, 33, 55, 77, 90, 44 ဆိုပြီးရမယ်။ ဒုတိယ အင်ယ်ဆုံး ကိန်းဖြစ်တဲ့ 11 ဟာ 1 အခန်းကို ရောက်သွားပြီဖြစ်ပါတယ်။ ဒါကြောင့် နောက်တစ်ခန်းထပ်ပြီး လျှော့တွက်မှာ ဖြစ်ပါတယ်။

**Out 2:** min = 2

in 3: 10, 11, 33, 55, 77, 90, 44, min = 2

in 4: 10, 11, 33, 55, 77, 90, 44, min = 2

in 5: 10, 11, 33, 55, 77, 90, 44, min = 2

in 6: 10, 11, 33, 55, 77, 90, 44, min = 2

ပြီးတာနဲ့ out အခန်း နှင့် min အခန်း swap လုပ်မယ်။ Swap(2,2) ဆိုတော့ 10, 11, 33, 55, 77, 90, 44 ဆိုပြီးရမယ်။

**Out 3:** min = 3

in 4: 10, 11, 33, 55, 77, 90, 44 min = 3

in 5: 10, 11, 33, 55, 77, 90, 44 min = 3

in 6: 10, 11, 33, 55, 77, 90, 44 min = 6

ပြီးတာနဲ့ out အခန်း နှင့် min အခန်း swap လုပ်မယ်။ Swap(3,6) ဆိုတော့ 10, 11, 33, 44, 77, 90, 55 ဆိုပြီးရမယ်။

**Out 4:** min = 4

in 5: 10, 11, 33, 44, 77, 90, 55 min = 4

in 6: 10, 11, 33, 44, 77, 90, 55 min = 6

ပြီးတာနဲ့ out အခန်း နှင့် min အခန်း swap လုပ်မယ်။ Swap(4,6) ဆိုတော့ 10, 11, 33, 44, 55, 90, 77 ဆိုပြီးရမယ်။

**Out 5:** min = 5

in 6: 10, 11, 33, 44, 55, 90, 77 min = 6

ပြီးတာနဲ့ out အခန်း နှင့် min အခန်း swap လုပ်မယ်။ Swap(5,6) ဆိုတော့ 10, 11, 33, 44, 55, 77, 90 ဆိုပြီးရမယ်။ စစ်စိုးတစ်လုံးပဲ ကျဉ်တော့တာကြောင့် sorting စီလိုပြီးသွားပြီ ဖြစ်ပါတယ်။ ဒီလောက်ဆို နားလည်လောက်မယ် ထင်ပါတယ်။

ထပ်ပြီး တိုက်တွန်းပါရစေ။ လေ့ကျင့်ခန်းတွေလုပ်ပါ။ များများ Trace လိုက်ပါ။



လေ့ကျင့်ကြည့်ကြပါ။

1. LA = 44, 33, 11, 55, 77, 90, 40, 60, 99, 22, 88, 66 ကို Selection sort အလွယ်နည်းဖြင့် sorting လုပ်ပါ။
2. Selection sort algorithm ကို ကြိုးစဉ်ယ်လိုက်အတွက် ပြန်ရေးပေးပါ။
3. Question 1 ကို Question တွင်ရေးထားတဲ့ ကြိုးစဉ်ယ်လိုက် algorithm ဖြင့် trace လိုက်ပါ။

## Insertion Sort

insertionSort(LA, N)

1. Set UB = N - 1
2. Repeat for out = 1 to UB by 1
3. Set temp = LA [out]
4. in = out
5. Repeat while  $in > 0$  and  $LA[in-1] \geq temp$  then:
6.      $LA[in] = LA [in - 1]$
7.     in = in - 1
8. End of Loop
9.  $LA [in] = temp$
10. End of loop

Insertion sort က တစ်ခုခုကို အသေထားပြီး ရှုက အခန်းတွေနဲ့ တိုက်တာဖြစ်ပါတယ်။

မြင်သာအောင် Trace လိုက်ကြည့်လိုက်ရအောင်။ အပေါ်မှာ တော်တော်ရှင်းခဲ့ပြီးပြီဆိုတော့ ဒီတစ်ခါ အများကြီး မရှင်းပဲ Trace လိုက်ကြည့်ပါမယ်။

44, 33, 11, 55, 77, 90, 10

N = 7, UB = 7 - 1 = 6

**Out 1:**

**Temp = LA [1] = 33**

in ဟာ out ကနေ 0 ထက်ကြီးတဲ့ အထိ သွားမှာဖြစ်ပါတယ်။

**in 1:** 44, 33, 11, 55, 77, 90, 10

in -1 အခန်းနှင့် temp နဲ့ တိုက်ပါမယ်။ temp ထက်ကြီးခဲ့ရင် in - 1 အခန်းထဲက တန်ဖိုးကို in အခန်းထဲ ထည့်မှာဖြစ်ပါတယ်။

44, 44, 11, 55, 77, 90, 10

in 0: in ကို ၁ လျော့သွားလို ၀ ရောက်ခဲ့ရင် သို့မဟုတ် temp ထက်မကြီးတော့ရင် in အခန်းထဲကို temp ထည့်မယ်။

$LA[0] = temp$

33, 44, 11, 55, 77, 90, 10

## Out 2:

**temp = LA[2] = 11**

in 2: 33, 44, 11, 55, 77, 90, 10

in-1 အခန်းတန်ဖိုးဟာ temp ထက်ကြီးလို in ထဲ in - 1 ကို ထည့်မယ်။

33, 44, 44, 55, 77, 90, 10

in 1: 33, 44, 44, 55, 77, 90, 10

in -1 အခန်းတန်ဖိုးဟာ temp ထက်ကြီးလို in ထဲ in - 1 ကို ထည့်မယ်။

33, 33, 44, 55, 77, 90, 10

in 0:  $a[in] = temp$

11, 33, 44, 55, 77, 90, 10

**Out 3:**

**temp = LA[3] = 55**

in 3: 11, 33, 44, 55, 77, 90, 10

in-1 အခန်းတန်ဖိုးဟာ temp မကြီးတော့လို့ looping ထဲကထွက်မယ်။

LA[in] = temp

LA[ 3] =temp

11, 33, 44, 55, 77, 90, 10

**Out 4:**

**temp = LA[4] = 77**

in 4: 11, 33, 44, 55, 77, 90, 10

in-1 အခန်းတန်ဖိုးဟာ temp မကြီးတော့လို့ looping ထဲကထွက်မယ်။

LA[in] = temp

LA[ 4] =temp

11, 33, 44, 55, 77, 90, 10

**Out 5:****temp = LA[5] = 90**

in 5: 11, 33, 44, 55, 77, 90, 10

in -1 အခန်းတန်ဖိုးဟာ temp မကြိုးတော့လို့ looping ထဲကထွက်မယ်။

LA[in] = temp

LA[ 4] =temp

11, 33, 44, 55, 77, 90, 10

**Out 6:****temp = LA[6] = 10**

in 6: 11, 33, 44, 55, 77, 90, 10

in -1 အခန်းနှင့် temp နဲ့ တိုက်ပါမယ်။ temp ထက်ကြီးခဲ့ရင် in - 1 အခန်းထဲက တန်ဖိုးကို in အခန်းထဲ ထည့်မှာဖြစ်ပါတယ်။

11, 33, 44, 55, 77, 90, 90

in 5: 11, 33, 44, 55, 77, 90, 10

in -1 အခန်းနှင့် temp နဲ့ တိုက်ပါမယ်။ temp ထက်ကြီးခဲ့ရင် in - 1 အခန်းထဲက တန်ဖိုးကို in အခန်းထဲ ထည့်မှာဖြစ်ပါတယ်။

11, 33, 44, 55, 77, 77, 90

in 4: 11, 33, 44, 55, 77, 90, 10

in -1 အခန်းနှင့် temp နဲ့ တိုက်ပါမယ်။ temp ထက်ကြီးခဲ့ရင် in - 1 အခန်းထဲက တန်ဖိုးကို in အခန်းထဲ ထည့်မှာဖြစ်ပါတယ်။

11, 33, 44, 55, 55, 77, 90

in 3: 11, 33, 44, 55, 77, 90, 10

in -1 အခန်းနှင့် temp နဲ့ တိုက်ပါမယ်။ temp ထက်ကြီးခဲ့ရင် in - 1 အခန်းထဲက တန်ဖိုးကို in အခန်းထဲ ထည့်မှာဖြစ်ပါတယ်။

11, 33, 44, 44, 55, 77, 90

in 2: 11, 33, 44, 55, 77, 90, 10

in -1 အခန်းနှင့် temp နဲ့ တိုက်ပါမယ်။ temp ထက်ကြီးခဲ့ရင် in - 1 အခန်းထဲက တန်ဖိုးကို in အခန်းထဲ ထည့်မှာဖြစ်ပါတယ်။

11, 33, 33, 44, 55, 77, 90

in 1: 11, 33, 44, 55, 77, 90, 10

in -1 အခန်းနှင့် temp နဲ့ တိုက်ပါမယ်။ temp ထက်ကြီးခဲ့ရင် in - 1 အခန်းထဲက တန်ဖိုးကို in အခန်းထဲ ထည့်မှာဖြစ်ပါတယ်။

11, 11, 33, 44, 55, 77, 90

in 0: loop ထဲကထဲ LA[in] = temp

LA[0] = temp

10, 11, 33, 44, 55, 77, 90



## လေ့ကျင့်ကြည့်ကြပါ။

1. Sorting Algorithm တွေဖြစ်တဲ့ Bubble Sort, Selection Sort နှင့် Insertion sort ရဲ ခုမှာ  
ဘယ်တစ်ခုဟာ အလုပ်လုပ်ရတာ အသက်သာဆုံးလို့ ထင်ပါသလဲ။ Trace လိုက်ထားတာကို  
ကြည့်ပြီး အကြမ်းယျင်း သိနိုင်ပါတယ်။
2. LA = 44, 33, 11, 55, 77, 90, 40, 60, 99, 22, 88, 66 ကို Insertion sort algorithm ဖြင့် sorting  
စီပါ။
3. Insertion sort algorithm ကို ကြီးစဉ်ပေါ်လိုက် စီရန် ပြင်ရေးပေးပါ။
4. LA = 44, 33, 11, 55, 77, 90, 40, 60, 99, 22, 88, 66 ကို Insertion sort algorithm ဖြင့်  
ကြီးစဉ်ပေါ်လိုက် စီပေးပါ။

## Merge Sort

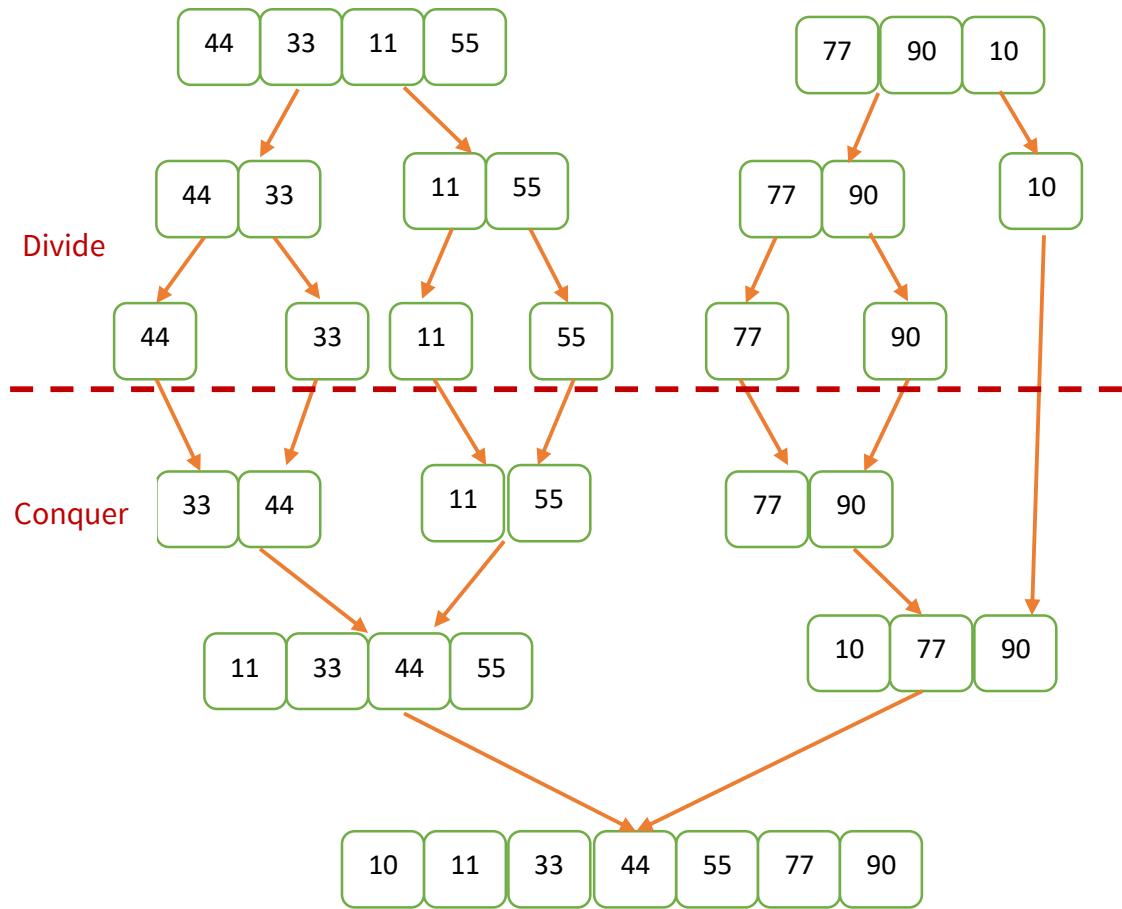
Merge sort ရဲ တွက်ပုံတွက်နည်းက Divide and Conquer ပုံစံပြုဖြစ်ပါတယ်။ ပထမတော့  
အစိတ်အပိုင်း ငယ်ယ်လေးဖြစ်အောင် ခွဲခွဲသွားပြီး ပြန်မှ ပြန်ပေါင်းတာဖြစ်ပါတယ်။

44, 33, 11, 55, 77, 90, 10

အရင်ဆုံး နှစ်ပိုင်းပိုင်းပါမယ်။ ရှေ့အပိုင်းက 0 to N/2 အထိဖြစ်ပြီး နောက်အပိုင်းကတော့ N/2+1 to N-1 အထိဖြစ်ပါတယ်။

N = 7

ပထမပိုင်း 0 to N/2 = 3, ဒုတိယပိုင်း N/2+1 to N-1 → 4 to 6



- အပေါ်ဆုံးကနေ Divide ဆိုပြီး စာတန်းထိုးပြထားတဲ့အထိက ခွဲတဲ့အပိုင်းပါ။ ထပ်ခါ ထပ်ခါ ခွဲပါတယ်။ ဘယ်ထိခွဲမလဲဆိုရင် တစ်ခုပဲကျွန်တော့ပြီး ထပ်ခွဲလို့ မရတော့တဲ့အထိ ထပ်ခါ ထပ်ခါ ခွဲပါမယ်။
- Conquer လို့ စာတန်းထိုးပြထားတဲ့ နေရာကနေ စပြီး ပြန်ပေါင်းတာပါ။ ဘာလို့ အဲဒီနေရာကနေ ပြန်ပေါင်းတာလဲဆိုရင် ခွဲတာ တစ်ခုစီပဲ ကျွန်တော့လို့ ထပ်ခွဲစရာ မရှိတော့လို့ ပြန်ပေါင်းတာပါ။ ပေါင်းတဲ့ အချိန်မှာတော့ တိုက်စစ်ပြီး ပေါင်းပါတယ်။ ငယ်စဉ်ကြီးလိုက်ဆို ငယ်တာ ရှေ့ကထားပြီး ပြန်ပေါင်းတာဖြစ်ပါတယ်။ နောက်ဆုံး အားလုံး ပေါင်းပြီးသွားပြီ ဆိုမှ sorting စီပြီးသား array ကို ရမှာ ဖြစ်ပါတယ်။

ပုံနဲ့ ရှင်းပြထားတဲ့ စာကို တွဲကြည့်လိုက်ရင် သဘောပေါက်မယ်ထင်ပါတယ်။

```
mergesort(a) /*accept a array as parameter*/
```

1. if  $N == 1$  then:

return a

2.  $L1 = a[0] \dots a[N/2]$
3.  $L2 = a[N/2+1] \dots a[N - 1]$
4.  $L1 = \text{mergesort}( L1 )$
5.  $L2 = \text{mergesort}( L2 )$
6. return  $\text{merge}( l1, \text{Length}(L1)-1, l2, \text{Length}(L2)-1 )$

**Line 1 and Line 2:** မှာစစ်ထားတာက array size တစ်ခုပဲ ကျန်တာလား။ ဒါဆို ခွဲစရာမလိုတော့လို လက်ခံထားတဲ့ array a ကိုပဲ return ပြန်ပေးလိုက်တယ်။

**Line 3 and Line 4:** မှာ ဝင်လာတဲ့ a array လေးကို 0 to  $N/2$  အထိကို array တစ်ခု၊  $N/2+1$  to  $N-1$  အထိကို array တစ်ခု ဆိုပြီး array ၂ ခုအားလုံးကိုယ်တာဖြစ်ပါတယ်။

**Line 4 and Line 5:** ခွဲထားတဲ့ array ၂ ခုကို 1 ခန်းပဲကျန်တော့တဲ့ အထိ ထပ်ခါ ထပ်ခါ ခွဲချင်လို recursion ပြန်ခေါ်လိုက်တာဖြစ်ပါတယ်။

**Line 6 :** ကတော့ ခွဲထားတဲ့ array ၂ ခုကို ပြန်ပေါင်းဖို့ merge function ကို လုမ်းခေါ်တာ ဖြစ်ပါတယ်။ ပထား array ရယ်၊ သူ့ရဲ့ upper bound ရယ်၊ ဒုတိယ array ရယ်၊ သူ့ရဲ့ upper bound ရယ် ထည့်ပေးလိုက်တာဖြစ်ပါတယ်။ upper bound ဆိုတာ array အခန်းအရေအတွက် length ထဲကနေ 1 နှင့်တာ သိပြီး ဖြစ်မှာပါ။

```
merge(a, UBA, b, UBB)
```

```
/* two arrays a and b, UBA is upper bound of a array, UBB is upper bound of b array*/
```

1. declare c as array
2. Set ia = 0 /\*index for a\*/
3. Set ib = 0 /\* index for b\*/
4. Set ic = 0 /\* index for c array to store sorted values\*/

```

5. Repeat while ia<= UBA and ib<=UBB: /*while a and b have elements */
6. if a[ia] < b[ib] then:
7. c[ic] = a[ia];
8. ia = ia + 1
9. Else:
10. c[ic]= b[ib]
11. ib = ib + 1
12. ic = ic +1
13. End of Loop
14. Repeat while ia <= UBA: /* a has elements but b hasn't*/
15. c[ic] = a[ia];
16. ia = ia + 1
17. ic = ic + 1
18. End of Loop
19. Repeat while ib <= UBB: /* b has elements but a hasn't*/
20. c[ic] = a[ib];
21. ib = ib + 1
22. ic = ic + 1
23. End of Loop
24. return c

```

---

- **Line 1 :** အဖော်ထည့်ဖို့ array c ကြော်တယ်။
- **Line 2, Line 3, Line 4 :** array a ကိုထောက်ဖို့ ia, array b ကို ထောက်ဖို့ ib, array c ကို ထောက်ဖို့ ic ဆိုပြီး သတ်မှတ်တယ်။ ပြီးတော့ အစဆုံး 0 ခန်းကနေ စထောက်ကြမှာ ဖြစ်လို့ variable 3 ခုလုံးထဲ 0 ထည့်လိုက်တယ်။
- **Line 5 to Line 13:** Array a ရော့၊ Array b နှစ်ခုလုံး data ရှိနေသေးတဲ့ အချိန်။

နှုန္နာကြည့်ကြရအောင်။

ia =0, ib = 0 , ic =0

a = { 11,22}, b= {10, 55, 77}

 ia, ib 0 ဖြစ်နေတဲ့အတွက် a[0] နဲ့ b[0] နဲ့ တိုက်မယ်။ b[0] ကင်းတော့ b[0] ကို c[ic], c[0] ထဲ သွားထည့်မယ်။ b ကို ထည့်လိုက်တာဖြစ်လို့ ib ကို ၁ တိုးမယ်။ c ထဲ ထည့်တာဖြစ်လို့ ic ကိုလည်း ၁ တိုးမယ်။ ia ကတော့ ထိစရာမလိုဘူး။

ia = 0, ib = 1, ic = 1, c = {10}

 ဒီတစ်ခါ a[ia] နဲ့ b[ib] ကိုကြည့်မယ်။ a[0] နဲ့ b[1] ဖြစ်တယ်။ a[0] ကင်းတယ်။ ဒီတော့ c[ic] ထဲကို a[ia] ကို ထည့်မယ်။ ပြီးရင် ia, ic ကို ၁ တိုးမယ်။ ib ကို ထိရန်မလို့။

ia = 1, ib = 1, ic = 2, c = {10, 11}

\*\*သတိထားမိမှာပါ။ a ထည့်ရင် ia ကို တစ်တိုးတယ်။ b ထည့်ရင် ib ကို တစ်တိုးတယ်။ c ကတော့ ဘာဖြစ်ဖြစ် အမြဲအထည့်ခံနေရတာဖြစ်လို့ a နဲ့ b ဘယ်သူ ငယ်ငယ် ic ကတော့ အမြဲ တစ်တိုးရပါတယ်။ ဒါကြောင့် ic တစ်တိုးတာကို if နဲ့ else ထဲမှာ မရေးပဲ **line 12** မှာ ထုတ်ရေးထားတာ ဖြစ်ပါတယ်။ If ထဲ တစ်ခါ၊ else ထဲ တစ်ခါ၊ နှစ်ခါ မရေးချင်လို့ ဖြစ်ပါတယ်။\*\*\*

 ဒီတစ်ခါ တိုက်စစ်ရမှာက a[1] နဲ့ b[1] ဖြစ်တယ်။ a[1] ကင်းတယ်။ ဒီတော့ c[ic] ထဲကို a[ia] ကို ထည့်မယ်။ ပြီးရင် ia, ic ကို ၁ တိုးမယ်။ ib ကို ထိရန်မလို့။

ia = 2, ib = 1, ic = 3, c = {10, 11, 22}

ကြည့်ကြည့်လိုက်ပါ။ a က a = { 11, 22 } ဆိုပြီး 0, 1 နှစ်ခန်းပဲရှိတာကို c ထဲ အကုန်ထည့်ပြီးသွားလို့ ia တန်ဖိုး 2 ဖြစ်ပြီး UBA = 1 ထက်ကြီးသွားပြီ။ ဆိုလိုတာက a, b နှစ်ခု မရှိတော့ဘူး။ b ပဲ ကျွန်ုတော့တယ်။ b က {10, 55, 77} ဆိုပြီး 0, 1, 2 သုံးခန်းရှိတာ ၀ တစ်ခန်းထည့်ပြီးလို့ ib က 1 ဖြစ်နေတယ်။ array နှစ်ခုလုံး ရှိနေရင်တာ တိုက်စစ်ပြီး ထည့်ရတာဖြစ်ပါတယ်။ အခုတော့ b array တစ်ခုပဲ ကျွန်ုတော့လို့ တိုက်စရာမလိုပဲ b ထဲကဟာတွေကို ကူးထည့်ဖို့ပဲ ကျွန်ုပါတော့တယ်။

## ကူးထည့်မယ်။

ဘယ်က စပြီး ကူးထည့်မလဲဆိုရင် 0 ခန်းထည့်ပြီးလို့ ib က 1 ဖြစ်နေတော့ 1 ကနေ UBB ထိ ကူးထည့်မှာဖြစ်ပါတယ်။ ဒါဆို b က {10, 55, 77} ရှိတာဆိုတော့ UBB က 2 ဖြစ်လို့ 1,2 နှစ်ခန်းကူးထည့်ပါမယ်။ ခုနက ic က 3 ဖြစ်နေပါပြီ။

$c[3] = b[1]$  ,  $c = c\{10,11,22, 55\}$

$c[4] = b[2]$ ,  $c\{10,11,22, 55, 77\}$

ကြည့်ပါ c ထဲကို b ထည့်တာဖြစ်လို့ တစ်ကြိမ်ပြီးတိုင်း ic နဲ့ ib ကို 1 တိုးပါတယ်။ အခုံဟာက b ကျန်နေခဲ့လို့ b ထဲက တန်ဖိုးတွေ c ထဲ ကူးထည့်တာကို ပြတာ ဖြစ်ပါတယ်။ a ကျန်ခဲ့ရင်လည်း a ထဲက ဟာတွေကို ထိနိုင်းအတိုင်း ကူးထည့်ရမှာဖြစ်ပါတယ်။

Algorithm မှာ a ကျန်သလား စစ်တာက **line 10 to Line 18** ဖြစ်ပါတယ်။

b ကျန်သလား စစ်တာက **line 19 to Line 23** ဖြစ်ပါတယ်။

နောက်ဆုံးတော့ a နဲ့ b ကို c ယုံစဉ်ကြီးလိုက် ပေါင်းပြီးသား array c ကို return ပြန်ပေးလိုက်တာ ဖြစ်ပါတယ်။ ဒါဆိုရင်တော့ merge sort ကို နားလည်လောက်ပြီထင်ပါတယ်။



### လောက်ချုပ်မှုပါ။

1. Merge sort ကို ကြိုးစဉ်cယ်လိုက် စီချင်ရင် ဘယ်နေရာကို ပြောင်းရမည်နည်း။
2. LA = 44, 33, 11, 55, 77, 90, 40, 60, 99, 22, 88, 66 ကို Merge sort algorithm ဖြင့် cယ်စဉ်ကြိုးလိုက် sorting စီပါ။

## Quick Sort

Quick sort ကတေသာ နောက်ဆုံးအခန်းကို pivot လိုပါတယ်။ ပြီးရင် pivot ထက်ယ်တာ တစ်ပိုင်း၊ ကြီးတာ တစ်ပိုင်း နှစ်ပိုင်းခဲ့ပါတယ်။ ခဲ့လိုက်တဲ့ အပိုင်းတစ်ပိုင်းချင်းကိုလည်း နောက်ဆုံး အခန်းကို pivot ယူပြီး ပြီးရင် pivot ထက်ယ်တာ တစ်ပိုင်း၊ ကြီးတာ တစ်ပိုင်း နှစ်ပိုင်းထပ်ခဲ့ပါတယ်။ နမူနာအနေနဲ့ ပြောရရင်

- 44, 33, 11, 55, 77, 90, 22, 66 ဆုံးရင် pivot က 66 ဖြစ်ပါမယ်။
- တစ်ကြိမ် loop ပတ်ပြီးရင် {44, 33, 11, 55, 22} 66 {77, 90} ဆုံးပြီး pivot ထက်ယ်တာ တစ်အုပ်စု pivot ထက်ကြီးတာ တစ်အုပ်စုဆုံးပြီး ရမယ်။ {44, 33, 11, 55, 22} ရဲ့ order ဟာအရေးမကြီးပါဘူး။ အရေးကြီးတာက ရှုံးအပိုင်းထဲမှာ 66 ထက်ယ်တာတွေချည်းပါမယ်။ နောက်အပိုင်းထဲမှာ 66 ထက်ကြီးတာတွေချည်းပါမှာ ဖြစ်ပါတယ်။
- ပြီးရင် pivot ထက်ယ်တာ ထပ်တွက်၊ pivot ထက်ကြီး တာ သပ်သပ် ပြန်တွက်မယ်။ ဒါကြောင့် quick sort ဟာလည်း recursion ဖြစ်မှာ ဖြစ်ပါတယ်။
- {44, 33, 11, 55, 22 } ကို ထပ်တွက်ရင် pivot = 22  
 $\{11\} 22 \{44, 33, 55\}$

ဒီတေသာ Quick Sort ဆိုတာ နောက်ဆုံးအခန်းကို pivot ယူပြီး pivot ထက်ယ်တာတစ်ပိုင်း၊ pivot ထက်ကြီးတာ တစ်ပိုင်းဆုံးပြီး ပိုင်းတယ်။

ပြီး ရင်ယ်တဲ့ ကြီးတဲ့ အပိုင်းတွေကို recursion နဲ့ ပြန်တွက်တယ်။ နောက်ဆုံး တစ်ခုကျန်တော့တဲ့ အထိ recursion ခေါ်ပြီး ပြန်တွက်ရမယ်။

**quickSort(LA, LB, UB)**

1. if LB < UB then:
2.     Loc = partition(LA, LB, UB)
3.     quickSort(LA, LB, Loc-1)
4.     quickSort(LA, Loc+1, UB)

QuickSort algorithm ကနေ partition function ကိုလျစ်ခေါ်တယ်။ partition function သဲ return ပြန်ပေးလိုက်တဲ့ location အပေါ်မူတည်ပြီး ငယ်တာတစ်ပိုင်း၊ ကြီးတာ တစ်ပိုင်း ထပ်ခဲ့တယ်။ ခဲ့ထားတဲ့ နှစ်ပိုင်းအတွက် recursion ပြန်ခေါ်တယ်။ ဒါပါပဲ။

**partition( LA, LB, UB)**

1. Loc = LB
2. pivot = LA[UB]
3. Left = LB
4. Right = UB
5. Repeat while true:
  6.     Repeat while LA[left] < pivot:
    7.         Left = Left + 1
    8.         End of Loop
  9.     Repeat while LA[Right] > pivot and Right>LB:
    10.         Right = Right - 1
    11.         End of Loop
  12.     If left >=right then:
    13.         Break
    14.         Else:
      15.             swap (Left , right)
  16. End of Loop
  17. Swap (Left, UB)
  18. Return Left

- Left က LB၊ right က UB ကနေစမယ်။ ဒါကြောင့် left ဟာ Right ထက်ယ်တယ်။ Left ဟာ Right ထက် ငယ်နေသရွှေ့ အောက်ကအလုပ်တွေကို ထပ်ခါထပ်ခါလုပ်မယ်။ (Line 5)
- ပြီးရင် left အခန်းထဲက တန်ဖိုးက pivot ထက်ယ်နေသမျှ left ကို ၁ တိုးမယ်။ ဆိုတော့ left အခန်းတန်ဖိုး pivot ထက် ကြိုးရင် သို့မဟုတ် ညီရင် ရပ်မယ်။ (Line 6 to 8)
- ပြီးရင် Right အခန်းထဲက တန်ဖိုးက pivot ထက်ကြိုးနေပြီး Right သည် LB ထက် ကြိုးနေသေးသမျှ Right ကို 1 လျော့မယ်။ ဆိုတော့ right အခန်းတန်ဖိုး pivot ထက် ငယ်ရင် ရပ်မယ်။ (Line 9 to 11)
- ပြီးသွားလို့
  1. Left >=Right ဆို break လုပ်မယ်။ break ဆိုတာကို outer loop ထဲ မှာ ရေးထားတာဖြစ်လို့ outer loop ထဲက ထွက်ပါမယ်။ break ဆိုတာရောက်တဲ့နေရာကနေ ထွက်တာဖြစ်ပါတယ်။
    - Outer loop ထဲက ထွက်တာနဲ့ swap(left, UB) လုပ်မယ်။ (Line 17)
    - Left ကို return ပြန်တယ်။ (Line 18)
    - ဒီတော့ partition function ကို call ခေါ်ထားတဲ့ quicksort ကိုပြန်ရောက်ပြီး ၀ to Left- 1 တစ်ပိုင်း၊ left + Left+1 to UB ဆိုပြီး ၃ ပိုင်းပိုင်းပါတယ်။
    - ၀ to Left- 1 , Left+1 to UB တို့အတွက် recursion ပြန်ခေါ်ပါမယ်။
  2. left က Right ထက်ယ်နေတယ်ဆိုရင် Left နဲ့ Right ကို နေရာချင်းလဲမယ်။ ပြီးရင် end of loop ဆိုလို့ outer loop (Linef 5) ကို ပြန်တက်တယ်။ outer loop ရဲ့ condition နေရာမှာ True ထည့်ထားလို့ looping ဟာ ဘယ်တော့မှ မထွက်တော့ပဲ ပတ်နေမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် အောက်မှာ break ကို သုံးပြီး ထွက်ရတာဖြစ်ပါတယ်။ (Line 12 and 13)

Trace လိုက်ကြည့်ကြရအောင်။

Left = LB = 0

Right = UB = 7

44, 33, 11, 55, 77, 90, 22, **66**: pivot = 66  


Left ကို 1 တိုးတိုးပြီး အခန်းထဲ တန်ဖိုး pivot ထက် ငယ်သရွှေ့ ရွှေ့တယ်။

77(left = 4) မှာ **ကြီး**တော့ ရပ်မယ်။

Right ကို 1 လျော့လျော့ပြီး အခန်းထဲ တန်ဖိုး pivot ထက် **ကြီး**သရွှေ့ ရွှေ့တယ်။

22(Right = 6) မှာ ငယ်တော့ရပ်တယ်။

Left (4)  $\geq$  Right(6) ဆိုတော့ false ဖြစ်လို့ swap (Left, Right) လုပ်မယ်။

Swap (4, 6)

44, 33, 11, 55, 22, **90**, 77, 66, Left = 4, Right = 6  


Left ကို 1 တိုးတိုးပြီး အခန်းထဲ တန်ဖိုး pivot ထက် ငယ်သရွှေ့ ရွှေ့တယ်။

90(left = 5) မှာ **ကြီး**တော့ ရပ်မယ်။

Right ကို 1 လျော့လျော့ပြီး အခန်းထဲ တန်ဖိုး pivot ထက် **ကြီး**သရွှေ့ ရွှေ့တယ်။

22(Right = 4) မှာ ငယ်တော့ရပ်တယ်။

Left (5)  $\geq$  Right(4) ဆိုတော့ true ဖြစ်လို့ break ဆိုပြီး outer loop ထဲက ထွက်မယ်။

Outer loop အပြင်မှာ swap(Left, UB) ဆိုတော့ swap (5,7)

44, 33, 11, 55, 22, 66, 77, 90

Return left လုပ်လိုက်တော့ return 5 ဖြစ်သွားမယ်။

LB to Left-1( 0 – 4), left(5), left+1 to UB(6 to 7) ဆိုပြီး 3 ပိုင်းခဲ့တော့

{44, 33, 11, 55, 22} 66 {77,90} ဆိုပြီးရမယ်။

**Recursion for {44, 33, 11, 55, 22} 66 Recursion for {77, 90}** ဆိုပြီးရမယ်။

**Recursion for {44, 33, 11, 55, 22} :** Pivot = 22, LB =0, UB = 4, Left = LB = 0, Right = UB = 4

44, 33, 11, 55, 22 (left က 44 မှာ pivot ထက်ကြီးတော့ရပ်, Right က 11 မှာ pivot ထက်ယောက်လိုရပ်)

Left(0) >= Right (2) က false ထွက်လို Swap(left, right) = Swap(0, 2) လုပ်မယ်။

11, 33, 44, 55, 22



Left(0) ကနေ အခန်းနံပါတ် ၁တိုးတိုး ဆက်သွား 33(1) မှာ pivot ထက်ကြီးလို ရပ်တယ်။

Right(2) ကနေ အခန်းနံပါတ် ၁လျော့လျော့ပြီး ဆက်သွား ၀ ခန်းရောက်လို ရပ်တယ်။

Left(1) >= Right (0) ဖြစ်လို outer loop ထဲကထွက်။

Swap(Left, UB) ဆိုတော့ Swap (1, 4) ကို လုပ်တယ်။

11, 22, 44, 55, 33

Left ကို retrun ပြန်။ retrun 1 ဆိုတော့ 0 to 0, 1, 2 to 4 ဆိုပြီး 3 ပိုင်းပိုင်းပါတယ်။

{11} 22 {44, 55, 33}

{11} 22 **Recursion for {44, 55, 33}**

**Recursion for {44, 55, 33} :** Pivot = 33, Left = 2, Right = 4

44, 11, 33 (left ကဲ 44(2) မှာ pivot ထက်ကြီးတော့ရပ်, Right က 11(3) မှာယ်လို့ရပ်)  


Left(2) >= Right(3) ဆိုတော့ မှားလို့ swap(left, right) ဆိုတော့ swap(2,3) လုပ်မယ်။

11, 44, 33(left ကဲ 44(3) မှာ pivot ထက်ကြီးတော့ရပ်, Right က 11 မှာ LB ရောက်လို့ရပ်)  


Left(3) >= Right (2) ဖြစ်လို့ outer loop ထဲကထွက်။

Swap (Left ,UB) = Swap (3,4)

11, 33, 44

Left ကို retnrun ပြန်။ retnrun 3 ဆိုတော့ LB(2) to 3, 3, 4 to UB(4) ဆိုပြီး 3 ပိုင်းပိုင်းပါတယ်။

{11} 33 {44}

**Recursion for {44, 33, 11, 55, 22} :** {11} 22 **Recursion for {44, 55, 33}**

: 11, 22, 33, 44, 55

**Recursion for {77, 90}:** Pivot = 90, LB =6, UB = 7, Left = LB = 6, Right = UB = 7

77, 90 (left ကဲ 90 မှာ pivot နဲ့ညီတော့ရပ်, Right က 77 မှာ LB ခန်းရောက်လို့လို့ရပ်)  


Left(7) >= Right (6) ဆိုတော့မှန်လို့ break ဆိုပြီး outer loop ထဲကထွက်။

Swap(Left, UB) ဆိုတော့ Swap (7, 7) ကို လုပ်တယ်။

77, 90

**Recursion for {44, 33, 11, 55, 22} 66 Recursion for {77, 90}**

11, 22, 33, 44, 55, 66, 77, 90 ဆိုပြီး ရသွားပါတယ်။



### လေ့ကျင့်ကြည့်ကြပါ။

1. Quick sort ကို ကြိုးစဉ်ထုတ်ပေါ်ကို စီချင်ရင် ဘယ်နေရာကို ပြောင်းရမည်နည်း။
2. LA = 44, 33, 11, 55, 77, 90, 40, 60, 99, 22, 88, 66 ကို Quick sort algorithm ဖြင့် စဉ်ထုတ်ပေါ်လိုက် sorting စီပါ။



### Chapter အနှစ်ချုပ်

- ဒီအခန်းမှာ တော့ မသိမဖြစ်တဲ့ အခြေခံ sorting algorithm တွေကို ရှင်းပြုခဲ့တာ ဖြစ်ပါတယ်။ တစ်ခြား sorting algorithm တွေလည်းရှိပါသေးတယ်။
- Merge sort နဲ့ Quick sort ကတော့ အပိုင်းလေးတွေခဲ့ခြားပြန်စုတဲ့ ပုံစံနဲ့ သွားပါတယ်။
- ပြီးတော့ သူတို့ နှစ်ခုဟာ recursion ဖြစ်တဲ့ အတွက် space complexity ပိုမှာ ဖြစ်ပါတယ်။
- Algorithm တွေကို Trace လိုက်ပြီး ဘယ် algorithm ကတော့ ပိုပြီးတွက်ရတယ်၊ ဘယ် algorithm နည်းနည်းတွက်ရတယ်လိုပြီး ကြောချိန်ကို ခန့်မှန်းလို့ရပေမဲ့ time complexity ချုတ်တာလောက်တော့ မတိကျနိုင်ပါဘူး။
- အစကတော့ Algorithm တွေရဲ့ time complexity (Worst Case (Big O notation (O)) နဲ့ Best Case (Omega Notation ( $\Omega$ ))) တို့နဲ့ ပါတွက်ချက်ပြီး နိုင်းယူဉ်ပြဖို့ ရည်ရွယ်ခဲ့ပေမဲ့ အရမ်း ရွှေ့ပွဲတွေးကုန်မှာ စိုးလို့ မထည့်လိုက်ပါဘူး။ နောက်အခြေအနေပေးရင် algorithm တစ်ခုကို ဘယ်လို့ space complexity, time complexity တွက်တာလေး လုပ်ပေးပါမယ်။

“ကျွန်မက သိပ်အပျင်းကြီးတော့

သီချင်းဆို ဘာသီချင်းညာသီချင်း ရွှေးဖွင့်နေရတာ မလုပ်ချင်ဘူး၊

ပြီးတော့ ကိုယ်က သီချင်းရွှေးဖွင့်ထားရင်

ဘာပြီး ဘာလာတော့မှာ သိနေတာဆိုတော့ ပျင်းစရာကြီး၊

ဒီတော့ စာတွက်ရင် ဘေးမှာ FM ဖွံ့ဖြိုးလိုက်တာပဲ၊

ဒီတော့ စာလုပ်ရတာ အထူးသဖြင့် ဒီလို နားလည်ရခက်တာတွေကို လေ့လာတဲ့အခါမှာ  
ပျင်းစရာမကောင်းတော့ဘူး၊

အထပ်ထပ်အခါခါ ပြန်ပြီး trace လိုက်လည်း FM သာဖွင့်ထား

ပျင်းစရာမကောင်းတော့ဘူး၊

တစ်ခြားဘေးက အသံတွေလည်း မကြားရလို့

အနောင့်အယုက်လည်း မဖြစ်တော့ဘူး၊

ပြီးတော့ စာထဲစိတ်ဝင်စားသွားရင်

ဘာသီချင်းလာနေမှန်းလည်း မသိတော့ဘူး”

## အခန်း (၁၅)

### Linked List

#### Linked List

Linked list မှာ

1. Simple Linked List
2. Doubly Linked List
3. Circular Linked List ဆိုပြီး 3 မျိုး ရှိပါတယ်။

Simple Linked List ကို နမူနာပြုပါမယ်။



|    | Value       | Link |
|----|-------------|------|
| 0  |             |      |
| 1  | I           | 3    |
| 2  |             |      |
| 3  | --- (space) | 4    |
| 4  | C           | 6    |
| 5  |             |      |
| 6  | A           | 7    |
| 7  | N           | 10   |
| 8  |             |      |
| 9  |             |      |
| 10 | ---(Space)  | 12   |
| 11 |             |      |
| 12 | F           | 13   |
| 13 | L           | 14   |
| 14 | Y           | NULL |

Simple Linked List (Singly Linked List လိုလည်း ခေါ်ပါတယ်) ဆိုတော့ variable name ကို SLL ပဲ  
ထားလိုက်ကြရအောင်။

Head ဟာ 1 ဖြစ်နေလို့

$SLL[1] \rightarrow Value = I$  ရမယ်။ ပြီးတော့  $SLL[1] \rightarrow Link = 3$  ကို ဆက်သွားမယ်။

$SLL[3] \rightarrow Value = space$  ရမယ်။ ပြီးတော့  $SLL[3] \rightarrow Link = 4$  ကို ဆက်သွားမယ်။

$SLL[4] \rightarrow Value = C$  ရမယ်။ ပြီးတော့  $SLL[4] \rightarrow Link = 6$  ကို ဆက်သွားမယ်။

$SLL[6] \rightarrow Value = A$  ရမယ်။ ပြီးတော့  $SLL[6] \rightarrow Link = 7$  ကို ဆက်သွားမယ်။

$SLL[7] \rightarrow Value = N$  ရမယ်။ ပြီးတော့  $SLL[7] \rightarrow Link = 10$  ကို ဆက်သွားမယ်။

$SLL[10] \rightarrow Value = Space$  ရမယ်။ ပြီးတော့  $SLL[10] \rightarrow Link = 12$  ကို ဆက်သွားမယ်။

$SLL[12] \rightarrow Value = F$  ရမယ်။ ပြီးတော့  $SLL[12] \rightarrow Link = 13$  ကို ဆက်သွားမယ်။

$SLL[13] \rightarrow Value = L$  ရမယ်။ ပြီးတော့  $SLL[13] \rightarrow Link = 14$  ကို ဆက်သွားမယ်။

$SLL[14] \rightarrow Value = Y$  ရမယ်။ ပြီးတော့  $SLL[14] \rightarrow Link = NULL$  ဆိုတော့ ဆက်သွားစရာ  
မလိုတော့ဘူး။ Head ကနေ စပြီးသွားလိုက်တာ “I CAN FLY” ဆိုတဲ့ အဖြေကို ရမှာဖြစ်ပါတယ်။



## လေ့ကျင့်ကြည့်ကြရအောင်။

ဒဲမှာ Poem ဆိုတဲ့ pointer ကနေ “Life is a challenge, meet it.” ဆိုတာကို ထောက်ချင်ပါတယ်။

Agreement pointer ကနေ “Try Our Best!” ဆိုတာကို ထောက်ချင်ပါတယ်။ လိုအပ်တဲ့ နေရာတွေကို ဖြည့်ပေးပါ။

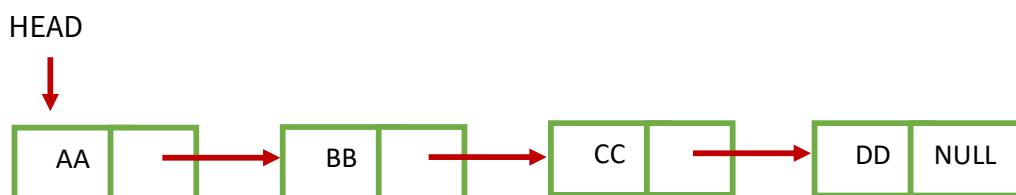
| Poem | Value     | Link |
|------|-----------|------|
|      | Life      |      |
|      | is        |      |
|      | Try       |      |
|      | a         |      |
|      |           |      |
|      | Our       |      |
|      | challenge |      |
|      |           |      |
|      |           |      |
|      | ,         |      |
|      | meet      |      |
|      | Best!     |      |
|      | It.       |      |
|      |           |      |
|      |           |      |

## Simple Linked List



ဒီလို နှစ်ခုအတွဲလိုက်လေးနဲ့ ပြလေ့ရှိပါတယ်။ အဲဒီအတွဲလေးကို Node လိုပေါ်ပါတယ်။ Node ထဲမှာ တန်ဖိုးသိမ်းဖို့ value ရယ်၊ နောက် Node ကို ထောက်ဖို့ Next ရယ်ဆိုပြီး ဂိုင်းပါပါတယ်။

1. Value - ဟာ ကိုယ်က ကိန်းပြည့်သိမ်းချင်ရင် int data type ဖြစ်ပြီး၊ အသာမ သိမ်းချင်ရင် float/double data type, စာလုံးလေး တစ်လုံးသိမ်းချင်ရင် char data type၊ စာသားသိမ်းချင်ရင် String data type စသည်ဖြင့် ထားလိုဂါတယ်။
2. Next - ကတော့ နောက် Node တစ်ခုကို ထောက်မှာဖြစ်လို့ Node pointer ဖြစ်ရပါမယ်။ pointer ဆိုတာ ထောက်တာပါ။ int ကို ထောက်ရင် int pointer, float ကို ထောက်ရင် float pointer။ ဒီမှာက Node ကိုထောက်တော့ Node pointer ဖြစ်ပါမယ်။ Node pointer ကို Node\* လိုပေးပါတယ်။



Head → Value = AA

AA → Next → Value = BB

BB → Next → Value = CC

CC → Next → Value = DD

DD → Next ဟာ NULL ဖြစ်တော့ ရပ်သွားရော်။ AA, BB, CC, DD ဆိုပြီးရပါတယ်။

### Traversing A Simple Linked List

Traversing ဆိုတော့ HEAD ကနေ စပြီး Next Next ဆိုပြီး နောက်ကို ဆက်ဆက်ပြီး NULL ရောက်သည်အထိ အခန်းစွဲသွားမှာ ဖြစ်ပါတယ်။

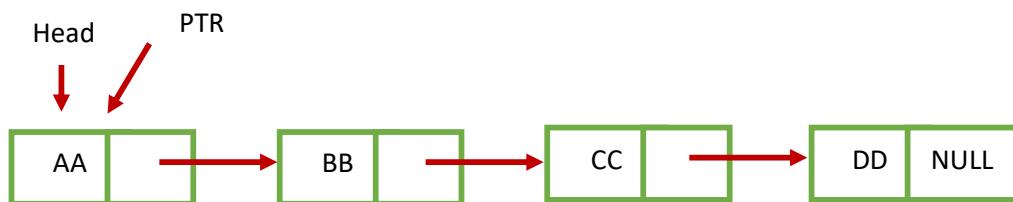
1. If HEAD == NULL then:

2. Write "Empty"
3. Else:
4. SET PTR = HEAD
5. Repeat while PTR != NULL then:
6.     Do something on PTR → Value
7.     PTR = PTR → Next
8. End of Loop

Head ဆိုတာ အမြတ်စီမံရှုံး Node ကို ထောက်ထားရပါတယ်။ နောက်ကို လျောက်ခွဲလို့မရပါဘူး။

ဒါကြောင့် line 4 မှာ PTR ထဲကို Head ထည့်လိုက်ပြီး PTR ကိုပဲ NULL မရောက်မချင်း ဆက်ပြီး ချွေားမှာဖြစ်ပါတယ်။ Head ကတေသာ ရှုံးမှာပဲ ကျွန်ုန်းများဖြစ်ပါတယ်။ Trace လိုက်ပြပါမယ်။

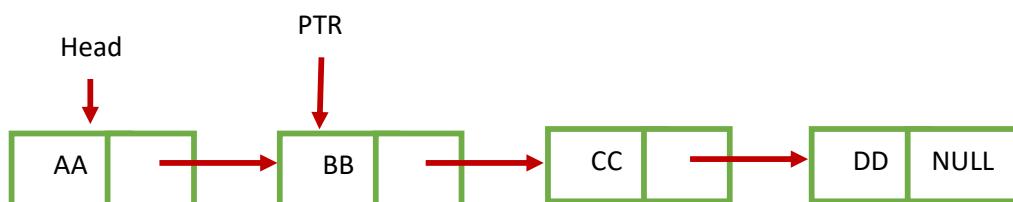
Line 4 : SET PTR = HEAD



Line 5: PTR က NULL နဲ့ မညီဘူး။

Line 6: ဒီတေသာ PTR → Value ဖြစ်တဲ့ AA ပေါ်မှာ တွက်တာချက်တာ၊ ထုတ်ပြတာ ကြိုက်တာလုပ်ပါ။

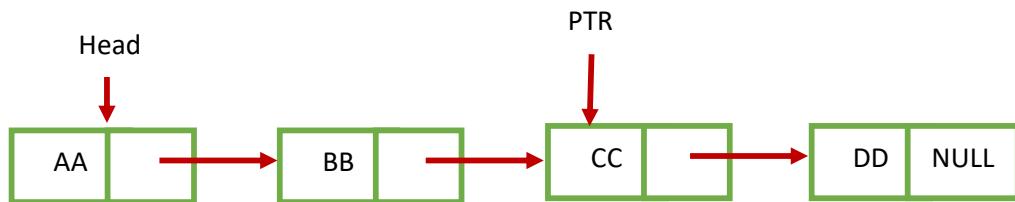
Line 7: PTR = PTR → Next ကို ထည့်တယ်ဆိုတာ update လုပ်တာဖြစ်ပါတယ်။ PTR ဟာ AA Node နေရာကနေ BB Node နေရာကို ရောက်လာမှာဖြစ်ပါတယ်။



📍 Line 5: PTR ဟာ NULL နဲ့ မည့်ဘူး။

Line 6: ဒီတော့ PTR → Value ဖြစ်တဲ့ BB ပေါ်မှာ တွက်တာချက်တာ၊ ထုတ်ပြတာ ကြိုက်တာလုပ်ပါ။

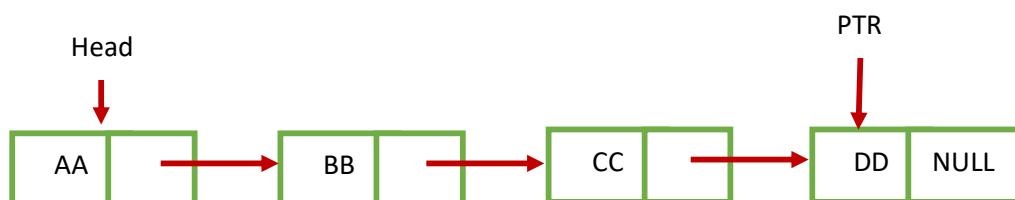
Line 7: PTR = PTR → Next ကို ထည့်တယ်ဆိုတာ update လုပ်တာဖြစ်ပါတယ်။ နောက်တစ် Node ကို ရွှေတာ ဖြစ်ပါတယ်။ PTR ဟာ BB နေရာကနေ CC Node နေရာကို ရောက်လာမှာ ဖြစ်ပါတယ်။



📍 Line 5: PTR ဟာ NULL နဲ့ မည့်ဘူး။

Line 6: ဒီတော့ PTR → Value ဖြစ်တဲ့ CC ပေါ်မှာ တွက်တာချက်တာ၊ ထုတ်ပြတာ ကြိုက်တာလုပ်ပါ။

Line 7: PTR = PTR → Next ကို ထည့်တယ်ဆိုတာ update လုပ်တာဖြစ်ပါတယ်။ နောက်တစ် Node ကို ရွှေတာ ဖြစ်ပါတယ်။ PTR ဟာ CC နေရာကနေ DD Node နေရာကို ရောက်လာမှာ ဖြစ်ပါတယ်။



📍 Line 5: PTR ဟာ NULL နဲ့ မည့်ဘူး။

Line 6: ဒီတော့ PTR → Value ဖြစ်တဲ့ DD ပေါ်မှာ တွက်တာချက်တာ၊ ထုတ်ပြတာ ကြိုက်တာလုပ်ပါ။

Line 7: PTR = PTR → Next ကို ထည့်တယ်ဆိုတာ update လုပ်တာဖြစ်ပါတယ်။ နောက်တစ် Node ကို ရွှေတာ ဖြစ်ပါတယ်။  $PTR = PTR \rightarrow Next = NULL$  ဆိုပြီး PTR ထဲ ကို NULL ရောက်သွားမှာ ဖြစ်ပါတယ်။

-  Line 5: PTR ဟာ NULL နဲ့ညီသွားတော့ looping ရပ်သွားမှာ ဖြစ်ပါတယ်။

### Displaying a Simple Linked List

1. If HEAD == NULL then:
2.     Write “Empty”
3. Else:
4.     SET PTR = HEAD
5.     Repeat while PTR != NULL:
6.         Write PTR → Value, “ ”
7.         PTR = PTR → Next
8.     End of Loop

ဒါကတော့ ရည်ရည်ဝေးဝေး ထပ်ရှင်းပြစ်ရာမလိုလောက်တော့ဘူး ထင်ပါတယ်။ အပေါ်က Traversing ရဲ့ do something နေရာမှာ Write ဖြစ်သွားတာလေးပါပဲ။



### လေ့ကျင့်ကြည့်ကြရအောင်။

1. Node ရဲ့ Value ဟာ int ဖြစ်တယ်ဆိုပါတော့။ Linked List ထဲက Node အားလုံးရဲ့ Value အားလုံးပေါင်းလဒ်ကို တွက်ပေးတဲ့ algorithm ကို ရေးပေးပါ။ ပေါင်းမှာ ဖြစ်တဲ့အတွက် total ဆိုတဲ့ အဖြစ်ထည့်မည့် variable ထဲကို 0 assign လုပ်ထားပေးပါ။ ပြီးမှ node တစ်ခုချင်းရဲ့ value တွေကို total ထဲ သွားသွားပေါင်းပါ။ Node အားလုံးတန်ဖိုး ပေါင်းတာ ပြီးသွားလို ဆက်သွားစရာမရှိတော့ပဲ NULL နဲ့ညီသွားပြီ ဆိုမှ total ကို return ပြန်ပေးပါ။
2. Linked list ထဲမှ Value 100 ထက်ကြီးတာတွေကို ထုတ်ပြမည့် algorithm ကို ရေးပါ။

3. ဂဏန်းတစ်လုံးကို parameter အဖြစ်လက်ခံပြီး အဲဒီတန်ဖိုးကို Linked list ထဲမှာ ပါမပါလိုက်ရှာပါ။ တွေ့တယ်ဆိုရင် parameter is found ဆိုပြီး ထုတ်ပြပါ။ မတွေ့ဘူးဆိုရင် parameter is not found လို့ ထုတ်ပြပါ။

### Inserting a Node into Simple Linked List

Node တစ်ခုကို simple linked list ထဲ ထည့်တော့မယ် ဆိုရင် ဖြစ်နိုင်ချေ 2 မျိုး ရှိပါတယ်။

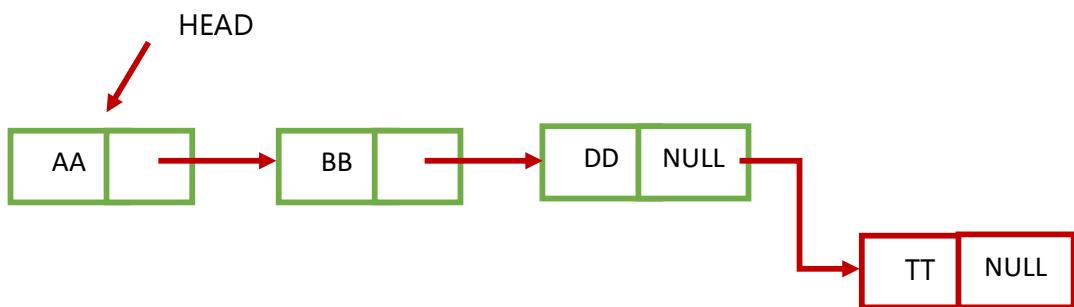
1. Node တစ်ခုမှ မရှိသေးလို့ HEAD ကထောက်စရာမှရှိပဲ NULL ဖြစ်နေတယ်။ ဒါတော့ ထည့်တဲ့ Node က ပထမဆုံး Node ဖြစ်သွားလို့ HEAD ထဲ ထည့်ရတာ။
2. လက်ရှိ နောက်ဆုံး Node ကိုရှာပြီး သူရဲ့နောက်မှာ သွားထည့်တာ။

#### 1. ရွှေ့ဆုံး Node အဖြစ်ထည့်တာ

If HEAD == NULL then:

HEAD = newNode

#### 2. နောက်ဆုံး Node ရဲ့နောက်မှာ သွားထည့်တာ။



TT Node က အသစ်ထည့်မည့် newNode ဖြစ်ပါတယ်။ ဒါတော့ နောက်ဆုံး Node ကို အရင်ရှာရမှာ ဖြစ်ပါတယ်။ နောက်ဆုံး Node ဆိုတာ Next က NULL ဖြစ်နေတဲ့ Node ဖြစ်ပါတယ်။ ပုံမှာ ကြည့်ကြည့်ပါ။ ဒါကြောင့် PTR ထဲကို HEAD ထည့်ပြီး Next က NULL နဲ့ မညီမချင်း သွားပါမယ်။ Next

က NULL ဖြစ်ပြီ ဆိတေသူမှ နောက်ဆုံး Node ကို ရောက်သွားပြီ ဖြစ်လို့ PTR ရဲ့ Next ထဲကို newNode ကို ထည့်ရမှာ ဖြစ်ပါတယ်။

Set PTR = HEAD

Repeat while PTR → Next != NULL:

PTR = PTR → Next

End of Loop

PTR → Next = newNode

Inserting Algorithm ကို အစအဆုံး ပြန်ရေးမယ်ဆိုရင်

#### Insert(newNode)

1. If HEAD == NULL then:
2.       HEAD = newNode
3. Else:
4.       Set PTR = HEAD
5.       Repeat while PTR → Next != NULL:
6.           PTR = PTR → Next
7.       End of Loop
8.       PTR → Next = newNode

#### Deleting a Node from Simple Linked List

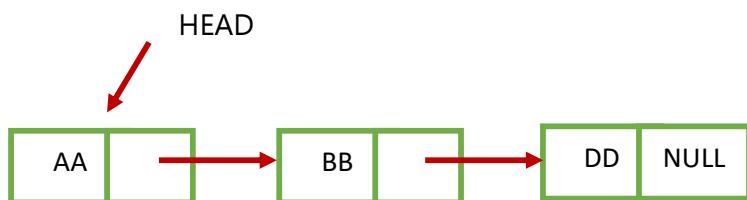
Value ကိုပေးမယ်။ အဲဒီပေးလိုက်တဲ့ value နဲ့တူတဲ့ value ရှိတဲ့ Node တစ်ခုကို simple linked list

ထဲကနေ လိုက်ရှာပြီး ဖျက်မယ်ဆိုရင် ဖြစ်နိုင်ချေ ၃ မျိုး ရှိပါတယ်။

1. ရှေ့ဆုံး Node ကိုဖျက်တာ၊
2. အလယ် Node ကို ဖျက်တာ၊

3. နောက်ဆုံး Node ကို ဖျက်တာ ထို့ဖြစ်ပါတယ်။

1. ရှေ့ဆုံး Node ကိုဖျက်တာ၊



**AA Node ကို ဖျက်ပါမယ်။**

- ရှေ့ဆုံး Node ကို ဖျက်မှာဖြစ်လို့ ဒုတိယ Node ကို HEAD က သွားထောက်ရမှာ ဖြစ်ပါတယ်။
- ပုံအရဆိုရင် BB Node ကို Head ကသွားထောက်ရမှာ ဖြစ်ပါတယ်။
- ဒုတိယ Node ကို Head ကသွားမထောက်ခင် PTRထဲကို Head ထည့်ပါမယ်။
- ပြီးမှ PTR ကို ဖျက်ပစ်ရမှာဖြစ်ပါတယ်။

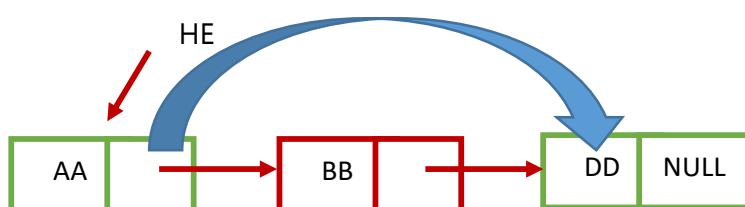
Set PTR = HEAD

If HEAD → value == x then:

    HEAD = HEAD → Next

    Free (PTR)

2. အလယ် Node ကိုဖျက်တာ၊



**BB Node ကို ဖျက်ချင်တယ် ဆိုပါတော့။**

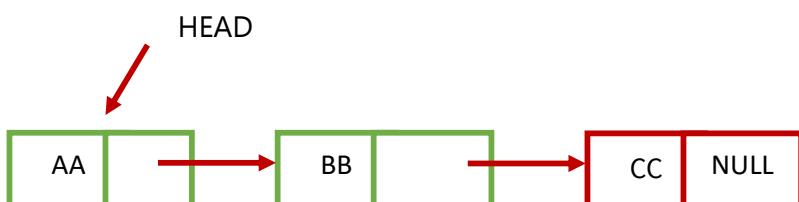
- ဖျက်မည့် Node ရဲ့ အရှေ့က Node (ပုံအရဆိုရင် AA Node) ကို prev လိုထားပြုမယ်။
- Arrow အပြောလေးနဲ့ ပြထားသလိုပါပဲ prev ရဲ့ Next ထဲကို ဖျက်မည့် Node ရဲ့ Next ကို ထည့်ရမှာ ဖြစ်ပါတယ်။ (ပြထားတဲ့ ပုံအရဆိုရင် Prev ရဲ့ Next ဆိုတာက AA ရဲ့ Next ဖြစ်ပြီး ဖျက်မည့် Node BB ရဲ့ Next ဆိုတာ DD ကို ပြောတာဖြစ်ပါတယ်။ BB ဖျက်လိုက်လို AA Node က DD Node ကို သွားထောက်ဖို့ ဖြစ်ပါတယ်။)

ဒါကြောင့် Prev ဆိုတဲ့ pointer တစ်ခုထပ်ပြီးလိုပါတယ်။ PTR က နောက်ကို မရွှေ့ခဲ့ PTR နေရာကို Prev ထားပေးခဲ့ရမှာ ဖြစ်ပါတယ်။ ဒါမှာ PTR က ဖျက်မည့် Node ကို ရောက်သွားချိန်မှာ ဖျက်မည့် Node ရဲ့ ရှေ့ Node ကို Prev က ထောက်ထားမှာ ဖြစ်ပါတယ်။

$\text{Prev} \rightarrow \text{Next} = \text{PTR} \rightarrow \text{Next}$

Free (PTR)

### 3. နောက်ဆုံး Node ကို ဖျက်တာ။



နောက်ဆုံး Node ကို ဖျက်မယ် ဆိုတာ ဘယ်လို သိမလဲဆိုရင် ဖျက်မည့် Node ရဲ့ Next ဟာ NULL နဲ့ ညီရင် ဖျက်မည့် Node ဟာ နောက်ဆုံး Node ဖြစ်ပါတယ်။ နောက်ဆုံး Node CC ကို ဖျက်မယ်ဆိုရင် Prev Node (BB Node) ဟာ နောက်ဆုံး Node ဖြစ်သွားမယ်။ ဒါတော့ BB ရဲ့ Next ထဲကို NULL ထည့်ပေးရမယ်။

`BBNode → Next = NULL` လိုပေးတာနဲ့ `BBNode→ Next = CCNode →Next` အတူတူပါပဲ။  
 ဘာကြောင့်လဲဆိုရင် `CCNode` ရဲ့ `Next` ဟာလည်း `NULL` ဖြစ်နေလိုပါ။ ဒါကြောင့် `Prev → Next = PTR`  
 $\rightarrow$ `Next` လိုပေးလိုပါတယ်။ (`PTR` ဆိုတာ ဖျက်မည့် `Node`, `Prev` ဆိုတာ ဖျက်မည့် `Node` ရဲ့ `Node`  
 $\rightarrow$ `Next` ဖြစ်ပါတယ်။)

`Prev → Next = PTR → Next`

`Free (PTR)`

တူနေလို့ အလယ် `Node` ဖျက်တာနဲ့ နောက်ဆုံး `Node` ဖျက်တာကို ပေါင်းရေးပါမယ်။

`Set Prve = PTR`

`While PTR → Value != x and PTR != NULL then:`

`Prev = PTR`

`PTR = PTR → Next`

`End of Loop`

`If PTR==NULL then:`

`Return false`

`Else:`

`Prev → Next = PTR → Next`

`Free (PTR)`

`Return true`

ဖျက်လို့ အောင်မြင်ရင် အောင်မြင်ကြောင်း `return true`, ရှာမတွေ့လို့ မဖျက်ရဘူးဆိုရင်  
 မအောင်မြင်လို့ `return false` ဆိုပြီး ပြန်ပေးမှာ ဖြစ်ပါတယ်။

PTR ဟာ ရှေ့ဆုံး Node ကနေစပြီး ရှာလိုက်တာ တွေ့ရင်ရပ်သွားမှာဖြစ်ပါတယ်။ မတွေ့လိုဆက်သွားရင်တော့ နောက်ဆုံး NULL နဲ့ ညီသွားမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် NULL နဲ့ ညီသွားသလားဆိုပြီး while loop မှာ ထည့်စစ်ထားတာဖြစ်ပါတယ်။ Loop ထဲက ထွက်လာတဲ့အချိန်မှာလည်း PTR ဟာ NULL နဲ့ ညီလို ထွက်လာတာလား စစ်တယ်။ NULL နဲ့ ညီလိုထွက်လာတာဆိုရင် မတွေ့လို PTR ဟာ NULL အထိ ရောက်သွားတာဖြစ်ပါတယ်။ PTR ဟာ NULL နဲ့ မညီပဲ ထွက်လာတယ်ဆိုရင်တော့ PTR → Value == x ဖြစ်လိုဖြစ်ပါတယ်။ x ကို တွေ့လိုထွက်လာတာဖြစ်ပါတယ်။

Delete Algorithm ကို အစအဆုံး ပြန်ရေးမယ်ဆိုရင်

---

delete(x)

---

1. Set PTR = HEAD
  2. If Head == NULL then: /\*no Node in Linked List\*/
  3.       Return false
  4. If HEAD → value == x then: /\*Found x in first Node\*/
  5.       HEAD = HEAD → Next
  6.       Free (PTR)
  7.       Return true
  8. Set Prve = PTR
  9. While PTR → Value != x and PTR != NULL then:
  10.      Prev = PTR
  11.      PTR = PTR → Next
  12. End of Loop
  13. If PTR==NULL then: /\*x not found\*/
  14.      Return false
  15. Else: /\*found in middle or Last Node\*/
  16.      Prev → Next = PTR → Next
  17.      Free (PTR)
  18.      Return true
-



## လေ့ကျင့်ကြည့်ကြရအောင်။

Linked List ထဲမှာ 40, 30, 20, 10, 55, 33, 77 ဆိုပြီး အစဉ်လိုက်ရှိနေပါတယ်။ Head ဟာ 40Node ကို ထောက်နေမှာဖြစ်ပြီး၊ 77 ကတော့ နောက်ဆုံး Node ဖြစ်မှာ ဖြစ်ပါတယ်။ အောက်ပါတို့ကို trace လိုက်ပါ။

1. delete(40)
2. delete(10)
3. delete(77)

### Inserting a Node into Simple Linked List As a Sorted Linked List

Node တစ်ခုကို simple linked list ထဲ ငယ်စဉ်ကြီးလိုက် sorted order ဖြင့်ထည့်တော့မယ် ဆိုရင် ဖြစ်နိုင်ချေ ၃ မျိုး ရှိပါတယ်။

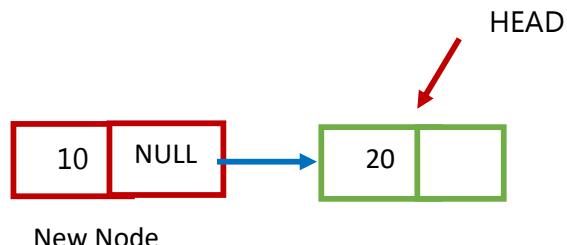
1. HEAD က NULL ဖြစ်နေတော့ ထည့်တဲ့ Node က ပထမဆုံး Node ဖြစ်သွားလို့ Head က ထောက်ရတာ။
2. အလယ်မှာသွားထည့်တာ။
3. နောက်ဆုံးမှာ သွားထည့်တာ တို့ဖြစ်ပါတယ်။
  
  
1. ရှုံးဆုံး Node အဖြစ်ထည့်တာ
  - a. HEAD ဟာ NULL ဖြစ်နေတော့ HEAD မှာထည့်မယ်။

If HEAD == NULL then:

```
HEAD = newNode
```

### b. ထည့်မည့်တန်ဖိုးက HEAD ထက်ကယ်နေတယ်။

10 ထည့်ချင်တယ် ဆိုပါတော့။ HEAD ထောက်နေတဲ့ value က 20 ဖြစ်နေတယ်။ ဒီတော့ Head ရဲ့ ရှေ့မှာသွားထည့်မယ်။



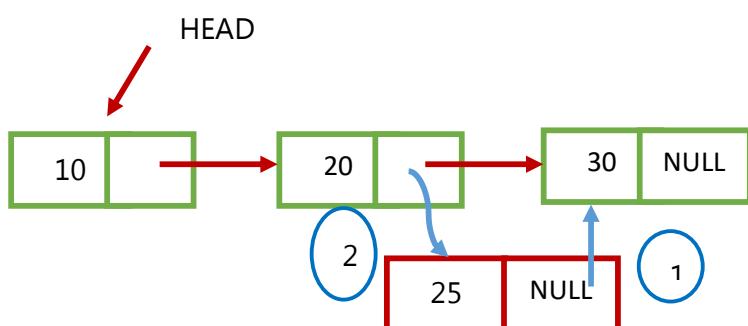
- ပထမဆုံးတော့ newNode ရဲ့ Next ဟာ NULL ဖြစ်နေတယ်။ NULL အစား Head ထောက်နေတဲ့ ကောင်ကို သွားထောက် ရမယ်။
- newNode ဟာ ပထမဆုံး Node ဖြစ်သွားလို့ Head က သွားထောက်ရပါမယ်။

Else If  $\text{newNode} \rightarrow \text{Value} < \text{HEAD} \rightarrow \text{Value}$  then:

$\text{newNode} \rightarrow \text{Next} = \text{HEAD}$

$\text{HEAD} = \text{newNode}$

### 2. အလယ်မှာ သွားထည့်တာ။



ငယ်စဉ်ကြီးလိုက် စိတဲ့ ပုံစံနှင့် ထည့်မှာ ဖြစ်လို့ Node တစ်ခု ထည့်တော့မယ်ဆိုရင် Head နေရာကနေစပီး ကြည့်ပါမယ်။ ထည့်ချင်တဲ့ newNode တန်ဖိုးထက် ငယ်နေခဲ့ရင် နောက်ကို ဆက်သွားပါမယ်။ ကြီးပြီဆိုတော့မှ အဲဒီကြီးတဲ့ Node ရဲ့ ရှေ့မှာ newNode ကို ထည့်ရမှာ

ဖြစ်ပါတယ်။ simple linked list ဟာ Next ပဲပါလို့ နောက်ကိုပဲသွားလို့ ရပါတယ်။ ရှုပြန်သွားလို့ မရတဲ့အတွက် ဒီမှာလဲ Prev node လို့မှာဖြစ်ပါတယ်။

ပုံမှာပြထားတဲ့အတိုင်း 25 ကိုထည့်ချင်တယ်၊ ပထမ Node 10ဟာ ထည့်ချင်တဲ့ 25 ထက်ငယ်တယ် ဆက်သွားမယ်၊ ဒုတိယ Node 20 ဟာ ထည့်ချင်တဲ့ 25 ထက်ငယ်တယ် ဆက်သွားမယ်၊ 30 မငယ်တော့ဘူး ရပ်မယ်။

အဲဒီအချိန်မှာ 30 က PTR ဖြစ်ပြီး၊ Prev က 20 ကို ထောက်ထားမယ်။

- newNode ရဲ့ Next ဟာ 30 ဖြစ်တဲ့ PTR ကို သွားထောက်ရမယ်။
- Prev ဖြစ်တဲ့ 20 ရဲ့ Next ဟာ newNode ကို သွားထောက်ရပါမယ်။

Set PTR = HEAD

Set Prev = PTR

Repeat while PTR != NULL and PTR → value < newNode → Value:

    Prev = PTR

    PTR = PTR → Next

End of Loop

If PTR == NULL then:

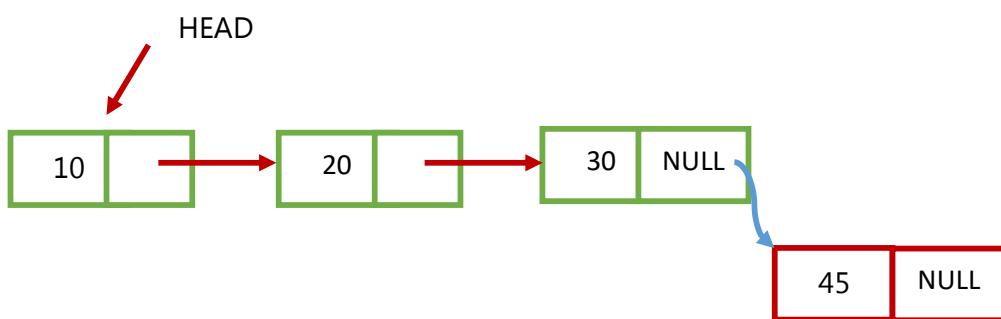
-----

Else:

    newNode → Next = PTR

    Prev → Next = newNode

### 3. နောက်ဆုံးမှာ သွားထည့်တာ



ဒီမှာဆိုရင် ထည့်ချင်တာက 45 ဆိုတော့ ရှိပြီးသား 10,20,30 Node values တွေအားလုံးထက်  
ကြီးနေတော့ PTR က သွားရင်း NULL နဲ့ညီတော့ looping ရပ်သွားတာဖြစ်ပါတယ်။ PTR NULL  
ဖြစ်သွားတဲ့ အချိန်မှာ 30 ဟာ Prev ဖြစ်နေခဲ့မှာပါ။ ဒီတော့ Prev → Next = newNode ဆိုရင်  
အဆင်ပြေသွားပါပြီ။

PTR က NULLနဲ့ညီသွားလို့ looping ထဲက ထွက်လာ တာအတွက် တန်ည်းအားဖြင့် နောက်ဆုံး  
Node အဖြစ်ထည့်တာအတွက် သွားရေးပါမယ်။

End of Loop

If PTR == NULL then:

Prev → Next = newNode

Inserting as a Sorted Linked List Algorithm ကို အစအဆုံး ပြန်ရေးမယ်ဆိုရင်

---

InsertSorted(newNode)

---

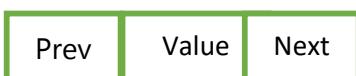
1. If HEAD == NULL then: /\*no node in linked list\*/
2.       HEAD = newNode
3. Else If newNode → Value < HEAD → Value then: /\*newNode value is < head node\*/
  4.           newNode → Next = HEAD
  5.           HEAD = newNode
6. Set PTR = HEAD
7. Set Prev = PTR
8. Repeat while PTR != NULL and PTR → value < newNode → Value:
  9.           Prev = PTR
  10.          PTR = PTR → Next
11. End of Loop
12. If PTR == NULL then: /\*newNode value is greater than values of existing nodes\*/
  13.          Prev → Next = newNode
14. Else: /\*insert newNode in middle of existing two nodes\*/
  15.          newNode → Next = PTR
  16.          Prev → Next = newNode



လေ့ကျင့်ကြည့်ကြရအောင်။

1. အပေါ်က insertSorted algorithm ကို ကြိုးစဉ်ထုတွယ်လိုက်ထည့်တာအတွက်ဆို ဘယ်နေရာတွေ  
ပြင်ရေးရမလဲ။
2. BB, CC, DD, EE, FF ရှိနေတဲ့ Sorted Linked List ထဲကို
  - a. CD ထည့်ရန် trace လိုက်ပါ။
  - b. GG ထည့်ရန် trace လိုက်ပါ။
  - c. AA ထည့်ရန် trace လိုက်ပါ။

## Doubly Linked List



Doubly Linked List မှာဆိုရင် pointer က နှစ်ခု ဖြစ်သွားတာပါ။ တစ်ခုက ရှေ့ကNode ကို  
ထောက်မည့် Prev ဖြစ်ပြီး၊ ကျန်တစ်ခုက နောက်ကNode ကိုထောက်မည့် Next ဖြစ်ပါတယ်။

- Head  
      Tail  

- 
- ရှေ့ထောက်မည့် Prev နှင့် နောက်ကိုထောက်မည့် Next ဆိုပြီး pointer နှစ်ခု ပါတာဆိုတော့ ရှေ့နောက် အပြန်ပြန် အလုန်လှန်သွားလို့ရပြီ ဖြစ်ပါတယ်။
  - နောက်ကို သွားချင်ရင် Head ကနေစမယ်။ ပြီးရင် Next ဟာ Null နဲ့ မညီမချင်းသွားလို့ရမယ်။
  - ရှေ့ကိုသွားချင်ရင် Tail ကနေစမယ်။ ပြီးရင် Prev ဟာ Null နဲ့ မညီမချင်း သွားလို့ရမယ်။

## ရှေ့ကနေ နောက်ကို ထုတ်ကြည့်ရအောင်

Set PTR = Head

Repeat while PTR != Null then:

Write PTR → Value, “ ”

PTR = PTR → Next

End of Loop

## နောက်ကနေ ရှေ့ကို ပြန်ထုတ်ချင်ရင်

Set PTR = Tail

Repeat while PTR != Null then:

Write PTR → Value, “ ”

PTR = PTR → Prev

End of Loop

Insert တို့ Delete တို့မှာလဲ Next တစ်ခုတည်းမဟုတ်တော့ပဲ Prev ထည့်ပြီးစဉ်းစားရတာလေးပဲ  
ပိုလာမှာဖြစ်ပါတယ်။

### Inserting a Node into Doubly Linked List

Node တစ်ခုကို simple linked list ထဲ ထည့်တော့မယ် ဆိုရင် ဖြစ်နိုင်ချေ 2 မျိုး ရှိပါတယ်။

1. Linked List ထဲမှာ Node တစ်ခုမှ မရှိသေးတဲ့ အခြေအနေ။
2. လက်ရှိ နောက်ဆုံး Node ကိုရှာပြီး သူရဲ့နောက်မှာ သွားထည့်တာ။ ဒါဆို အခုထည့်လိုက်တဲ့ newNode ကို Tail က ထောက်ရမယ်။

## 1. Linked List ထဲမှာ Node တစ်ခုမှ မရှိသေးတဲ့ အခြေအနေ။

Linked list ထဲမှာ Node တစ်ခုမှ မရှိသေးရင် Head ရော့ Tailပါ ထောက်စရာမရှိလို့ NULL ဖြစ်နေမှာ ဖြစ်ပါတယ်။ အသစ်ဝင်လာတဲ့ newNode သည် တစ်ခုတည်းသော Node ဖြစ်မှာ ဖြစ်ပြီး တစ်ခုတည်းရှိတာဖြစ်လို့ ရွှေ့ဆုံး Node ဆိုလည်း newNode ပါပဲ။ နောက်ဆုံး Node ဆိုလည်း newNode ပဲဖြစ်ပါတယ်။ ဒါကြောင့် Headရော့ Tail ပါ သူကို ထောက်ရမှာ ဖြစ်ပါတယ်။

If Head == NULL then:

    Head = newNode

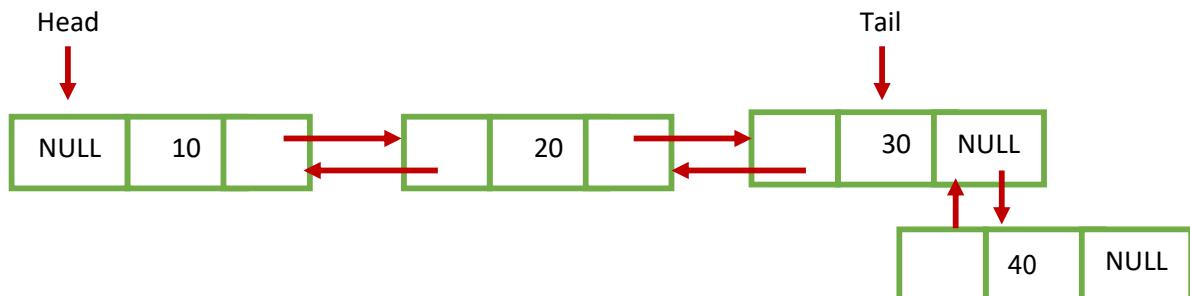
    Tail = newNode

ဒီနေရာမှာ Head == NULL အစား Tail == NULL လို့ ပြောင်းစစ်လည်း အတူတူပဲဖြစ်ပါတယ်။

ဘာလိုလဲဆိုရင် Node တစ်ခုမှ မရှိသေးတဲ့ အခြေအနေမှာ Head ရော့ Tail ပါ NULL

ဖြစ်နေမှာကြောင့် ဖြစ်ပါတယ်။

## 1. လက်ရှိ နောက်ဆုံး Node ကိုရှာပြီး သူရဲ့နောက်မှာ သွားထည့်တာ။



ပုံမှာ ပြထားတဲ့အတိုင်း newNode 40 ကို လက်ရှိ Tail ရဲ့နောက်မှာ ထည့်လိုက်ရုံးပါပဲ။ ပြီးရင် newNode

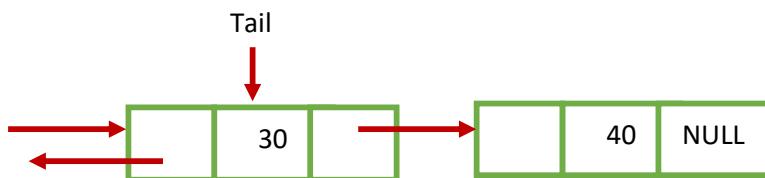
ဟာနောက်ဆုံး Node ဖြစ်သွားလို့ Tail က newNode ကို သွားထောက်ပေးရမှာ ဖြစ်ပါတယ်။

Tail → Next = newNode

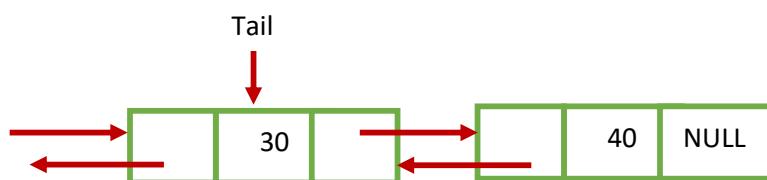
newNode → Prev = Tail

Tail = new Node

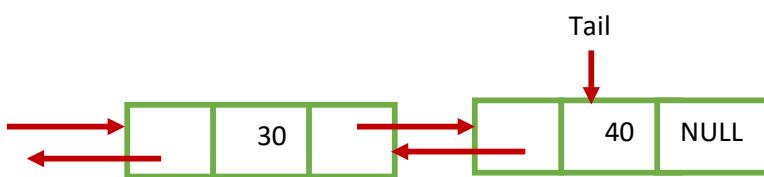
Tail → Next = newNode ဆိုတာ 30 Node ရဲ့ Next က အသစ်ဝင်လာတဲ့ 40 Node ကိုသွားထောက်တာ ဖြစ်ပါတယ်။



`newNode → Prev = Tail` ဆိုတာ အသစ်ဝင်လာတဲ့ 40 Node က 30 Node ကိုသွားပေါ်တာဖြစ်ပါတယ်။



`Tail = newNode` ဆိုတာကတော့ `newNode` ဟာ နောက်ဆုံးဖြစ်သွားလို့ Tail ကသွားပေါ်တာဖြစ်ပါတယ်။



Doubly Linked List ရဲ့ Inserting Algorithm ကို အစအဆုံး ပြန်ရေးမယ်ဆိုရင်

---

#### InsertIntoDLL(newNode)

---

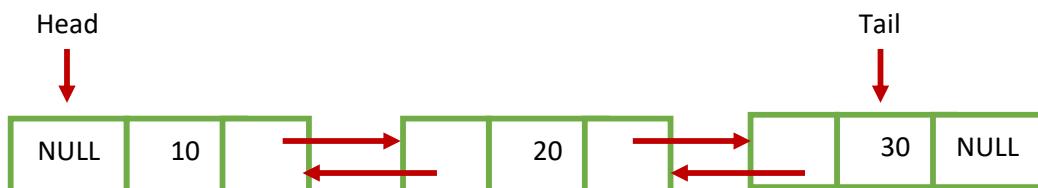
1. If Head == NULL then:
  2.     Head = newNode
  3.     Tail = newNode
  4. Else:
  5.     Tail → Next = newNode
  6.     newNode → Prev = Tail
  7.     Tail = newNode
-

## Deleting a Node from Doubly Linked List

Node တစ်ခုကို doubly linked list ထဲကနေ လိုက်ရှာပြီး ဖျက်တော့မယ် ဆိုရင် simple linked list လိုပဲ ဖြစ်နိုင်ချေ ၃ မျိုး ရှိပါတယ်။

1. ရှေ့ဆုံး Node ကိုဖျက်တာ၊
2. အလယ် Node ကို ဖျက်တာ၊
3. နောက်ဆုံး Node ကို ဖျက်တာ တို့ ဖြစ်ပါတယ်။

### 1. ရှေ့ဆုံး Node ကိုဖျက်တာ၊



- PTR ထဲကို Head ထည့်။
- Head ကို Head ရဲ့ Next (ပုံအရဆို 20 Node ကို) ရွှေ့ရမယ်။
- ရွှေ့ပြီးသား Head (20 Node) ရှေ့ဆုံးဖြစ်လာမှာဖြစ်လို့ သူ့ရွှေ့မှာ ဘာမှ မရှိတော့လို့ သူ့ရဲ့ prev ထဲကို NULL ထည့်။
- ပြီးမှ အရင် Head နေရာ ထားခဲ့တဲ့ PTR (10 Node) ကို ဖျက်ပစ်ရမှာဖြစ်ပါတယ်။

Set PTR = Head

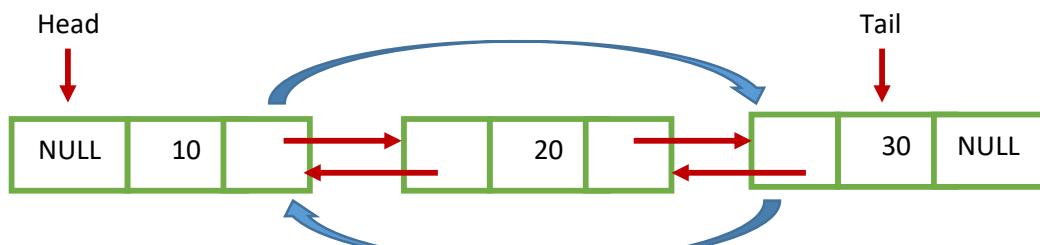
If Head → value == x then:

Head = Head → Next

Head → Prev = NULL

Free (PTR)

## 2. အလယ် Node ကိုဖျက်တာ။



**20 Node ကို ဖျက်ချင်တယ် ဆိုပါတော့။**

- ပုံအရဆိုရင် 10 Node ရဲ့ Next ဟာ 30 Node ကို သွားထောက်ပြီး၊ 30 Node ရဲ့ prev ဟာ 10 node ကို သွားပြီး ထောက်ရမှာ ဖြစ်ပါတယ်။ ပြီးမှ ဖျက်မည့် Node ကို Free လုပ်ပါမယ်။
- 10 Node ဆိုတာ ဖျက်မည့် node ရဲ့ Prev
- 30 Node ဆိုတာ ဖျက်မည့် Node ရဲ့ Next

While PTR → Value != x and PTR != NULL:

PTR = PTR → Next

End of Loop

If PTR==NULL then:

Return false

Else if PTR != Tail:

PTR → Next → Prev = PTR → Prev

PTR → Prev → Next = PTR → Next

Free (PTR)

Return true

## 2. နောက်ဆုံး Node ကို ဖျက်တာ။

နောက်ဆုံး Node ကို ဖျက်တယ်ဆိုတာ Tail Node ကို ဖျက်တာဖြစ်လို့ Tail ရဲ့ prev ဟာ Tail ဖြစ်သွားမှာ ဖြစ်ပါတယ်။

If Tail → Value == x then:

```

PTR = Tail
Tail = Tail → Prev
Free (PTR)
Return true

```

Doubly Linkde List Delete Algorithm ကို အစအဆုံး ပြန်ရေးမယ်ဆိုရင်

`deleteFromDDL(x)`

1. Set PTR = Head
2. If Head == NULL then: /\*no node to delete\*/
3.     Return false
4. If Head → value == x then: /\*delete first node in doubly linked list\*/
5.     Head = Head → Next
6.     Head → Prev = NULL
7.     Free (PTR)
8. Else if Tail → Value == x then: /\*delete last node\*/
9.     PTR = Tail
10.    Tail = Tail → Prev
11.    Free (PTR)
12.    Return true
13. While PTR → Value != x and PTR != NULL: /\*find node to delete\*/
14.     PTR = PTR → Next
15. End of Loop
16. If PTR==NULL then: /\*no found\*/
17.     Return false
18. Else: /\*delete middle node between existing two nodes\*/
19.     PTR → Next → Prev = PTR → Prev
20.     PTR → Prev → Next = PTR → Next
21.     Free (PTR)
22.     Return true



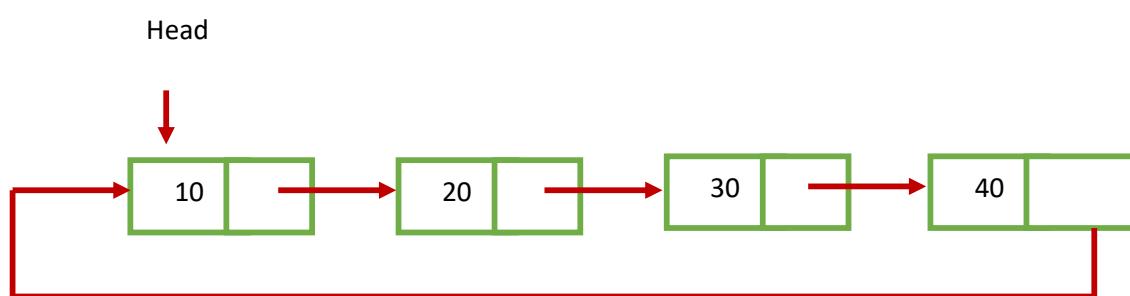
## လေ့ကျင့်ကြည့်ကြရအောင်။

1. Simple Linked List ရဲ sorted ပုံစံနဲ့ထည့်တာကို ကြည့်ပြီး doubly linked list အတွက် sorted ပုံစံနဲ့ ထည့်တာကို ပြန်ရေးပေးပါ။ ကြီးစဉ်ယ်လိုက်ထည့်တာရယ်၊ ငယ်စဉ်ကြီးလိုက်ထည့်တာရယ်အတွက် algorithm ၂ ခုရေးပေးရမှာဖြစ်ပါတယ်။
2. Doubly Linked List မှာ head, tail ဆိုပြီး pointer ၂ခုကို အသုံးပြုပြီး အလုပ်လုပ်သွားတာဖြစ်ပါတယ်။ head pointer တစ်ခုပဲ အသုံးပြုပြီး အလုပ်လုပ်လို့ ရပါတယ်။ ဒါကြောင့် doubly linked list အတွက် algorithms တွေကို head pointer တစ်ခုပဲအသုံးပြုပြီး ပြန်ရေးပေးပါ။ ပြီးရင် pointer 2 ခုသုံးတာ နဲ့ pointer 1 ခုသုံးတာ ဘယ်ဟာက သုံးရအဆင်ပြေသလဲဖြေပါ။

## Circular Linked List

Circular Linked List ကတော့ ပိုလာတာ တစ်ခုလေးပဲ ရှိပါတယ်။ နောက်ဆုံး Node ကနေပြီး ရှုံးဆုံး Node ကို ပြန်ထောက်တာ ဖြစ်ပါတယ်။ ဒါကြောင့် Circular Linked List မှာ နောက်ဆုံး Node လား စစ်ချင်ရင် အရင်လို့ Next က NULL နဲ့ညီတာအစား Next က Head နဲ့ ညီတာပြောင်းစစ်ရမှာ ဖြစ်ပါတယ်။

နောက်တစ်ခုကတော့ နောက်ဆုံးမှာ newNode ကိုသွားထည့်ရင် newNode ရဲ့ Next ထဲကို Head ပြန်ထည့်ပေးခဲ့ရမှာ ဖြစ်ပါတယ်။ အလယ်က Node တွေ၊ အရှုံးဆုံး Node အလုပ်လုပ်ပုံတွေဟာ ကွားခြားချက်မရှိပါဘူး။





## လေ့ကျင့်ကြည့်ကြရအောင်။

Simple Linked List နဲ့ doubly linked list ကို သေချာပြောပြီးသွားပြီ၊ ပြီးတော့ circular linked list ရဲ့ ကွာခြားချက်ကိုလည်း ပြောခဲ့ပြီးပြီ ဖြစ်တာကြောင့် traversing, insert, delete, insertSorted စတဲ့ algorithms တွေကို circular linked list အတွက် ကြိုးစားပြီး ရေးကြည့်ပေးပါ။



## Chapter အနှစ်ချုပ်

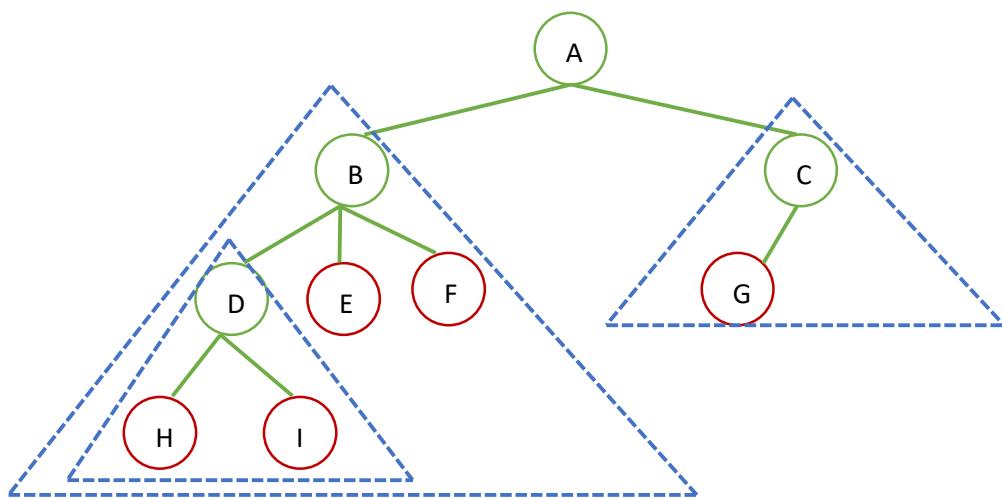
- ဒီအခန်းမှာ ပြောသွားတဲ့ linked list ဆိုတာ လဲ array ကို လိုအပ်ချက်အရ ပြောင်းသုံးတဲ့ ပုံစံတစ်မျိုး ဖြစ်ပါတယ်။
- ဒါကြောင့် Array ကို ပုံစံမျိုးစုံပြောင်းသုံးတဲ့ အထဲက Stack, Queue, Hashing, Linked List အကြောင်းတွေ ပြောခဲ့ပြီးပြီ ဖြစ်ပါတယ်။
- အဲဒါတွေအားလုံးဟာ ဒီနောက်ကပိုင်း programming language တွေမှာ language က ကြိုတင်ပြီး implement လုပ်ထားပေးပြီး (ကြိုတင်ရေးသားထားပေးပြီး) ဖြစ်ပါတယ်။
- ဒါကြောင့် programmer က ရေးစရာမလိုတော့ပဲ language က ကြိုရေးထားတာကို အလွယ်တကူ ယူသုံးလို့ရပါတယ်။
- သို့သော် အကျိုးအပြစ် မသိပဲ တွေ့ရာလျောက်သုံးရင် သေချာပေါက် run time မှာ သေချာပေါက် စကား ပြောမှာ ဖြစ်ပါတယ်။
- Linked List ကိုပဲ ကြည့်ကြည့်ပါ၊ တစ်ခုခုထည့်ချင်တာဖြစ်ဖြစ်၊ တစ်ခုခု ဖျက်ချင်တာဖြစ်ဖြစ် next စတဲ့ link တွေပြောင်းချိတ်နေရတာနဲ့ အချိန်ယူနေတာဖြစ်ပါတယ်။ ရှိုးရိုး array မှာတော့ အဲဒီလို လုပ်ဖို့မလိုအပ်လို့ အချိန်ကုန်သက်သာပါတယ်။

## အခန်း (၁၆)

### Tree

#### Tree or Hierarchical Structure

Tree ဆိတာ အဆင့်ဆင့် ခွဲခွဲသွားတဲ့ Hierarchical ပုံစံနဲ့ သိမ်းချင်တဲ့ အခါမှာ သုံးတယ်ဆိတာ ရေ့မှာ ပြောခဲ့ ပြီးပါပြီ။

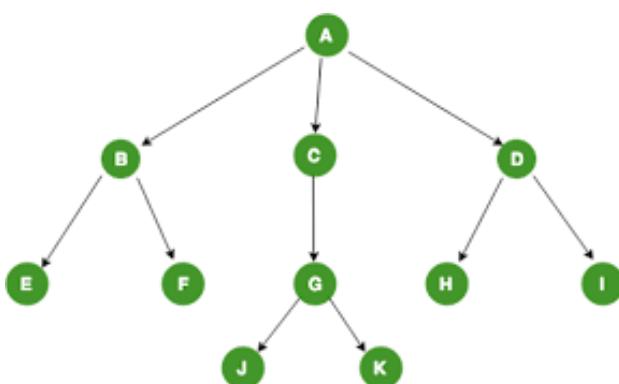


- **Node:** ဒီမှာ ပြထားတဲ့ အပိုင်းလေး တစ်ခုချင်းစီကို Node လိုက်ပါတယ်။
- **Key:** အဲဒီ Node ထဲမှာပါတဲ့ A,B,C ဆိတာလေးတွေကိုတော့ key လိုက်ပါတယ်။
- ဒါကြောင့် Node တစ်ခုမှာ Key ရယ်၊ ပြီးတော့ သူအောက်က Node တွေကို ပြန်ထောက်မည့် Node pointer တွေရယ် ပါမှာဖြစ်ပါတယ်။
- **Parent and children:** A ဆိတဲ့ Node အောက်မှာ B နဲ့ C ဆိုပြီး Node နှစ်ခု ရှိပါတယ်။ ဒါကြောင့် A ကို B နဲ့ C ရဲ့ parent လိုက်ပြောပြီး၊ အပြန်အလှန်အနေနဲ့ B နဲ့ C ကို A ရဲ့ child တွေလို့ ပြောပါတယ်။
- **Siblings:** B နဲ့ C ဟာ parent တဲ့လို့ သူတို့ကို siblings လို့ ခေါ်ပါတယ်။

- **Edge:** Node တစ်ခု နဲ့ တစ်ခြား node တစ်ခု အကြေားဆက်ထားတဲ့ line လေးကို edge လို့ ခေါ်ပါတယ်။
- **Path:** A ကနေ H ကိုသွားမယ်ဆိုရင် ABDH ဆိုပြီး သွားရမှာဖြစ်ပါတယ်။ AB, BD, DH ဆိုတဲ့ edge လေးတွေဆက်ထားတာ ဖြစ်ပါတယ်။ အဲဒီလို့ edge လေးတွေဆက်ထားတာကို path လို့ ခေါ်ပါတယ်။
- **Root:** parent မရှိတဲ့ node ကို root node လို့ ခေါ်ပါတယ်။ ပုံမှာဆိုရင်တော့ A node ဖြစ်ပါတယ်။
- **Leaf:**ထပ်ခဲ့ထားတဲ့ child တစ်ယောက်မှ မရှိတဲ့ node တွေကို တော့ leaf node လို့ ခေါ်ပါတယ်။ ပြထားတဲ့ ပုံမှာတော့ E, F, G, H, I တို့ ဖြစ်ပါတယ်။
- **Subtree:** Tree ဆိုတာ subtree လေးတွေနဲ့ဖွံ့စည်းထားတာ ဖြစ်ပါတယ်။ ဒီမှာ ကြိုက်ပုံစံ လေးတွေ ပိုင်းပြထားတာ လေးကို subtree လို့ ခေါ်ပါတယ်။ subtree တစ်ခုချင်းမှာလဲ root node နဲ့ child node တွေ ပါမှာဖြစ်ပါတယ်။
- **Level:** Level သတ်မှတ်တဲ့ အခါမှာ Root node ကို Level 0 လို့ သတ်မှတ်ပြီး အောက်ရောက်လေ level တိုးတိုးသွားလေ ဖြစ်ပါတယ်။ ပြထားတဲ့ ပုံမှာဆိုရင် B နှင့် C ဆိုရင် level 1၊ D, E, F, G ဆိုရင် level 2 ဖြစ်ပါမယ်။



လေ့ကျင့်ကြည့်ကြရအောင်။



ပုံကိုကြည့်ပြီး အောက်ပါမေးခွန်းတွေကို ဖြေပါ။

1. Node ဘယ်နှစ်ခုရှိပါသလဲ။
2. Edge ဘယ်နှစ်ခု ရှိပါသလဲ။
3. Root Node ကိုရှာပါ။
4. Node A ရဲ့ child တွေကို ချရေးပေးပါ။
5. J နှင့် K ရဲ့ parent ကိုရှာပါ။
6. C ရဲ့ siblings ကို ရှာပါ။
7. Level ဘယ်နှစ်ခုရှိပါသလဲ။
8. J နှင့် K ရဲ့ level က ဘယ်လောက်ပါလဲ။
9. Leaf node တွေကို ရှာပါ။
10. A ကနေ J ကိုသွားလို့ရမည့် path ကိုရှာပါ။ အဲဒီ path ထဲမှာ Edge ဘယ်နှစ်ခု ပါပါသလဲ။
11. Subtree များကိုရှာပြီး ဆွဲပြုပါ။

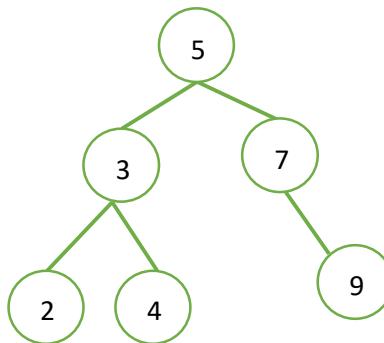
## Binary Tree

Binary tree မှာ

- Node တွေဟာ left child နဲ့ right child ဆိုပြီး အများဆုံး child ၂ ခုပဲရှိရပါမယ်။ အများဆုံးလို့ပြောထားတာဖြစ်လို့ node တွေဟာ child မရှိလည်း ရတယ်၊ child ၁ ခုပဲရှိလည်း ရတယ်၊ child ၂ခုရှိလည်း ရတယ်၊ child ၂ ခုထက်တော့ ကျော်လို့မရပါဘူး။
- Left child ရဲ့ key ဟာ parent node ရဲ့ key ထက်ငယ်ရပါမယ်။
- Right child ရဲ့ key ဟာ parent node ရဲ့ key ထက် ကြီးရပါမယ်။

| Left Child | Key | Right Child |
|------------|-----|-------------|
|------------|-----|-------------|

Node ဟာ tree မှာ ပြတဲ့အချိန်မှာ အထိုင်းလေးတွေနဲ့ ပြပေမဲ့ နောက်ကွယ်က အလုပ်လုပ်တဲ့ အခါမှာတော့ ယခုပြထားတဲ့ ပုံအတိုင်း အလုပ်လုပ်သွားမှာ ဖြစ်ပါတယ်။ ဒါကြောင့် Node တစ်ခုမှာ keyရယ်၊ left childရယ်၊ right child ရယ်ဆိုပြီး ပါမှာဖြစ်ပါတယ်။ ထုံးစံအတိုင်း key ကတော့ ကိုယ်ကြိုက်တဲ့ data type ထားလိုက်ပါတယ်။ Left child နဲ့ right child ကတော့ node တွေကို ထောက်မှာ ဖြစ်လို့ node pointer ဖြစ်မှာ ဖြစ်ပါတယ်။ left child ရှိခဲ့ရင် left child ထဲကို left child ဖြစ်တဲ့ node ကိုသိမ်းပြီး၊ မရှိခဲ့ရင် left child ထဲကို NULL ထည့်မှာ ဖြစ်ပါတယ်။ right child ကလည်း အလားတူပဲဖြစ်ပါတယ်။ အောက်က ပုံကို နမူနာကြည့်ပါ။



- 5 node ဟာ root node ဖြစ်ပါတယ်။ left child ရဲ့ key ဖြစ်တဲ့ 3 ဟာ သူ့ရဲ့ key ထက်ငယ်ပြီး၊ right childရဲ့ key ဖြစ်တဲ့ 7 ဟာ သူ့ထက်ကြီးပါတယ်။
- 3 node မှာလည်း left childရဲ့ key ဖြစ်တဲ့ 2 ဟာ သူ့ထက်ငယ်ပြီး၊ right child ရဲ့ key ဖြစ်တဲ့ 4 ဟာ သူ့ထက်ကြီးပါတယ်။
- 7 node မှာတော့ right child တစ်ခုပဲရှိပါတယ်။ right child ဖြစ်တဲ့ အတွက် သူ့ထက်ကြီးပါတယ်။ left child မရှိတဲ့ အတွက် left child တန်ဖိုးဟာ NULL ဖြစ်နေမှာ ဖြစ်ပါတယ်။

ဒါတော့ Binary tree မှာ ရှာချင်တာပဲဖြစ်ဖြစ်၊ ထည့်ချင်တာပဲဖြစ်ဖြစ်၊ ဖျက်ချင်တာပဲဖြစ်ဖြစ် left child ဟာ parent ထက်ငယ်တယ်၊ right child ဟာ parent ထက်ကြီးရမယ်ဆိုတာ ခေါင်းထဲ ထည့်ထားဖို့လိုပါတယ်။

## Finding a Node in Binary Tree

တန်ဖိုးလေး တစ်ခုပေးပြီး Binary tree ထဲကနေ အဲဒီ တန်ဖိုးနဲ့ ညီတဲ့ key ရှိတဲ့ node ကို လိုက်ပြီးရှာပါမယ်။

- ထိပ်ဆုံး node ဖြစ်တဲ့ root node ကနေစပြီး ရှာပါမယ်။
- ငယ်ရင် left child ကိုသွားမယ်။
- ကြီးရင် right child ကို သွားမယ်။
- ညီရင် တွေ့လို ညီတဲ့ node ကို return ပြန်ပေးမယ်။
- သွားရင်း NULL နဲ့ညီသွားရင် မတွေ့တော့တာဖြစ်လို မတွေ့ကြောင်း NULL return ပြန်ပါမယ်။

### Find (x)

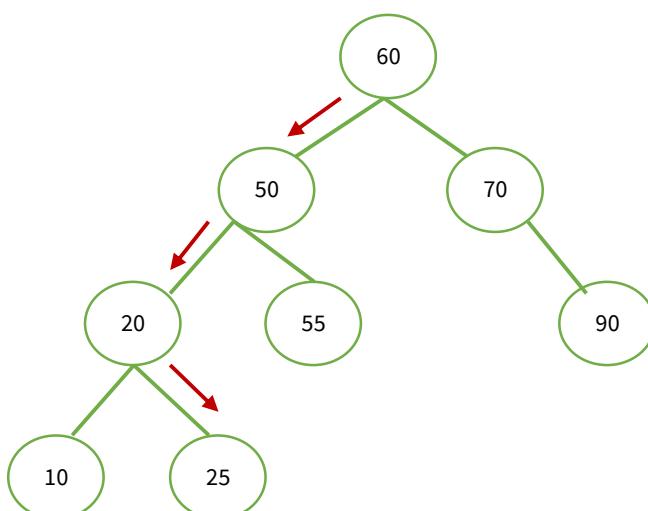
1. If root == NULL then: /\*no node in tree\*/
2.     Return NULL
3.     Set current = root
4.     Repeat while current != NULL and current → key != x:
5.         If x < current → key then:
6.             current = current → leftChild
7.         Else:
8.             current = current → rightChild
9.     end of loop
10.    Return current

Root node ကနေစပြီး ရှာမှာဖြစ်လို line 3 မှာ current ထဲကို root ထည့်လိုက်ပါတယ်။ current ဆိုတာကတော့ Linked list အခန်းက PTR လိုပဲ လိုရာကိုသွားဖို့ သုံးတာဖြစ်ပါတယ်။

လိုက်ရှာတာ ဖြစ်တဲ့အတွက် မတွေ့တာလဲ ဖြစ်နိုင်တယ်၊ မတွေ့တော့ရင် NULL နဲ့ ညီသွားမယ်။ ဒါကြောင့် line 4 ရဲ့ while မှာ မတွေ့တော့လို့ NULL နဲ့ ညီရင်လည်း ထွက်၊ သို့မဟုတ် ဝင်လာတဲ့ x နဲ့ current node ရဲ့ key နဲ့ ညီသွားလည်း ထွက်ဆိုပြီး ရေးထားပါတယ်။ ဒီတော့

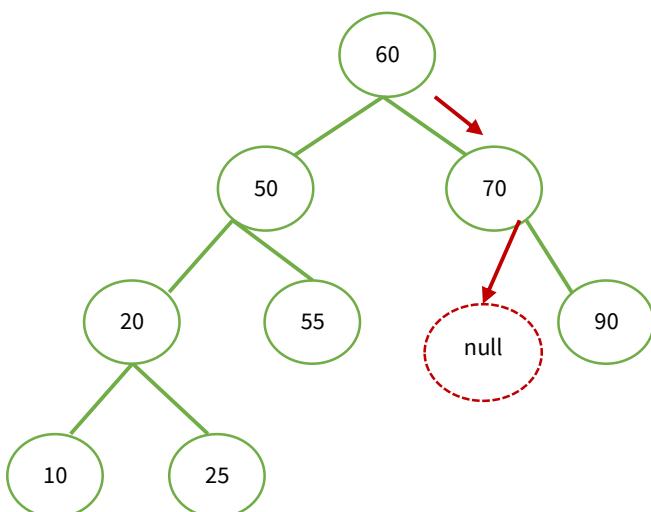
- Looping ထဲကို ရောက်လာတယ်ဆိုတာ မညီလို့ရောက်လာတာ။ မညီတော့ ဖြစ်နိုင်ချေ ၂ ခုပဲရှိတယ်။ ငယ်တာနဲ့ ကြီးတာ။ ငယ်ရင် current ကို current ရဲ့ left child ကိုရွှေ့၍ ကြီးရင် current ကို current ရဲ့ right child ကို ရွှေ့ဆိုပြီး looping body ထဲ ရေးထားတာ ဖြစ်ပါတယ်။
- Looping ထဲကထွက်တယ်ဆိုတာလည်း ဖြစ်နိုင်ချေနှစ်ခုပဲ ရှိတယ်။ current ရဲ့ key ဟာ x နဲ့ ညီသွားလို့ သို့မဟုတ် current ဟာသွားရင်း သွားရင်း ညီတာမတွေ့တော့ဘဲ NULL ရောက်သွားတာ။
  1. ဒါဆို Current ဟာ NULL နဲ့ ညီရင် မတွေ့တော့တာဖြစ်လို့ NULL return ပြန်မယ်။  
Current က NULL ဖြစ်နေတဲ့ အချိန်မှာ NULL ကို return ပြန်တာကြောင့် return current ရေးလိုက်ရင် ရပါပြီ။
  2. ထိုအတူပဲ current → key ဟာ x နဲ့ ညီလို့ ထွက်လာတာဆိုရင်လည်း current မှာ တွေ့တာဖြစ်တဲ့အတွက် return current ဆိုပြီးရေးရမှာဖြစ်ပါတယ်။

အခြေနေ နှစ်ခုလုံး လုပ်ရမှာ တူနေတဲ့အတွက် looping အပြင်မှာ condition စစ်စရာ မလိုအပ်တော့ဘဲ return current ဆိုပြီး တစ်ကြောင်းတည်း ရေးလိုက်တာဖြစ်ပါတယ်။



ပြထားတဲ့ Tree ထဲကနေ 25 ကိုရှုတာလေးကို arrow လေးတွေနဲ့ trace လိုက်ပြထားပါတယ်။ ရှာချင်တာက 25!

- Current ထဲကို root node ထည့်တယ်။
- 60 ထက် 25 ကငယ်တယ် left child ကိုသွားမယ်။
- 50 ကိုရောက်တယ်။ 50 ထက် 25 ကငယ်တယ်။ left child ကိုသွားမယ်။
- 20 ကိုရောက်တယ်။ 20 ထက် 25 ကကြီးတယ်။ right child ကိုသွားမယ်။
- 25 ကိုရောက်တယ်။ ညီသွားပြီဖြစ်လို့ အဲဒီ node ကို return ပြန်လိုက်တယ်။



အခုပြထားတာကတော့ tree ထဲကနေ 67 ကိုရှာချင်တာဖြစ်ပါတယ်။

- Current ဟာ root node ကနေစတယ်။
- 67 ဆိုတော့ current ရောက်နေတဲ့ 60 ထက်ကြီးတယ်၊ right child ကိုသွားမယ်။
- 70 ကိုရောက်တယ်။ ရှာချင်တဲ့ 67 က 70 ထက်ငယ်တော့ left child ကိုသွားမယ်။
- 70 မှာ left child မရှိတော့ left child နေရာ NULL ဖြစ်နေမယ်။ ဒါတော့ 70 ရဲ့ left child ကိုသွားတဲ့ current ဟာ NULL နဲ့ ညီသွားမယ်။ မတွေ့ကြောင်း NULL ကို return ပြန်ပေးလိုက်မယ်။



## လေ့ကျင့်ကြည့်ကြရအောင်။

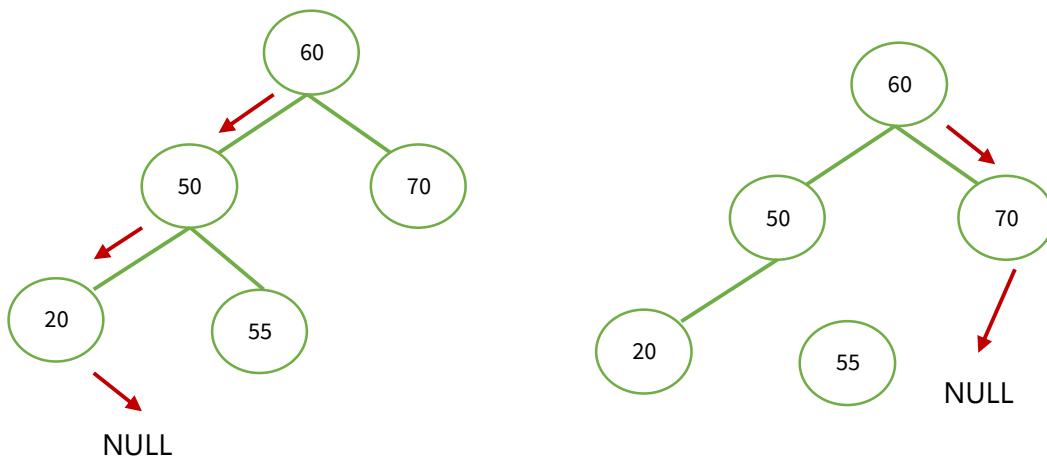
1. 11, 17, 4, 34, 16, 8, 2, 40 အစဉ်လိုက်ဝင်လာတာကို ဘယ်လိုပုံစံထွက်မလဲ binary tree ဆွဲကြည့်ပါ။ Left child ဟာ parent ထက်ငယ်ရမယ်၊ right child ဟာ parent ထက်ကြီးရမယ် ဆိုတာကို မမေ့ပါနဲ့။
  2. ဆွဲပြီးသား tree ထဲမှ 2 ကိုရှာတာကို ပုံဆွဲပြီး trace လိုက်ပြပါ။
  3. ဆွဲပြီးသား tree ထဲမှ 40 ကိုရှာတာကို ပုံဆွဲပြီး trace လိုက်ပြပါ။
  4. ဆွဲပြီးသား tree ထဲမှ 45 ကိုရှာတာကို ပုံဆွဲပြီး trace လိုက်ပြပါ။
- ဒီလေ့ကျင့်ခန်းလုပ်ပြီးမှ အောက်က ခေါင်းစဉ်ကို ဆက်ဖတ်ပါ။ မပြီးပဲ ဆက်မဖတ်ပါနဲ့။

### Inserting a Node into Binary Tree

- Root ဟာ NULL နဲ့သို့နေရင် node တစ်ခုမှ မရှိသေးလို့ အခုဝင်လာတဲ့ newNode ဟာ root node ဖြစ်မှာ ဖြစ်ပါတယ်။

If root == NULL then:

```
root = newNode
```



ပထမပုံကတော့ 25 ထည့်ချင်တာအတွက် ဆွဲပြထားတာ ဖြစ်ပါတယ်။

- ထုံးစံအတိုင်း current ဟာ root node ကနေပြီး စမယ်။
- ထည့်ချင်တာက 25 ဆိုတော့ current ရောက်နေတဲ့ 60 ထက်ငယ်လို့ current ကို left child ကို ရွှေ့မယ်။
- 50 တွေ့တယ်။ နေရာမလွတ်သေးဘူး။ 25 ဟာ 50 ထက်ငယ်တယ်။ ဒါတော့ current ကို left child ကို ရွှေ့မယ်။
- 20 တွေ့တယ်။ နေရာမလွတ်သေးဘူး။ 25 ဟာ 20 ထက်ကြီးတယ်။ ဒါတော့ current ကို right child ကိုရွှေ့မယ်။
- Current ဟာ NULL ဖြစ်သွားတယ်။ NULL ဖြစ်သွားတယ် ဆိုတာ နေရာလွတ်သွားတာ။ အဲဒါတော့ အဲဒါနေရာမှာ ထည့်ရမှာဖြစ်ပါတယ်။ right child သွားရင်း NULL နဲ့ ဖြစ်သွားတာ ဖြစ်လို့ NULL မရောက်ခင် 20 node ရဲ့ right child အနေနဲ့ newNode(25) ကို ထည့်ရမှာဖြစ်ပါတယ်။ သို့သော် current ဟာ NULL ရောက်မှု NULL မရောက်ခင် သူနောက်ဆုံးသွားခဲ့တဲ့ 20 ကို ပြန်တက်လို့ မရပါဘူး။ ဒါတော့ ထုံးစံအတိုင်း Prev ဆိုတဲ့ node တစ်ခု ကိုထားမှာဖြစ်ပါတယ်။ current နောက်ရွှေ့မယ်လုပ်တိုင်း current မရွှေ့ခင် current နေရာကို Prev ကို ထားခဲ့ပါတယ်။ ဒါတော့မှ current လဲ NULL ရောက်ရော့ prev က 20 node မှာ ကျွန်ုန်းနဲ့မှာဖြစ်ပါတယ်။

ဒုတိယ ပုံကတော့ 67 ထည့်မှာအတွက် ဆဲပြထားပါတယ်။

- ထုံးစံအတိုင်း current ဟာ root node ကနေပြီး စမယ်။
- ထည့်ချင်တာက 67 ဆိုတော့ current ရောက်နေတဲ့ 60 ထက်ကြီးလို့ current ကို right child ကို ရွှေ့မယ်။ current ကို right child မရွှေ့ခင် Prev ကို current နေရာ ထားခဲ့ပါမယ်။ ဒါတော့ Prev ဟာ 60 မှာကျွန်ုန်းနဲ့ပြီး current က 70 ကိုရောက်သွားမယ်။

- 70 တွေ့တယ်။ နေရာမလွတ်သေးဘူး။ 67 ဟာ 70 ထက်ငယ်တယ်။ ဒီတော့ current ကို left child ကို ရွှေ့မယ်။ current မရွှေ့ခင် current နေရာကို Prev ကို ပေးခဲ့မယ်။ Prev ဟာ 70 မှာ ကျွန်ုင်နေခဲ့မယ်။ current ကို left child ရွှေ့တော့ current ဟာ NULL ဖြစ်သွားမယ်။
- Current NULL ဖြစ်သွားတာဟာ နေရာလွတ်တာဖြစ်ပါတယ်။ Left child သွားရင်း NULL ဖြစ်တာကြောင့် Prev ရဲ့ left child အနေနဲ့ထည့်မှာဖြစ်ပါတယ်။

အချုပ်ပြန်ပြောရရင် current သည် root ကနေစမယ်။ နေရာလွတ်မလွတ်သိရအောင် current ကို NULL နဲ့ညီတဲ့အထိ ရွှေ့မယ်။ current မရွှေ့ခင်မှာ current နေရာကို Prev ကို ပေးခဲ့မယ်။ left child ရွှေ့ရင် NULL နဲ့ညီသွားရင် newNode ကို Prev ရဲ့ left child အနေနဲ့ထည့်မယ်။ right child ကိုသွားရင် NULL နဲ့ညီတာဆိုရင် newNode ကို Prev ရဲ့ right child အနေနဲ့ထည့်မယ်။ ဒါကြောင့် left child ကို သွားတာလား၊ right child ကိုသွားတာလားဆိုတာ မှတ်ထားဖို့ လိုတယ်။ အဲဒီအတွက် isLeftChild ဆိုတဲ့ variable ကို အသုံးပြုပါမယ်။ left child ကိုသွားရင် isLeftChild ထဲ true ထည့်ပြီး right child ကိုသွားရင် isLeftChild ထဲ false ထည့်မှာဖြစ်ပါတယ်။

---

Insert ( newNode )

---

If root == NULL then:

root = newNode

Set current = root

Set Prev = NULL

Set isLeftChild = true

Repeat while current != NULL:

Prev = current

If newNode → key < current → Key then:

current = current → leftChild

isLeftChild = true

```

Else:
 current = current → rightChild
 isLeftChild = false

End of Loop

If isLeftChild then:
 Prev → leftChild = newNode
Else:
 Prev → rightChild = newNode

```

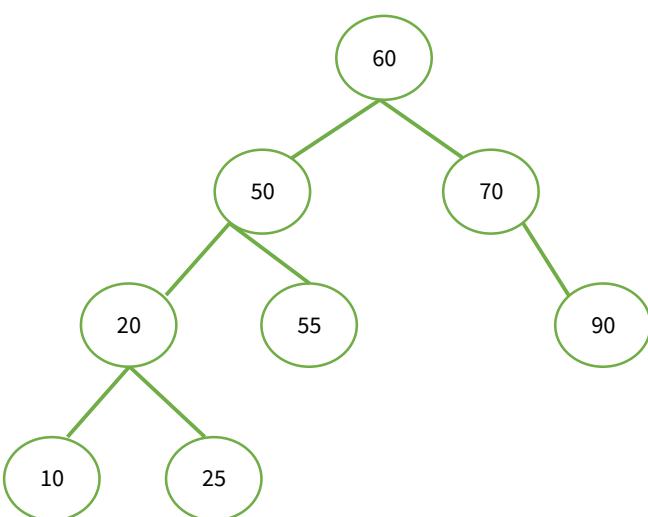
---

Set isLeftChild = true

ဆိတ္တဲ့ စာကြောင်းဟာ ဒီ variable ကို အသုံးပြုမည့်အကြောင်း သိဖို့သာ ဖြစ်ပါတယ်။ true ထည့်ထည့်  
false ထည့်ထည့် အရေးမပါလှပါဘူး။ ဘာကြောင့်လဲဆိုရင် အဲဒီတန်ဖိုးတွေဟာ while loop ထဲမှာ  
ပြောင်းထည့်လိုက်မှာဖြစ်တာကြောင့် ဖြစ်ပါတယ်။



လေ့ကျင့်ကြည့်ကြရအောင်။



Algorithm ကို trace လိုက်ပြီး tree ထဲကို 100, 15, 52 တို့ကို ထည့်ပါ။

## Traversing a Binary Tree

Traversing ဆိုတာတော့ linked list နဲ့ တင် သိခဲ့လောက်ပါပြီ။ Node ပေါက်စွဲရောက်အောင်သွားတာဖြစ်ပါတယ်။ Tree မှာတော့သွားတဲ့နည်း ၃ နည်း ရှိပါတယ်။

1. Pre-order (Root, L, R)
2. In-order (L, Root, R)
3. Post-order (L, R, Root)

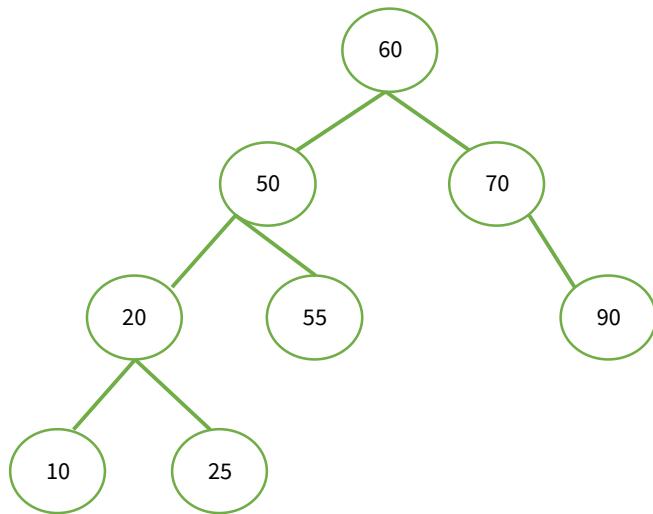
တို့ပဲဖြစ်ပါတယ်။

Pre ဆိုတာကြိုတင်လုပ်တဲ့ အဓိပ္ပာယ်ဖြစ်တဲ့ အတွက် Root ကို ရှုံထား၊ ပြီးမှ Left, Right ဆိုပြီး သွားမှာဖြစ်ပြီး၊ in ဆိုတာ အလယ် ဆိုတဲ့ အဓိပ္ပာယ် ဖြစ်တာကြောင့် Left, Right ကြားမှာ Root ဆိုပြီး (Left, Root, Right) သွားမှာဖြစ်ပါတယ်။ Post ကတော့ နောက်ဆိုတဲ့ အဓိပ္ပာယ်ဖြစ်လို့ Left, Right, Root ဆိုပြီး သွားပါမယ်။ Root က နောက်ဆုံးလုပ်မှာ ဖြစ်ပါတယ်။

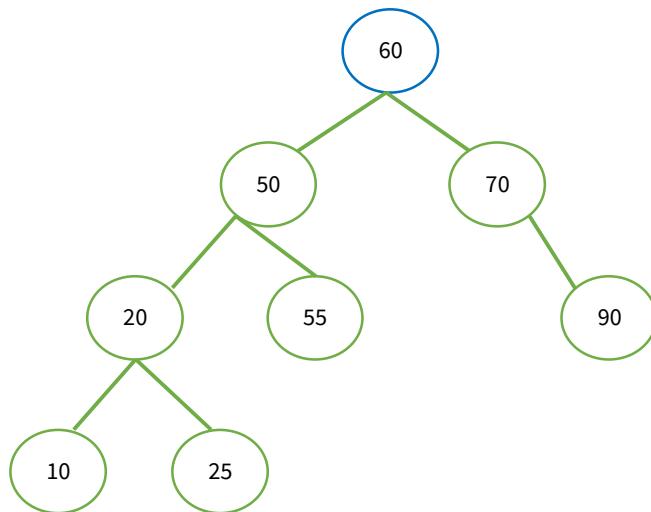
လူတစ်ယောက် တနေ့တည်း သွားစရာနေရာ သုံးလေးခု ရှိတယ်ဆိုပါတော့။ ဘယ်နေရာကို အရှင်သွားမလဲဆိုတာတော့ အဲဒီလူရဲ့ ဆုံးဖြတ်ချက်ပေါ်မူတည်ပါတယ်။ ဒီလိုပဲ ဒီမှာလ pre-order, in-order, post-order ဆိုပြီး ရှိတဲ့ အထဲက လိုအပ်ချက်ပေါ်မှာ မူတည်ပြီး ကိုယ်ကြိုက်တဲ့ နည်းနဲ့ သွားလို့ရပါတယ်။

## Pre-order Traversing

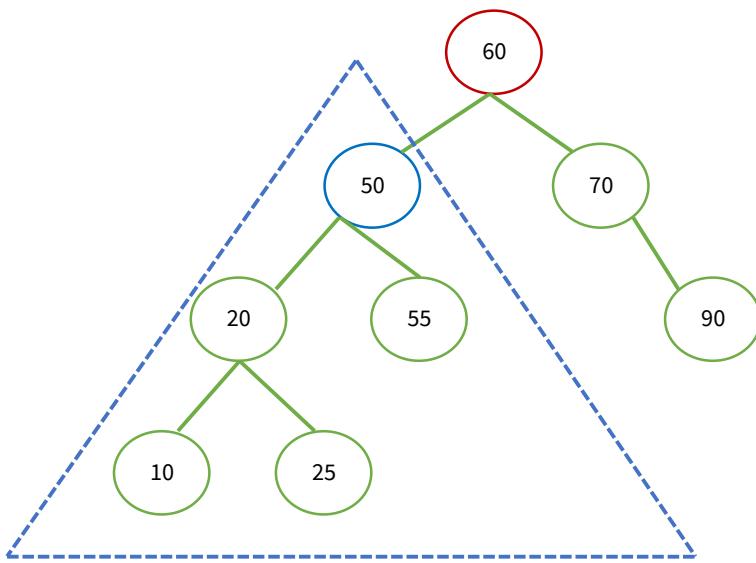
Pre-order ဆိုတော့ Root ကို အရင် output ထုတ်ပြုမယ်။ ပြီးမှာ Left child, Right Child လာမှာဖြစ်ပါတယ်။



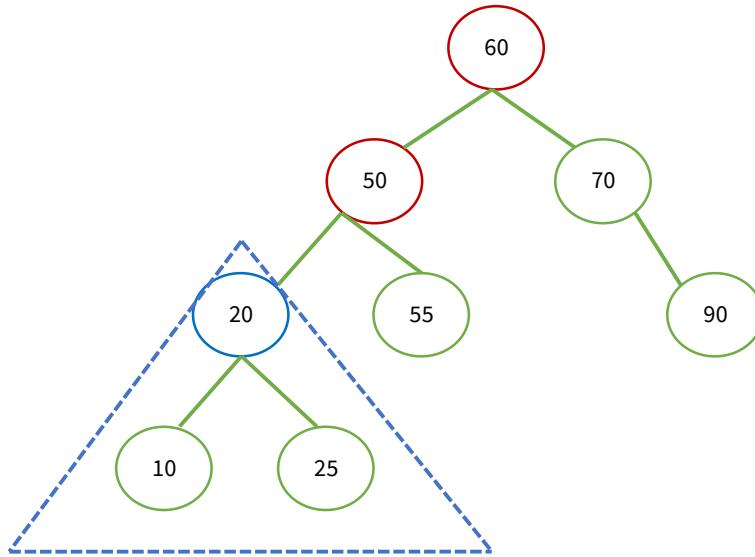
ဒီပဲကို pre-order နဲ့ output ထုတ်ကြည့်ပါမယ်။



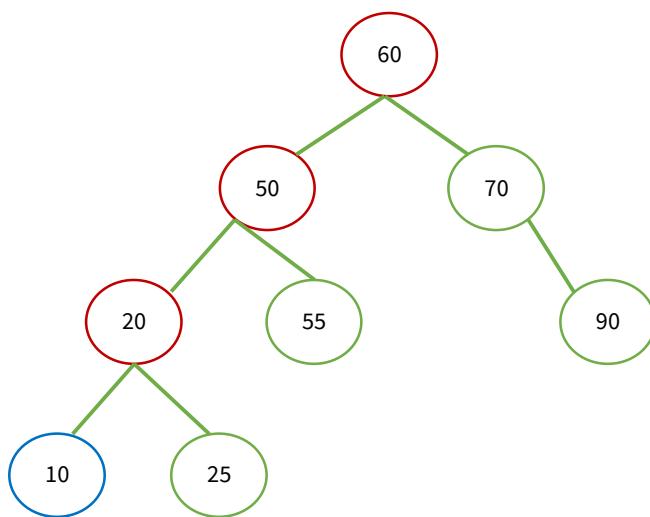
Root, Left, Right ဆိုတော့ Root ဖြစ်တဲ့ 60 ကို အရင် output ထုတ်လိုက်ပြုမယ်။ Root ပြီးရင် Left ဆိုတဲ့ အတွက် Left child 50 ကိုသွားမှာ ဖြစ်ပါတယ်။



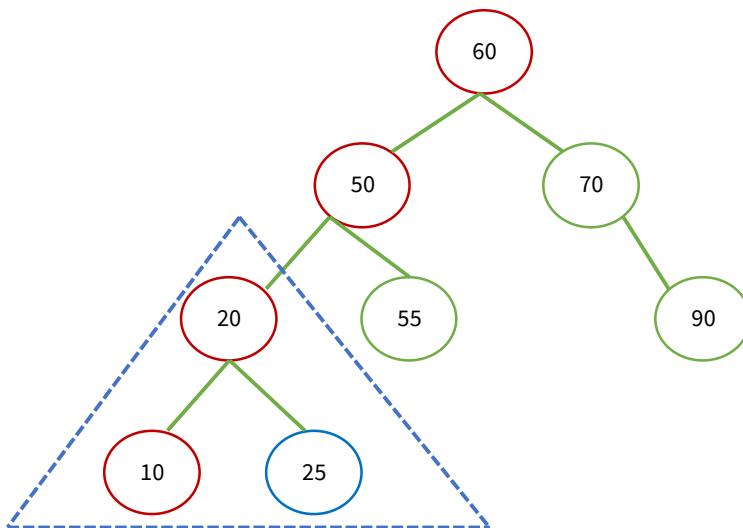
50 ကို **ပြန်ကြည့်လိုက်တော့ ဖြေားပါမယ်။** ဒိုင်းပြထားတဲ့ subtree လေးရဲ့ root ပြန်ဖြစ်နေတယ်။ Root အရင် လုပ်မှုသိတဲ့အတွက် 50 ကို output ထုတ်လိုက်တယ်။ Root ပြီးတော့ Left ဆိုတော့ left child 20 ကိုသွားပါမယ်။



**မြင်တဲ့အတိုင်းပါပဲ။** 20 ဟာ အခုပြထားတဲ့ subtree ရဲ့ root ဖြစ်နေတဲ့အတွက် root အဖြစ် 20 ကို အရင်ထုတ်လိုက်ပါမယ်။ Root အနေနဲ့ ထုတ်တာဖြစ်လို့ root ပြီးရင် left 10 ကိုသွားပါမယ်။

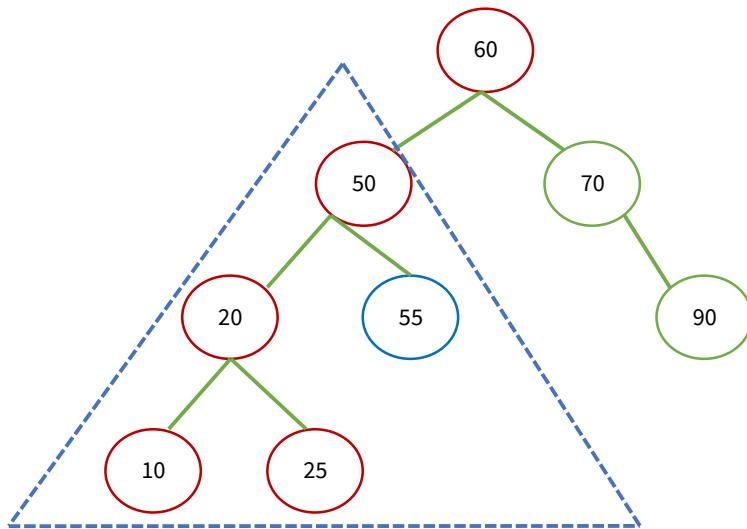


10 ကို root အနေနဲ့ ထုတ်လိုက်ပါတယ်။ Root, Left, Right ဆိုတော့ left child နဲ့ right child ကိုကြည့်ပါတယ်။ 10 မှာ left child ရော့၊ right child ပါ NULL ဖြစ်နေလို့ မရှိလို့ ဆက်သွားစရာမလိုပဲ ပြီးသွားပါတယ်။ 20 node ကနေကြည့်လိုက်ရင် 20 root ပြီးသွားပြီ၊ အခု left 10 ပြီးသွားပြီ။ ဒီတော့ Root, Left, Right မှာ root, left ပြီးသွားပြီဖြစ်လို့ right 25 ကိုသွားမှာဖြစ်ပါတယ်။ ဂွဲပြားအောင် ရောက်တဲ့ node ကို အပြာရောင်၊ သွားပြီးသား node တွေကို နိုညိုရောင် ဆိုပြီး ခြယ်ပေးထားပါတယ်။

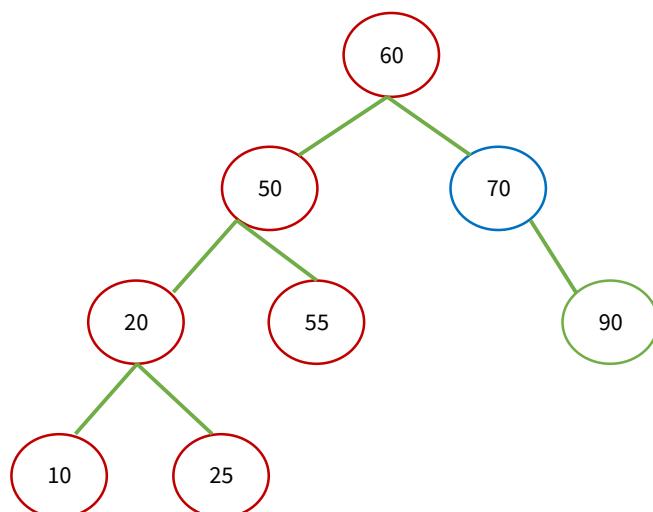


25 ကိုရောက်တဲ့ အချိန်မှာလဲ 25 ကို root အနေနဲ့ အလုပ်လုပ်မှာဖြစ်ပါတယ်။ Root, Left, Right ဆိုတဲ့အတိုင်း left child, right child တွေကြည့်တယ်။ သို့သော် 25 က child တစ်ယောက်မှ မရှိလို ဆက်မသွားရတော့ဘူး။ အခုဆိုရင် ထုတ်ပြလိုပြီးသွားတဲ့ ဖြိုကံပုံ ဝိုင်းပြထားတာလေးကို ကြည့်ပါ။

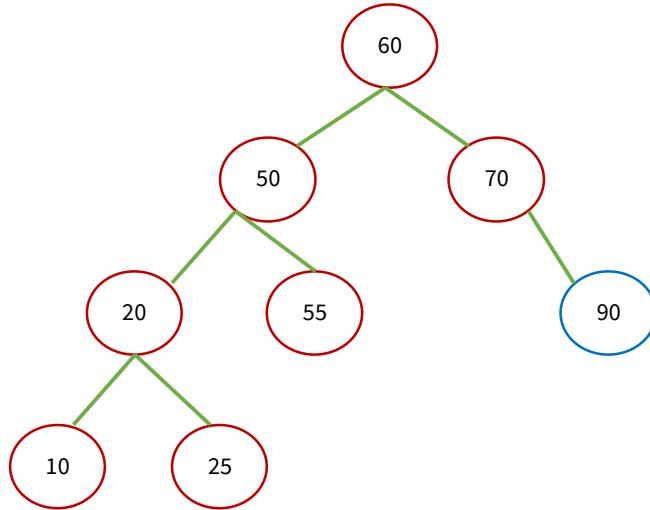
သူတို့ဟာ 50 ရဲ့ left child တွေဖြစ်ပါတယ်။ 50 ဆိုတဲ့ root ကို ထုတ်ပြီးသွားပြီ၊ ပြီးတော့ သူရဲ့ left ကိုလည်း ထုတ်ပြီးသွားပြီဖြစ်လို့ ဒီတစ်ခါ ဆက်သွားရမှာကတော့ 50 ရဲ့ right child ဖြစ်တဲ့ 55 ဖြစ်ပါတယ်။



55 ကိုလည်း root node အနေနဲ့ ထုတ်ပါတယ်။ ပြီးတော့ သူရဲ့ left child, right child တွေကို ကြည့်ပါသေးတယ်။ ဒါပေမဲ့ သူမှာ left child ရော့၊ right child ရော့ မရှိလို့ ရပ်သွားပါတယ်။ တိုက်နဲ့ ပြထားတာကို ကြည့်လိုက်ရင် 60 ဟာ root အနေနဲ့ ထုတ်ပြီးသွားပြီး သူရဲ့ left တစ်ခြမ်းလုံးလည်း ထုတ်ပြီးသွားပါပြီ။ Root, Left ပြီးတော့ ဒီတစ်ခါ သွားရမှာက right ဖြစ်လို့ 60 ရဲ့ right child 70 ကို သွားပါမယ်။



70 ကို root node အနေနဲ့ထုတ်ပါတယ်။ root ပြီးတော့ left ကိုသွားမယ်။ 70 မှာ left ကမရှိတော့ left ပြီးသွားပြီ။ ဒီတော့ 70 ရဲ့ right ဖြစ်တဲ့ 90 ကိုသွားမှာဖြစ်ပါတယ်။



90 ကို root node အနေနဲ့ထုတ်ပါတယ်။ root ပြီးတော့ left, right ကို ကြည့်တယ်။ 90 မှာ left ရော့၊ right ပါ မရှိတော့တဲ့အတွက် ရပ်သွားတယ်။ node အားလုံး သွားပြီးသွားလို ဆက်သွားစရာမရှိတော့ပဲ ရပ်သွားပါတယ်။ သွားခဲ့တဲ့ အစဉ်လိုက်ကို ပြန်ရေးပြရရင်

60, 50, 20, 10, 25, 55, 70, 90

ဖြစ်ပါတယ်။ ဒီနေရာမှာ node တစ်ခုကို ရောက်ရင် root အဖြစ်ထုတ်၊ ပြီးရင် left child သွား၊ left child ကို root အဖြစ်လုပ်၊ ပြီးတော့ root အဖြစ်လုပ်တာကြောင့် left child ဆက်သွား ဆိုပြီး left တွေ တန်းစီသွားတာ။ left ဖက် အကုန်သွားပြီးတော့ right ဖက်သွား၊ right ဖက်ရောက်သွားတဲ့ node ကိုလည်း root အဖြစ်ပြန်လုပ်။ ဆိုတော့ left, right တွေဟာ root အဖြစ် recursion ပြန်ခေါ်သလို ဖြစ်နေတယ်။

ဒါကြား pre-order algorithm ကို ရေးမယ် ဆိုရင်

---

```
preOrder(Node localRoot)
```

---

If localRoot != NULL then:

    Write localRoot→Key, “ , ”

    preOrder(localRoot → leftChild)

    preOrder(localRoot → rightChild)

---

### In-order Traversing

In-order ဆိုရင် Left, Root, Right ဖြစ်တဲ့အတွက်

---

```
inOrder(Node localRoot)
```

---

If localRoot != NULL then:

    inOrder(localRoot → leftChild)

    Write localRoot→Key, “ , ”

    inOrder(localRoot → rightChild)

---

### Post-order Traversing

Post-order ဆိုရင် Left, Right, Root ဖြစ်တဲ့အတွက်

---

```
postOrder(Node localRoot)
```

---

If localRoot != NULL then:

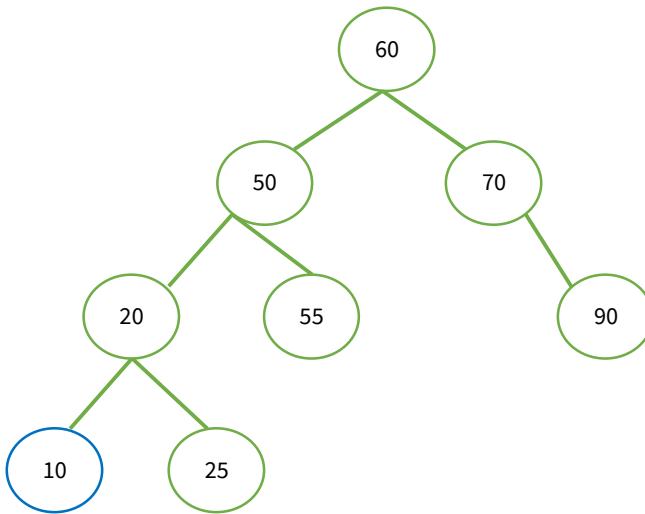
    postOrder(localRoot → leftChild)

    postOrder(localRoot → rightChild)

    Write localRoot→Key, “ , ”

---

Pre-order အတွက် တွက်ပြခဲ့တဲ့ ပုံကို post-order အတွက် trace လိုက်ပြပါမယ်။ Post order ဆိုတော့ Left, Right, Root ကို ခေါင်းထဲ ထည့်ထားပါ။



Root node ဖြစ်တဲ့ 60 ကနေစမယ်။ Left, right, root ဆိုတဲ့အတွက် root ကနေဘက်ဆုံးမှ အလုပ်လုပ်ရမှာ ဖြစ်လို့ left ကိုသွားတယ်။

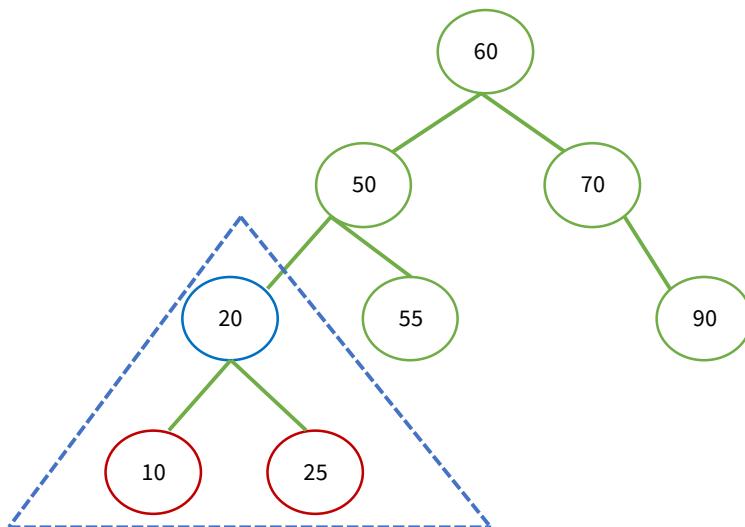
50 ကိုရောက်တယ်၊ 50 ဟာ root ဖြစ်တော့ left ကို ဆက်သွားတယ်။

20 ကိုရောက်တယ်၊ 20 ဟာ root ဖြစ်တော့ left ကို ဆက်သွားတယ်။

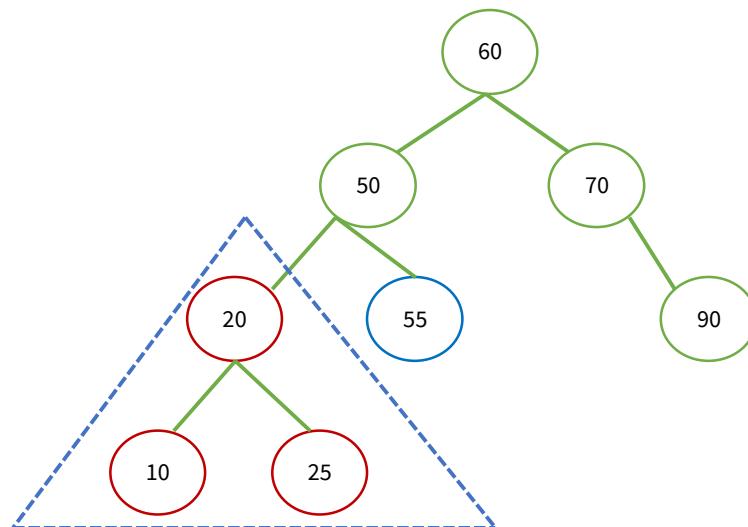
10 ကိုရောက်တယ်၊ 10 ဟာ root ဖြစ်တော့ left ကို ဆက်သွားတယ်။

Left က NULL ဖြစ်နေတော့ left မရှိတော့လို့ Left ပြီးပြဖြစ်တယ်။ ဒါတော့ 10 ရဲ့ right ကိုသွားတယ်။ right မရှိတော့ right လည်းပြီးသွားပြီ။ Left, right , root ဆိုတော့ Left, Right ပြီးသွားလို့ 10 ကို root အနေနဲ့ထုတ်လိုက်တယ်။

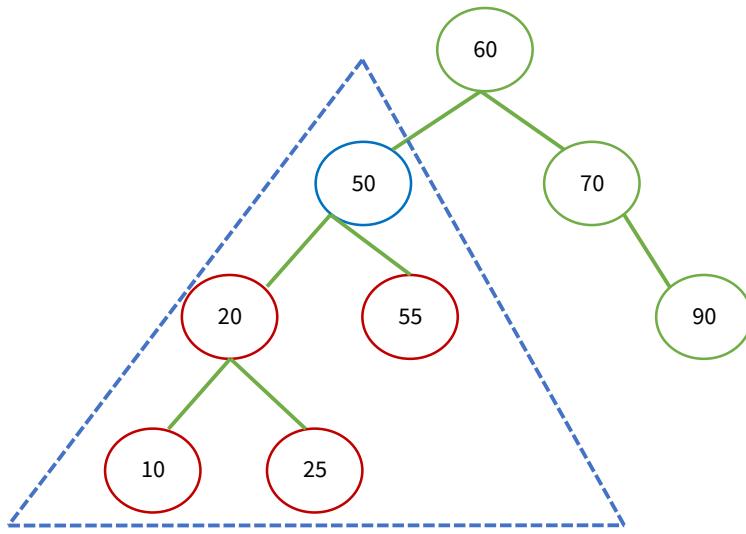
10 ဆိုတာဟာ 20 ရဲ့ left ဖြစ်တယ်။ 20 ရဲ့ left ပြီးတော့ 20 ရဲ့ right ကိုသွားတယ်။ 25 ကိုရောက်တယ်။  
 25 ကို root node အဖြစ်ကြည့်တယ်။ root ဆိုတော့ left, right ကို အရင်သွားရမယ်။ 25 မှာ left, right  
 မရှိတော့ ပြီးပြီလို့ ယူဆပြီး 25 ကို root အနေနဲ့ ထုတ်လိုက်တယ်။



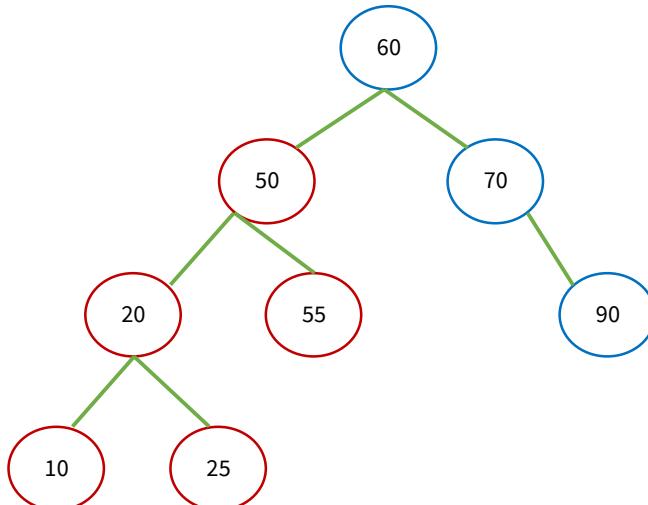
20 ကနေကြည့်လိုက်ရင် သူ့ရဲ့ left 10, သူ့ရဲ့ right 25 အလုပ်လုပ်လို့ ပြီးသွားပြီ။ Left, right ပြီးတော့  
 20 ကို root အနေနဲ့ ထုတ်လိုက်တယ်။ တို့ငံ့ ပြထားသလိုပဲ 50 ကနေကြည့်လိုက်ရင် သူ့ရဲ့ left  
 ဖက်ခြမ်း ပြီးသွားပြီ။ left ပြီးတော့ right ဖြစ်တဲ့ 55 ကိုသွားမယ်။



55 ကို root အဖြစ်ကြည့်တယ်။ Left, Right, Root ဖြစ်လို့ 55 ရဲ့ left, right ကိုသွားတယ်။ မရှိတော့မှ  
 left, right ပြီးပြီဖြစ်လို့ 55 ကို root အဖြစ်ထုတ်လိုက်တယ်။



55 ထုတ်လည်းပြီးရော 50 ဟာ သူရဲ့ left, right ပြီးသွားပြီဖြစ်လို 50 ကို root အဖြစ်ထုတ်တယ်။ 50 လည်း ထုတ်ပြီးသွားတော့ 60 ကနေကြည့်လိုက်ရင် သူရဲ့ left ဖက်ခြမ်းပြီးသွားပြီဖြစ်ပါတယ်။ ဒါတော့ right ဖြစ်တဲ့ 70 ကိုသွားမှာ ဖြစ်ပါတယ်။



ထုံးစံအတိုင်း 70 ကို root အဖြစ်ကြည့်မယ်။ Left, right, root ဆိုတော့ 70 ရဲ့ left ကိုသွားတယ်။ Left မရှိတော့ left ပြီးပြီဖြစ်လို 70 ရဲ့ right ကိုသွားတယ်။ 90 ကိုရောက်တယ်။ 90 ကို root အဖြစ် ကြည့်မယ်။ 90 ရဲ့ left, right ကို အရင်သွားတယ်။ မရှိတော့လို left, right ပြီးမှ 90 ကို root အဖြစ်ထုတ်တယ်။ 90 ပြီးတော့ 70 ရဲ့ left, right ပြီးသွားလို 70 ကို root အဖြစ်ထုတ်လိုက်တယ်။ 70

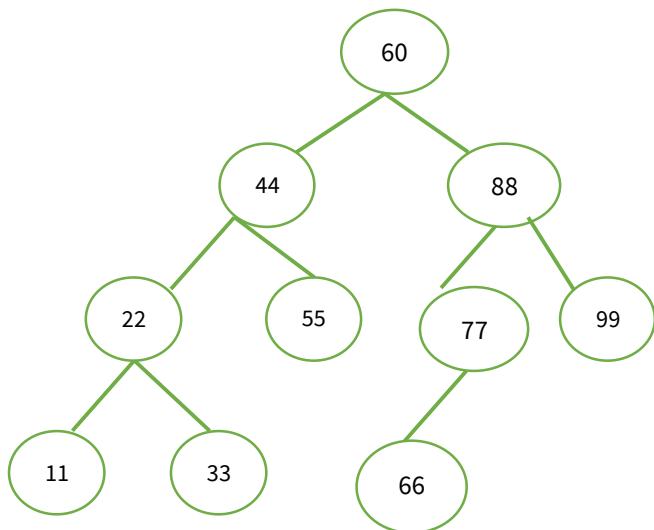
လည်းထုတ်ပြီးသွားတော့ 60 ဟာ left ဖက်ခြမ်းရော့၊ right ဖက်ခြမ်းပါ ပြီးသွားပြီဖြစ်လို့ 60 ကို root အဖြစ်နောက်ဆုံးထုတ်လိုက်တယ်။ node အားလုံးကို သွားပြီးပြီဖြစ်လို့ ပြီးသွားပြီဖြစ်ပါတယ်။

10,25, 20, 55, 50, 90, 70, 60

ဆိုပြီး အဖြေထွက်ပါတယ်။



လေ့ကျင့်ကြည့်ကြရအောင်။



1. Pre-order ဖြင့် output ထုတ်ပြပါ။
2. In-order ဖြင့် output ထုတ်ပြပါ။
3. Post-order ဖြင့် output ထုတ်ပြပါ။

## Heap

Data တွေကို tree ပုံစံနဲ့ သိမ်းဖို့ left child နဲ့ right child နှစ်ခုရှိတဲ့ heap ပုံစံမျိုးနဲ့ လည်း သိမ်းပါတယ်။

Heap ကို နှစ်မျိုးထပ်ခဲ့ပါတယ်။ Max-heap နဲ့ Min-heap ဖြစ်ပါတယ်။

Max-heap - parent node ဟာ child တွေထက် ကြီးတာ သို့မဟုတ် ညီတာ ဖြစ်ရပါမယ်။

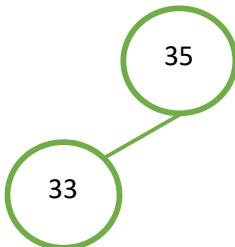
Min-heap –parent node ဟာ child တွေထက် ငယ်တာ သို့မဟုတ် ညီတာ ဖြစ်ရပါမယ်။

$$\text{LA} = \{ 35, 33, 42, 10, 14, 19, 27, 44, 26, 31 \}$$

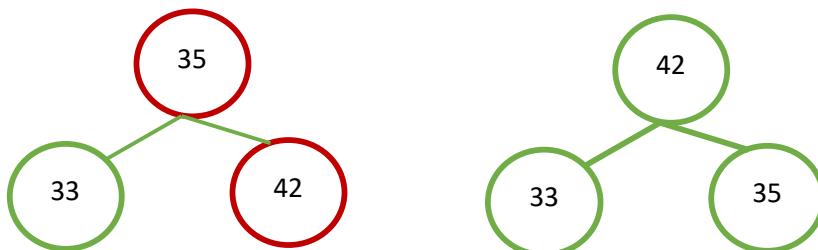
### Insert 35



**Insert 33:** 33 ဝင်လာတော့ လွှတ်တဲ့ 35 ရဲ့အောက်မှာ left child အနေနဲ့ ထည့်မှာ ဖြစ်ပါတယ်။  
ပြီးတော့ သူရဲ့ parent 35 ထက်ငယ်တော့ ဘာမှ လုပ်စရာမလိုပါဘူး။

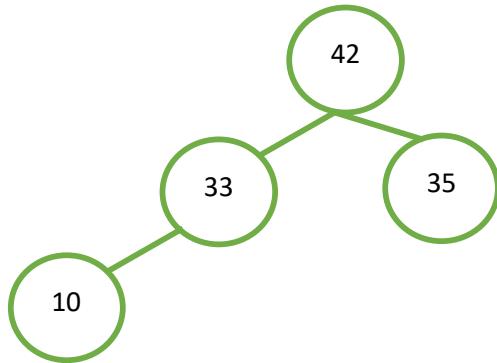


**Insert 42:** 42 ဝင်လာတော့ လွှတ်တဲ့ 35 ရဲ့အောက်မှာ လွှတ်နေတဲ့ right child အနေနဲ့ ထည့်မှာ ဖြစ်ပါတယ်။

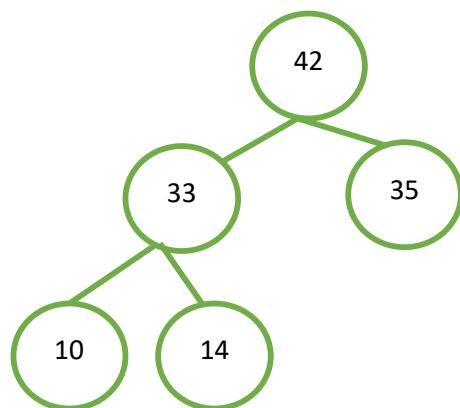


42 ကိုထည့်လိုက်တော့ parent ဖြစ်တဲ့ 35 ထက်ကြီးနေလို့ 42 နဲ့ 35 နေရာချင်းလဲလိုက်တယ်။  
အခုခံ့ရင် parent 42 ဟာ child 2 ခုလုံးထက်ကြီးသွားလို့ အဆင်ပြေသွားပါဖြူ။

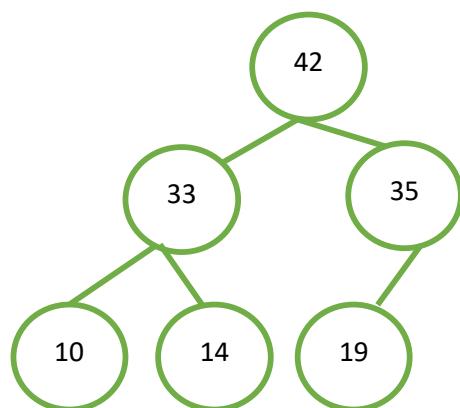
**Insert 10:** 10 ဝင်လာတော့ လွှတ်တဲ့ 33 ရဲ့အောက်မှာ လွှတ်နေတဲ့ left child အနေနဲ့ ထည့်မှာဖြစ်ပါတယ်။ parent ထက်ယောက်နေတော့ ဘာမှ လုပ်စရာမလိုပါဘူး။



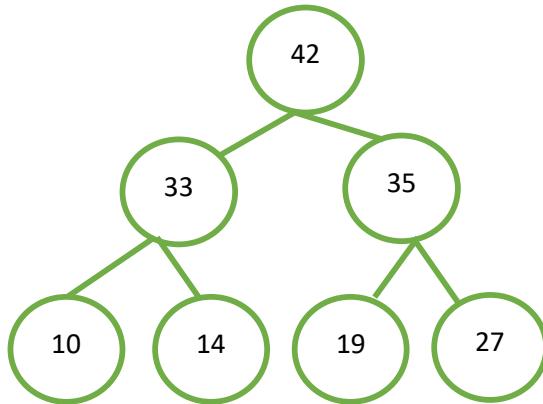
**Insert 14:** 14 ဝင်လာတော့ လွှတ်တဲ့ 33 ရဲ့အောက်မှာ လွှတ်နေတဲ့ right child အနေနဲ့ ထည့်မှာဖြစ်ပါတယ်။ parent ထက်ယောက်နေတော့ ဘာမှ လုပ်စရာမလိုပါဘူး။



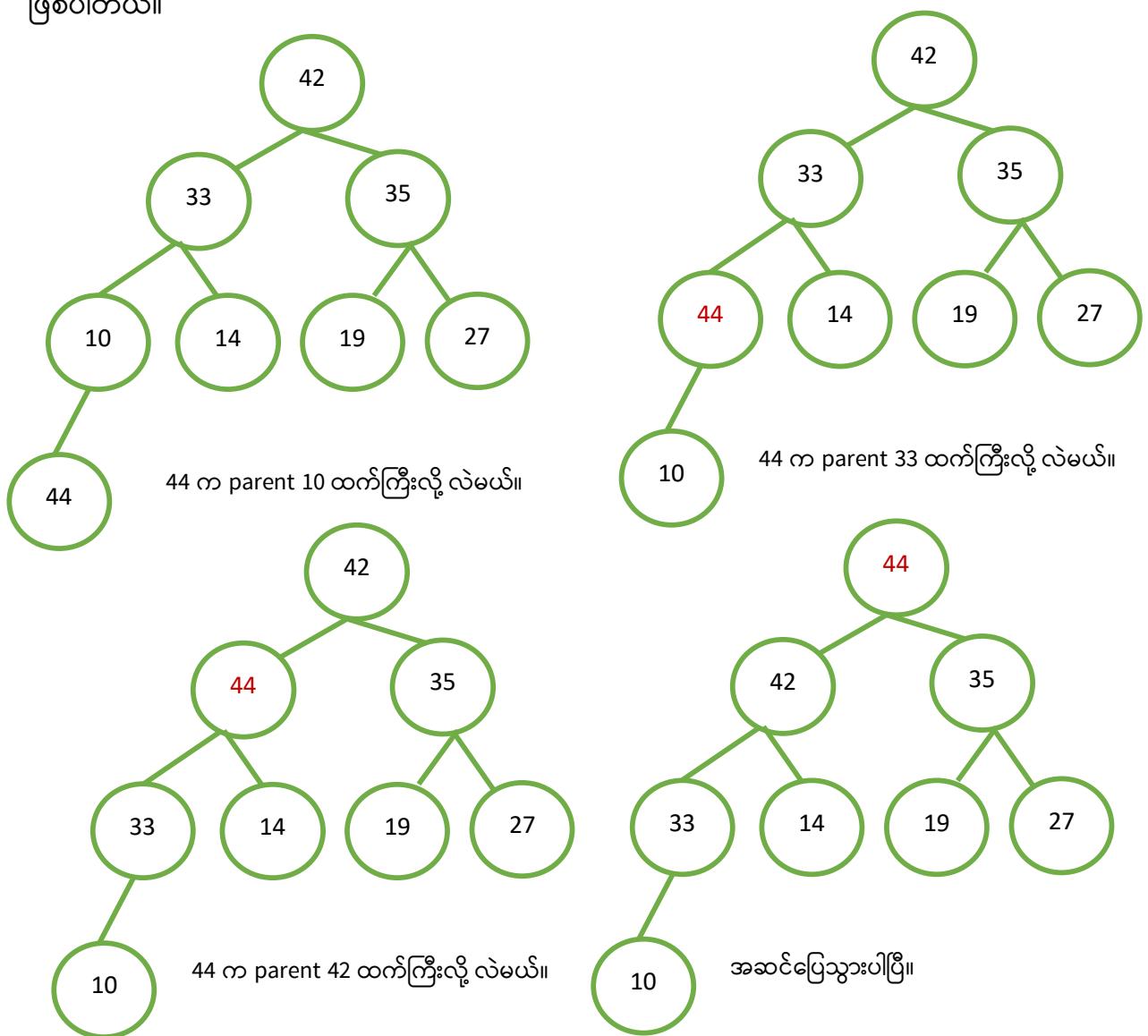
**Insert 19:** 19 ဝင်လာတော့ လွှတ်တဲ့ 35 ရဲ့အောက်မှာ လွှတ်နေတဲ့ left child အနေနဲ့ ထည့်မှာဖြစ်ပါတယ်။ parent ထက်ယောက်နေတော့ ဘာမှ လုပ်စရာမလိုပါဘူး။



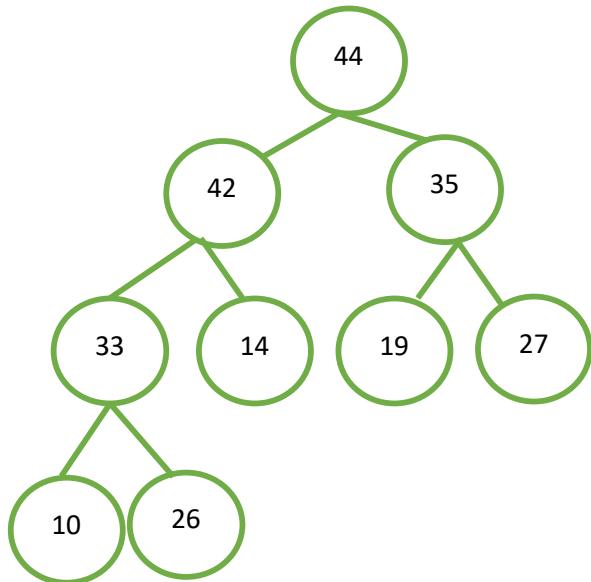
**Insert 27:** 27 ဝင်လာတော့ လွှတ်တဲ့ 35 ရဲ့အောက်မှာ လွှတ်နေတဲ့ right child အနေနဲ့ ထည့်မှာဖြစ်ပါတယ်။ parent ထက်ယ်နေတော့ ဘာမှ လုပ်စရာမလိုပါဘူး။



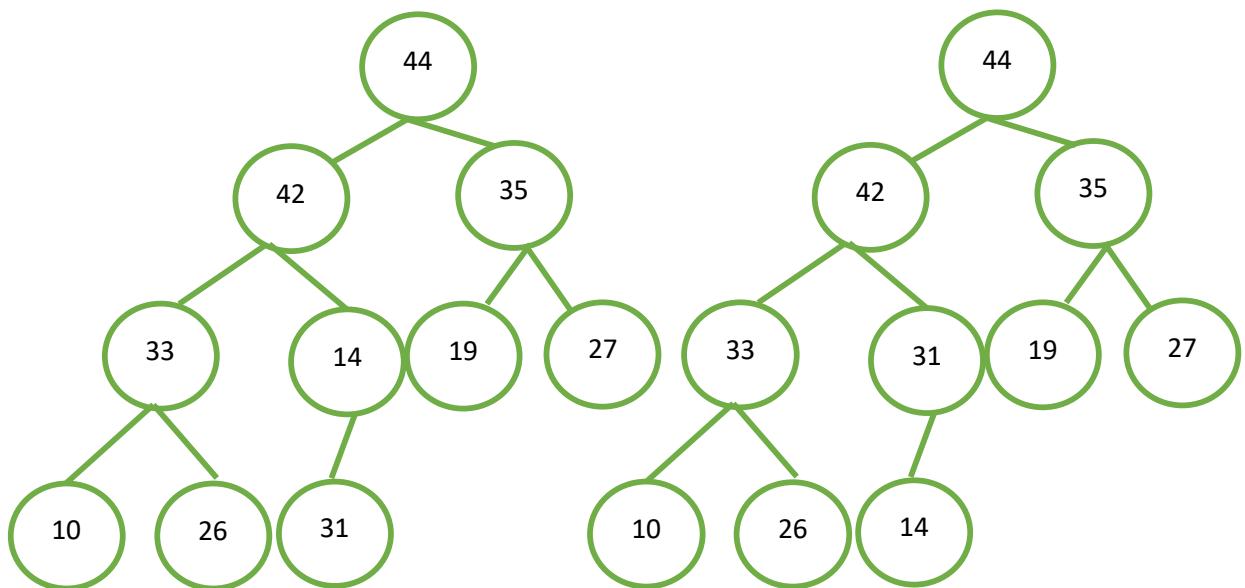
**Insert 44:** 44 ဝင်လာတော့ လွှတ်တဲ့ 10 ရဲ့အောက်မှာ လွှတ်နေတဲ့ left child အနေနဲ့ ထည့်မှာဖြစ်ပါတယ်။



**Insert 26:** 26 ဝင်လာတော့ လွှတ်တဲ့ 33 ရဲ့အောက်မှာ လွှတ်နေတဲ့ right child အနေနဲ့ ထည့်မှာဖြစ်ပါတယ်။ parent ထက်ယ်နေတော့ ဘာမှ လုပ်စရာမလိုပါဘူး။



**Insert 31:** 31 ဝင်လာတော့ လွှတ်တဲ့ 14 ရဲ့အောက်မှာ လွှတ်နေတဲ့ left child အနေနဲ့ ထည့်မှာဖြစ်ပါတယ်။ parent ထက်ကြီးနေလို့ parent နဲ့ နေရာချင်းလဲ။

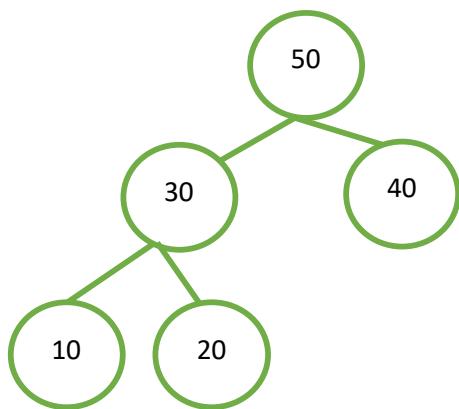


ဒီမှာက parent ကို child တွေထက်ကြီးတာထားလို့ Max- heap ဖြစ်ပါတယ်။

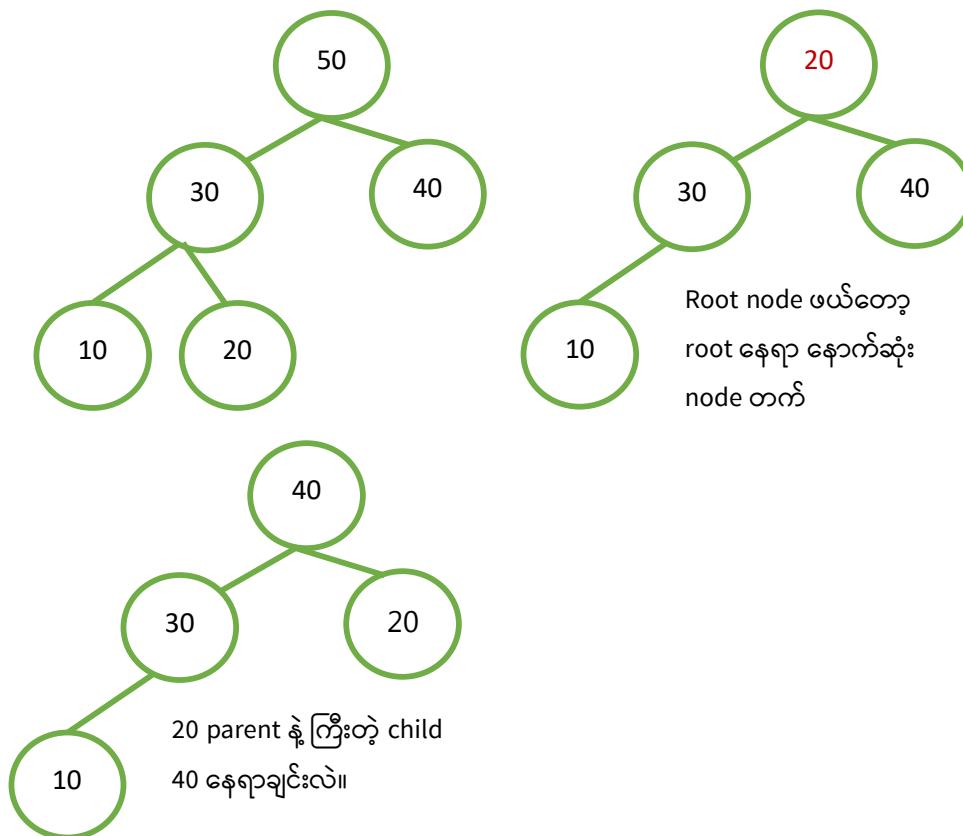
Min-heap နဲ့ ပြန်ပြီး လေ့ကျင့်ကြည့်ပါ။

## Heap Sort

Heap Sort ရဲ့ အလုပ်လုပ်ပုံကတော့ root node ကို ဖယ်ဖယ်သွားမှာဖြစ်ပြီ၊ နောက်ဆုံး node က root နေရာတက်လာရပါမယ်။ ပြီးမှ parent က ကြီးရမယ် သို့မဟုတ် ညီရမယ် ဆိတဲ့ စဉ်းကမ်းနဲ့ ကိုက်အောင် ပြန်စီမှာ ဖြစ်ပါတယ်။

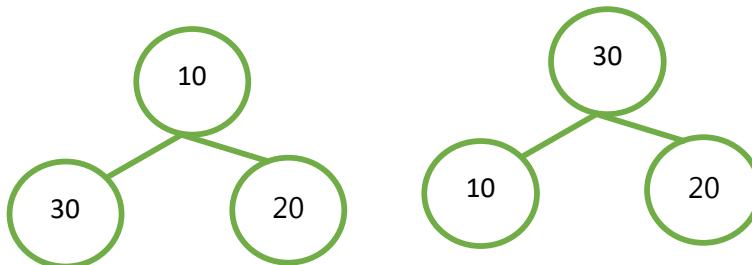


**Remove root node: 50**



Sorted : 50

**Remove root node:** 40 နေရာ နောက်ဆုံး node 10 တက်မယ်။

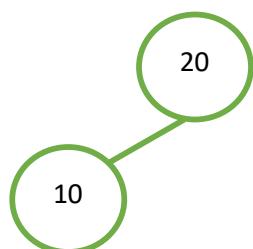


10 parent နဲ့ ကြီးတဲ့ child

30 နေရာချင်းလဲမယ်။

Sorted: 40, 50

**Remove root node:** 30 နေရာ နောက်ဆုံး node 20 တက်မယ်။ parent ကြီးတော့ ဘာမှ လုပ်စရာမလိုဘူး။



Sorted: 30, 40, 50

**Remove root node:** 20 နေရာ နောက်ဆုံး node 20 တက်မယ်။



Sorted: 20, 30, 40, 50

Node တစ်ခုတည်းကျန်တာကြောင့် ပြီးသွားပါပြီ။

Sorted: 10, 20, 30, 40, 50



လေ့ကျင့်ကြည့်ကြရအောင်။

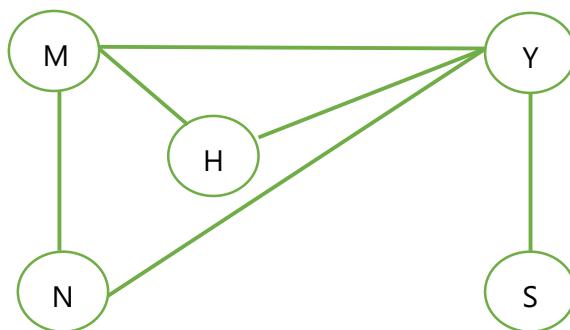
LA = { 35, 33, 42, 10, 14, 19, 27, 44, 26, 31 } ကို Min-Heap ဖံ့ဖိုးပါ။ ပြီးလျှင် heap sort ကိုသုံးပြီး sorting စီပေးပါ။

## အခန်း (၁၇)

### Graph

#### Graph ဆိတာဘာလဲ။

လေကြောင်းခရီးစဉ်တွေကို graph ရဲ့ နမူနာတစ်ခုအနေနဲ့ ကြည်လိုက်ကြရအောင်။

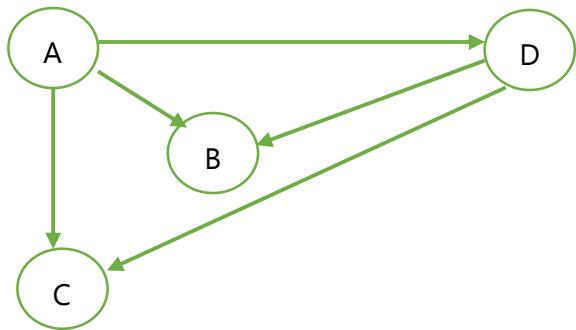


M – Mandalay, Y – Yangon, N – Nyaung U, H – He Hoe, S – Sittwe

ဒီမှာ အပိုင်းလေးတွေနဲ့ ပြထားတဲ့ မြို့အမည်တွေကို vertex လို့ခေါ်ပြီး vertex တွေအကြား ဆက်ထားတဲ့ လိုင်းတွေကို edge လို့ခေါ်ပါတယ်။

Edge တွေကို ကြည်ခြင်းအားဖြင့် Mandalay ကနေ Yangon, He Hoe, Nyang U စတဲ့ မြို့တွေကို သွားလိုရတယ်။ Yangon ကနေ Mandalay, He Hoe, Nyaung U, Sittwe စတဲ့ မြို့တွေကို သွားလိုရတယ်။ He Hoe ကနေ Mandalay နဲ့ Yangon၊ Nyanung U ကနေလည်း Mandalay နဲ့ Yangon၊ Sittwe ကနေ Yangon စတဲ့ ခရီးစဉ်တွေရှိတာ သိနိုင်ပါတယ်။

ဒီမှာ edge တွေက arrow မပါလို့ နှစ်ဖက်အပြန်အလှန်သွားလို့ ရတဲ့ သဘောဖြစ်ပါတယ်။ အဲဒါကို **undirected graph** လို့ ခေါ်ပါတယ်။ arrow ပါခဲ့ရင်တော့ **directed graph** လို့ ခေါ်ပါတယ်။ Directed Graph ကို နမူနာပြရရင်တော့



ဒီမှာဆိုရင် A ကနေ B,C, D ကိုသွားလိုရမယ်။ B, C ကနေ ဘယ်မှ သွားမရဘူး။ D ကနေ B, C ကို  
သွားလိုရတယ်။

Graph ထွေကိုလည်း traversing လုပ်တော့မယ်ဆိုရင် သွားတဲ့ နည်း၂ နည်းရှိပါတယ်။

1. Depth-First Search
2. Breadth-First Search တို့ဖြစ်ပါတယ်။

## Adjacency Matrix

Graph ကို မြင်သာအောင် **Adjacency Matrix** နဲ့ ပြပေးလိုရပါတယ်။ အပေါ်မှာ နမူနာပြခဲ့တဲ့  
လေကြောင်းခရီးစဉ် Undirected Graph အတွက် Adjacency Matrix ဆဲကြည့်ပါမယ်။ ရှိသမျှ vertex  
တွေအားလုံးကို row နဲ့ column မှာ ခေါင်းစဉ်တွေ အနေနဲ့ ချရေးပါ။ ပြီးရင် ပုံမှာ ခရီးစဉ်ရှိရင် ၁၊ မရှိရင်  
၀ ဆိုပြီး ဖြည့်ရမှာဖြစ်ပါတယ်။ HH, MM စတဲ့ မိမိကိုယ်မိမိ ပြန်ညွှန်းတဲ့ နေရာတွေကတော့  
ကိုယ့်မြို့ကိုယ်ပြန်သွားတာ မရှိလို့ ၀ ဖြစ်ပါတယ်။ တစ်ချို့ ရှားရှားပါးနေရာတွေမှာတော့  
မိမိကိုယ်ကိုယ်ပြန်ညွှန်းတဲ့ edge တွေ ပါတတ်ပါတယ်။

|   | H | M | N | S | Y |
|---|---|---|---|---|---|
| H | 0 | 1 | 0 | 0 | 1 |

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| M | 1 | 0 | 1 | 0 | 1 |
| N | 0 | 1 | 0 | 0 | 1 |
| S | 0 | 0 | 0 | 0 | 1 |
| Y | 1 | 1 | 1 | 1 | 0 |

ဒါကတော့ အပေါ်မှာ ပြခဲ့တဲ့ directed Graph အတွက် Adjacency Matrix ဖြစ်ပါတယ်။

|   | A | B | C | D |
|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 |
| B | 0 | 0 | 0 | 0 |
| C | 0 | 0 | 0 | 0 |
| D | 0 | 1 | 1 | 0 |

### Adjacency List

Graph ကို မြင်သာအောင် **Adjacency List** နဲ့ လည်း ပြပေးလိုရပါတယ်။ အပေါ်မှာ ပြခဲ့တဲ့ Undirected Graph အတွက် အရင်ဆွဲပါမယ်။

| Vertex |                                 |
|--------|---------------------------------|
| H      | $M \rightarrow Y$               |
| M      | $H \rightarrow N \rightarrow Y$ |
| N      | $M \rightarrow Y$               |
| S      | Y                               |

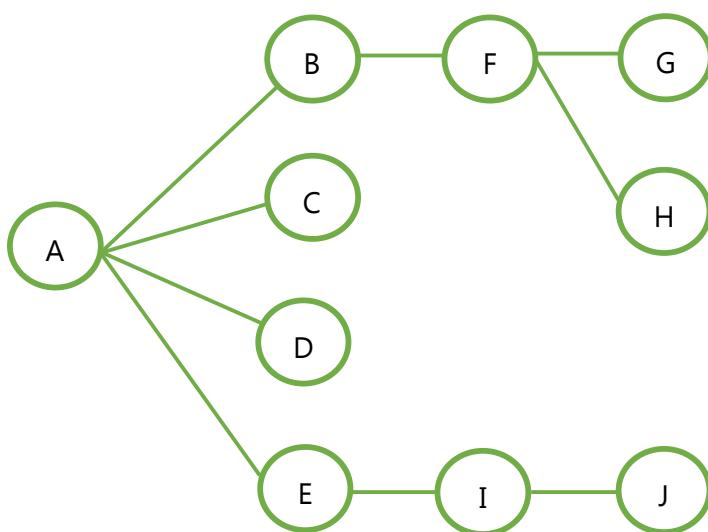
|   |                                               |
|---|-----------------------------------------------|
| Y | $H \rightarrow M \rightarrow N \rightarrow S$ |
|---|-----------------------------------------------|

နားလည်လွယ်တာကြောင့် ရှင်းပြစ်ရာ မလိုလောက်ဘူး ထင်ပါတယ်။ အပေါ်မှာ ပြခဲ့တဲ့ directed Graph အတွက် ဆက်ဆွဲပါမယ်။

| Vertex |                                 |
|--------|---------------------------------|
| A      | $B \rightarrow C \rightarrow D$ |
| B      |                                 |
| C      |                                 |
| D      | $B \rightarrow C$               |

### Depth-First Search (DFS)

Depth ဆိုတဲ့အတိုင်း အဆုံးအထိ သွားပြီးမှ ပြန်လာမှာ ဖြစ်ပါတယ်။



Depth-First Search နဲ့ ဒီဟာကို ထုတ်ရင်

A, B, F, G ဆိုပြီး အရင်ထုတ်မယ်။

G ပြီးတော့ ဆက်သွားစရာ ရှိလား ကြည့်တယ်။ မရှိတော့ ရှုံးပြန်လာတယ်။ F ရောက်တယ်။

F ကနေ ဆက်သွားစရာရှိလားကြည့်တော့ ရှိတယ်။ H ကိုဆက်သွားတယ်။ H ကနေ ဆက်သွားစရာ ရှိလားကြည့်တော့မရှိတော့ဘူး။ ဒီတော့ ရှုံးပြန်လာတယ်။

အခုပြောသလောက်ကြည့်ရင်ကို သိသာနေပြီ၊ နောက်ဆုံးသွားခဲ့တဲ့ နေရာကနေ ဆက်ဆက်သွားတယ်။ ဒီတော့ နောက်ဆုံးကောင် အရင်ဆုံးလုပ်တာဖြစ်လို့ Stack ကို သုံးပါမယ်။

| Process | Stack      | ရှင်းလင်းချက်                                                                                                                |
|---------|------------|------------------------------------------------------------------------------------------------------------------------------|
| Visit A | A          | Visit တာနဲ့ stack ပေါ်တင်တယ်။ A ကနေဆက်ပြီးသွားမယ်။                                                                           |
| Visit B | A, B       | Visit တာနဲ့ stack ပေါ်တင်တယ်။ နောက်ဆုံးတင်ခဲ့တဲ့ B ကနေ ဆက်ပြီးသွားမယ်။                                                       |
| Visit F | A, B, F    | Visit တာနဲ့ stack ပေါ်တင်တယ်။ နောက်ဆုံးတင်ခဲ့တဲ့ F ကနေ ဆက်ပြီးသွားမယ်။                                                       |
| Visit G | A, B, F, G | Visit တာနဲ့ stack ပေါ်တင်တယ်။ နောက်ဆုံးတင်ခဲ့တဲ့ G ကနေ ဆက်ပြီးသွားမယ်။                                                       |
| Pop     | A, B, F    | သို့သော် G မှာ ဆက်သွားစရာမရှိတော့ဘူး။ ဒါကြောင့် G ကို ဖြုတ်တယ်။ Stack ပေါ်မှာ နောက်ဆုံး F ဖြစ်နေလို့ F ကနေ ဆက်ပြီးသွားပါမယ်။ |
| Visit H | A, B, F, H | Visit တာနဲ့ stack ပေါ်တင်တယ်။ နောက်ဆုံးတင်ခဲ့တဲ့ H ကနေ ဆက်ပြီးသွားမယ်။                                                       |

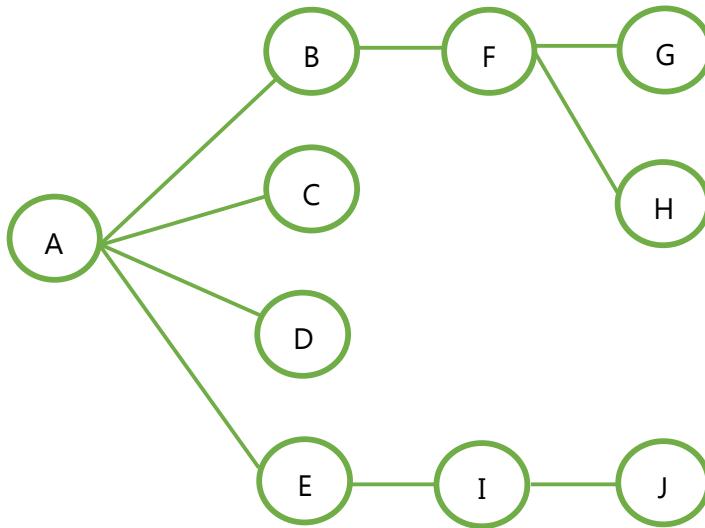
|                |         |                                                                                                                                                                     |
|----------------|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Pop</b>     | A, B, F | သို့သော် H မှာ ဆက်သွားစရာမရှိတော့ဘူး။ ဒါကြောင့် H ကိုဖြုတ်တယ်။ Stack ပေါ်မှာ နောက်ဆုံး F ဖြစ်နေလို့ F ကနေဆက်ပြီးသွားပါမယ်။                                          |
| <b>Pop</b>     | A, B    | သို့သော် F မှာ သွားစရာ အကုန်သွားပြီး သွားပြီ။ ဒါတော့ ဆက်သွားစရာမရှိတော့ဘူး။ ဒါကြောင့် F ကို ဖြုတ်တယ်။ Stack ပေါ်မှာ နောက်ဆုံး B ဖြစ်နေလို့ B ကနေဆက်ပြီးသွားပါမယ်။   |
| <b>Pop</b>     | A       | သို့သော် B မှာ သွားစရာ အကုန်သွားပြီး သွားပြီ။ ဒါတော့ ဆက်သွားစရာမရှိတော့ဘူး။ ဒါကြောင့် pop လုပ်လိုက်တယ်။ Stack ပေါ်မှာ နောက်ဆုံး A ဖြစ်နေလို့ A ကနေဆက်ပြီးသွားပါမယ်။ |
| <b>Visit C</b> | A, C    | Visit တာနဲ့ stack ပေါ်တင်တယ်။ C ကနေဆက်ပြီးသွားမယ်။                                                                                                                  |
| <b>Pop</b>     | A       | သို့သော် C မှာ ဆက်သွားစရာမရှိတော့ဘူး။ ဒါကြောင့် pop လုပ်လိုက်တယ်။ Stack ပေါ်မှာ နောက်ဆုံး A ဖြစ်နေလို့ A ကနေဆက်ပြီးသွားပါမယ်။                                       |
| <b>Visit D</b> | A, D    | Visit တာနဲ့ stack ပေါ်တင်တယ်။ D ကနေဆက်ပြီးသွားမယ်။                                                                                                                  |
| <b>Pop</b>     | A       | သို့သော် D မှာ ဆက်သွားစရာမရှိတော့ဘူး။ ဒါကြောင့် pop လုပ်လိုက်တယ်။ Stack ပေါ်မှာ နောက်ဆုံး A ဖြစ်နေလို့ A ကနေဆက်ပြီးသွားပါမယ်။                                       |

|                |            |                                                                                                                                             |
|----------------|------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Visit E</b> | A, E       | Visit တာနဲ့ stack ပေါ်တင်တယ်။ E ကနေဆက်ပြီး သွားမယ်။                                                                                         |
| <b>Visit I</b> | A, E, I    | Visit တာနဲ့ stack ပေါ်တင်တယ်။ I ကနေဆက်ပြီး သွားမယ်။                                                                                         |
| <b>Visit J</b> | A, E, I, J | Visit တာနဲ့ stack ပေါ်တင်တယ်။ J ကနေဆက်ပြီး သွားမယ်။                                                                                         |
| <b>Pop</b>     | A, E, I    | သို့သော် J မှာ ဆက်သွားစရာမရှိတော့ဘူး။ ဒါကြောင့် pop လုပ်လိုက်တယ်။ Stack ပေါ်မှာ နောက်ဆုံး I ဖြစ်နေလို့ ကနေ ဆက်ပြီးသွားပါမယ်။                |
| <b>Pop</b>     | A, E       | သို့သော် I မှာ ဆက်သွားစရာမရှိတော့ဘူး။ ဒါကြောင့် pop လုပ်လိုက်တယ်။ Stack ပေါ်မှာ နောက်ဆုံး E ဖြစ်နေလို့ E ကနေ ဆက်ပြီးသွားပါမယ်။              |
| <b>Pop</b>     | A          | သို့သော် E မှာ ဆက်သွားစရာမရှိတော့ဘူး။ ဒါကြောင့် pop လုပ်လိုက်တယ်။ Stack ပေါ်မှာ နောက်ဆုံး A ဖြစ်နေလို့ A ကနေ ဆက်ပြီးသွားပါမယ်။              |
| <b>Pop</b>     |            | သို့သော် A မှာ ဆက်သွားစရာမရှိတော့ဘူး။ ဒါကြောင့် pop လုပ်လိုက်တယ်။ Stack ဟာ empty ဖြစ်သွားလို့ အားလုံးကို DFS နဲ့ သွားခဲ့ပြီး ဖြစ်သွားပါပြီ။ |

A, B, F, G, H, C, D, E, I, J ဆိုပြီးသွားခဲ့တာ ဖြစ်ပါတယ်။ ဒီလောက်ဆိုရင် Stack ကိုသုံးပြီး Depth-First Search(DFS) ရဲ့ အလုပ်လုပ်ပုံကို သိလောက်ပါပြီ။

## Breadth-First Search (BFS)

Breadth-First Search (BFS) ကတေသာ Depth-First Search(DFS) နဲ့ ပြောင်းပြန် ဖြစ်ပါတယ်။ သူက အရင်ရောက်တဲ့ vertex နဲ့ ဆက်စပ်တာတွေကို အရင်သွားမှာ ဖြစ်လို့ Queue ကို အသုံးပြုရမှာ ဖြစ်ပါတယ်။



| Process | Queue   | ရှင်းလင်းချက်                                                                        |
|---------|---------|--------------------------------------------------------------------------------------|
| Visit A |         | A ဖြင့်ဆက်စပ်တာ အကုန်သွားပါမယ်။ လက်ရှိ A နဲ့ ဆက်စပ်တာ သွားနေလို့ A ကို မသိမ်းပါဘူး။  |
| Visit B | B       | လက်ရှိ A နဲ့ စက်ဆက်တာတွေကို သွားနေတာဖြစ်လို့ B ကို Queue ပေါ်ခဲားသိမ်းထားလိုက်ပါတယ်။ |
| Visit C | B, C    | လက်ရှိ A နဲ့ စက်ဆက်တာတွေကို သွားနေတာဖြစ်လို့ C ကို Queue ပေါ်ခဲားသိမ်းထားလိုက်ပါတယ်။ |
| Visit D | B, C, D | လက်ရှိ A နဲ့ စက်ဆက်တာတွေကို သွားနေတာဖြစ်လို့ D ကို Queue ပေါ်ခဲားသိမ်းထားလိုက်ပါတယ်။ |

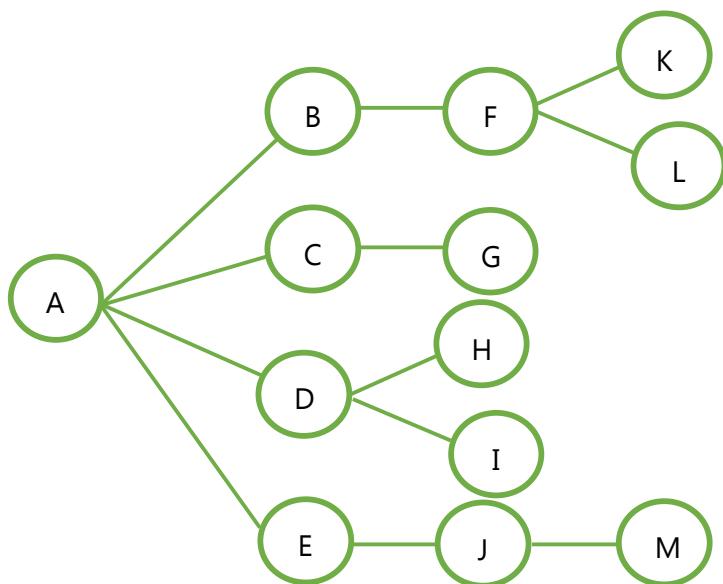
|                |            |                                                                                                                                                                                                                                                                   |
|----------------|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Visit E</b> | B, C, D, E | လက်ရှိ A နဲ့ စက်ဆက်တာတွေကို သွားနေတာဖြစ်လို့ E ကို Queue ပေါ်ခဲာ သိမ်းထားလိုက်ပါတယ်။ အခုခံ့ရင် A နဲ့ ဆက်စပ်ပြီး မသွားရသေးတာ ကုန်သွားပြီဖြစ်လို့ A ပြီးရင် အရင်ဆုံးသွားခဲ့တဲ့ B ကနေဆက်သွားပါမယ်။ ဒါကြောင့် ရှုဆုံး B ကို ဖျက်ချင်တော့ Queue ကနေ Delete လိုက်ပါမယ်။ |
| <b>Delete</b>  | C, D, E    | အခုခံ့ရင် ဖြုတ်လိုက်တဲ့ B နဲ့ ဆက်စပ်တာကို ဆက်သွားမှာ ဖြစ်ပါတယ်။                                                                                                                                                                                                   |
| <b>Visit F</b> | C, D, E, F | B နဲ့ ဆက်စပ်တာ ကုန်ပြီဖြစ်လို့ နောက်တစ်ခု ထပ်ပြီး Delete ပါမယ်။                                                                                                                                                                                                   |
| <b>Delete</b>  | D, E, F    | အခုခံ့ရင် ဖြုတ်လိုက်တဲ့ C နဲ့ ဆက်စပ်တာကို ဆက်သွားမှာ ဖြစ်ပါတယ်။ C နဲ့ ဆက်စပ်ပြီး မသွားရသေးတာ မရှိတော့လို့ နောက်တစ်ခု ထပ်ပြီး Delete ပါမယ်။                                                                                                                        |
| <b>Delete</b>  | E, F       | အခုခံ့ရင် ဖြုတ်လိုက်တဲ့ D နဲ့ ဆက်စပ်တာကို ဆက်သွားမှာ ဖြစ်ပါတယ်။ D နဲ့ ဆက်စပ်ပြီး မသွားရသေးတာ မရှိတော့လို့ နောက်တစ်ခု ထပ်ပြီး Delete ပါမယ်။                                                                                                                        |
| <b>Delete</b>  | F          | အခုခံ့ရင် ဖြုတ်လိုက်တဲ့ E နဲ့ ဆက်စပ်တာကို ဆက်သွားမှာ ဖြစ်ပါတယ်။                                                                                                                                                                                                   |
| <b>Visit I</b> | F, I       | E နဲ့ ဆက်စပ်ပြီး မသွားရသေးတာ ကုန်ပြီဖြစ်လို့ နောက်တစ်ခု ထပ်ပြီး Delete ပါမယ်။                                                                                                                                                                                     |
| <b>Delete</b>  | I          | အခုခံ့ရင် ဖြုတ်လိုက်တဲ့ F နဲ့ ဆက်စပ်တာကို ဆက်သွားမှာ ဖြစ်ပါတယ်။                                                                                                                                                                                                   |

|                |         |                                                                                                                                               |
|----------------|---------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Visit G</b> | I, G    |                                                                                                                                               |
| <b>Visit H</b> | I, G, H | F နဲ့ ဆက်စပ်ပြီး မသွားရသေးတာ ကုန်ပြီဖြစ်လို့ နောက်တစ်ခု ထပ်ပြီး Delete ပါမယ်။                                                                 |
| <b>Delete</b>  | G, H    | အခုဆိုရင် ဖြုတ်လိုက်တဲ့ I နဲ့ ဆက်စပ်တာကို ဆက်သွားမှာ ဖြစ်ပါတယ်။                                                                               |
| <b>Visit J</b> | G, H, J | I နဲ့ ဆက်စပ်ပြီး မသွားရသေးတာ ကုန်ပြီဖြစ်လို့ နောက်တစ်ခု ထပ်ပြီး Delete ပါမယ်။                                                                 |
| <b>Delete</b>  | H, J    | အခုဆိုရင် ဖြုတ်လိုက်တဲ့ G နဲ့ ဆက်စပ်တာကို ဆက်သွားမှာ ဖြစ်ပါတယ်။ G နဲ့ ဆက်စပ်ပြီး မသွားရသေးတာ ကုန်ပြီဖြစ်လို့ နောက်တစ်ခု ထပ်ပြီး Delete ပါမယ်။ |
| <b>Delete</b>  | J       | အခုဆိုရင် ဖြုတ်လိုက်တဲ့ H နဲ့ ဆက်စပ်တာကို ဆက်သွားမှာ ဖြစ်ပါတယ်။ H နဲ့ ဆက်စပ်ပြီး မသွားရသေးတာ ကုန်ပြီဖြစ်လို့ နောက်တစ်ခု ထပ်ပြီး Delete ပါမယ်။ |
| <b>Delete</b>  |         | Queue empty ဖြစ်သွားပြီဖြစ်တာကြောင့် အကုန်သွားလို့ ပြီးသွားပြီ ဖြစ်ပါတယ်။                                                                     |

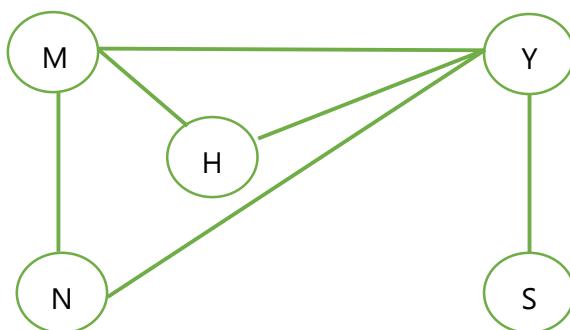
A, B, C, D, E, F, I, G, H, J ဆိုပြီးသွားခဲ့တာဖြစ်ပါတယ်။



လေ့ကျင့်ကြည့်ကြရအောင်။



1. DFS ဖြင့် visit လုပ်လိုရတဲ့ အစဉ်လိုက်ကို ထုတ်ပြပေးပါ။
2. BFS ဖြင့် visit လုပ်လိုရတဲ့ အစဉ်လိုက်ကို ထုတ်ပြပေးပါ။



3. DFS ဖြင့် visit လုပ်လိုရတဲ့ အစဉ်လိုက်ကို ထုတ်ပြပေးပါ။
4. BFS ဖြင့် visit လုပ်လိုရတဲ့ အစဉ်လိုက်ကို ထုတ်ပြပေးပါ။



## Chapter အနှစ်ချုပ်

- Robot တစ်ခုက နေရာတစ်ခုကနဲ့ goal ကို ရောက်အောင်သွားရမယ်။ အဲဒီလိုသွားတဲ့ အခါ ခန်းတွေ အများကြီးကို ဖြတ်ပြီးသွားရမယ်။ အဲဒီမှာ အခန်းလေးတွေဟာ ဘာနဲ့ တူသလဲ ဆိုရင် vertex နဲ့တူပါတယ်။ စတဲ့ အခန်းကနဲ့ ဆက်စပ်နေတဲ့ အခန်းတွေဖြတ်ပြီး goal ကို ရောက်အောင် သွားမှာဖြစ်ပါတယ်။ အီလို အခြေနေမျိုးမှာ DFS, BFS ကို သုံးလို့ရပါတယ်။
- အထူးသဖြင့်တော့ Game တွေမှာလည်း သုံးလို့ရပါတယ်။
- Beginner level ဖြစ်တာကြောင့် အလုပ်လုပ်ပုံကိုပဲရင်းပြုလိုက်ပါတယ်။



## အဆုံးသပ်

အခုစာအုပ်ဟာ Beginners တွေကို ရည်ရွယ်တာ ဖြစ်လို့ အခြေခံ သဘောတရားတွေ နဲ့ အမိက သိသင့်သိတိက်တာတွေကို ထည့်သွင်းရေးသားထားတာဖြစ်ပါတယ်။ ရှင်းလင်းချက်တွေ၊ လွှဲကျင့်ခန်းတွေကို ပုံမှန်လိုက်လုပ်ခဲ့ရင် အကျိုးတစ်စုံတစ်ရာတော့ ရရှိလိမ့်မယ်လို့လည်း မျှော်လင့် ပါတယ်။ ဖတ်တဲ့ သူ နားလည်စေဖို့ တတ်နိုင်သမျှ ကြိုးစားပြီး ရေးထားပါတယ်။

အဆုံးထိ ဖတ်ပေးတဲ့အတွက်လည်း ကျေးဇူးတင်ပါတယ်။ ဖတ်ရတာ အဆင်မပြေတာ၊ နားမလည်တာ တစ်စုံတစ်ခုရှိရင် ဝေဖန် အကြံပြုနိုင်ပါတယ်။ အကြံပြုချက်များကိုလည်း မျှော်လင့် နေပါတယ်။ တတ်နိုင်သမျှ ဖတ်ရသူ အကျိုးရှိအောင် အဆင်မပြေ တာတွေကို နောက်ကနေလိုက်ပြီး ဖြည့်ဆည်းပေးဖို့ ကြိုးစားသွားပါမယ်။

ဒီစာအုပ်ပြီးရင်တော့ အခုခေတ်ပြောပြောနေတဲ့ နေရာတကာ မှာ မပါမဖြစ် Data Science နဲ့ Artificial Intelligence (AI) တို့နဲ့ မှာပါဝင်တဲ့ Data ပုံစံအမျိုးမျိုး နှင့် အဲဒီ Data တွေပေါ်မှာ အလုပ်လုပ်တဲ့ data mining, machine learning အကြောင်းရေးပါအုံးမယ်။ အဲဒီ ဘာသာရပ်တွေဟာ ကျောင်းတက်တူန်းက အကြိုက်ဆုံး ဘာသာရပ်တွေဖြစ်ခဲ့ပြီး၊ Master Thesis ရော့၊ Ph.D. Thesis မှာပါ အဲဒီ Field နဲ့ လုပ်ခဲ့တာဖြစ်ပါတယ်။

ကျေးဇူးအထူးတင်စွာဖြင့်

Dr. Myat Mon Aye

9-5-2021

**Copyright 2021, Dr. Myat mon aye**

**All right reserved.**