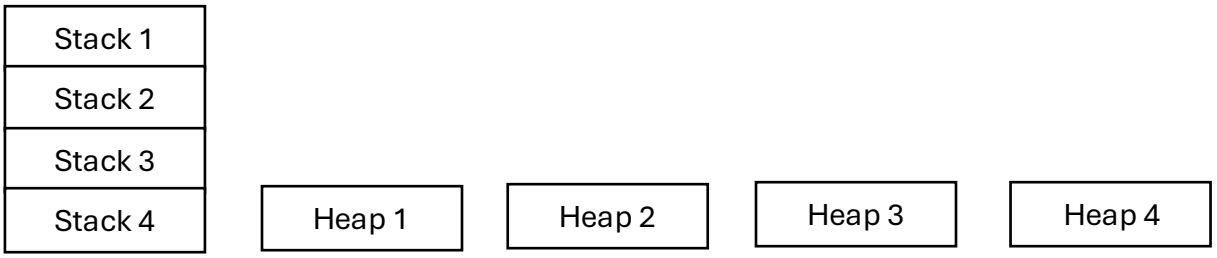
Lite förklaring

Jag har inget papper tillgängligt så skrev allt i Word, får införskaffa lite papper om det kommer fler liknande uppgifter. Jag märkte efter jag läste alla frågor igen att jag kanske missuppfattade några uppgifter och skrev saker i kod som skulle skrivits på papper? Oavsett så är det för sent för mig att ändra dem sakerna nu men är det något som är oklart eller gjort på fel sätt är det bara att skriva om ni vill att jag ska göra om eller ändra saker.

Fråga 1.

Stacken är som det låter, en stack. Allt staplas som boxar på varandra och man använder den översta boxen. För att komma åt de undre boxarna måste de övre flyttas. Allokering och rensning sker automatiskt. Heapen är annorlunda då den inte gör några staplar, den lägger snarare ut alla boxar bredvid varandra. Den är mycket mer dynamisk då man alltid kan komma åt allt minne. Den rensar dock inte minnet själv så man måste oroa sig för Garbage Collection.

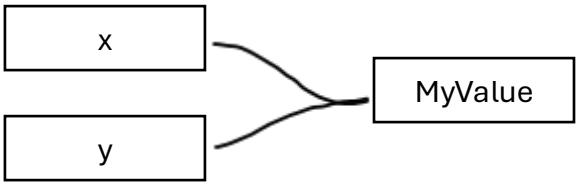


Fråga 2.

Value Types är typer som kommer från System.ValueType. Några exempel på Value Types är bool, char, int och double. Reference Types ärver av System.Object och är class, interface, object, delegate eller string. Skillnaden mellan dessa är att Reference Types alltid lagras i Heapen medan Value Types lagras där de deklareras.

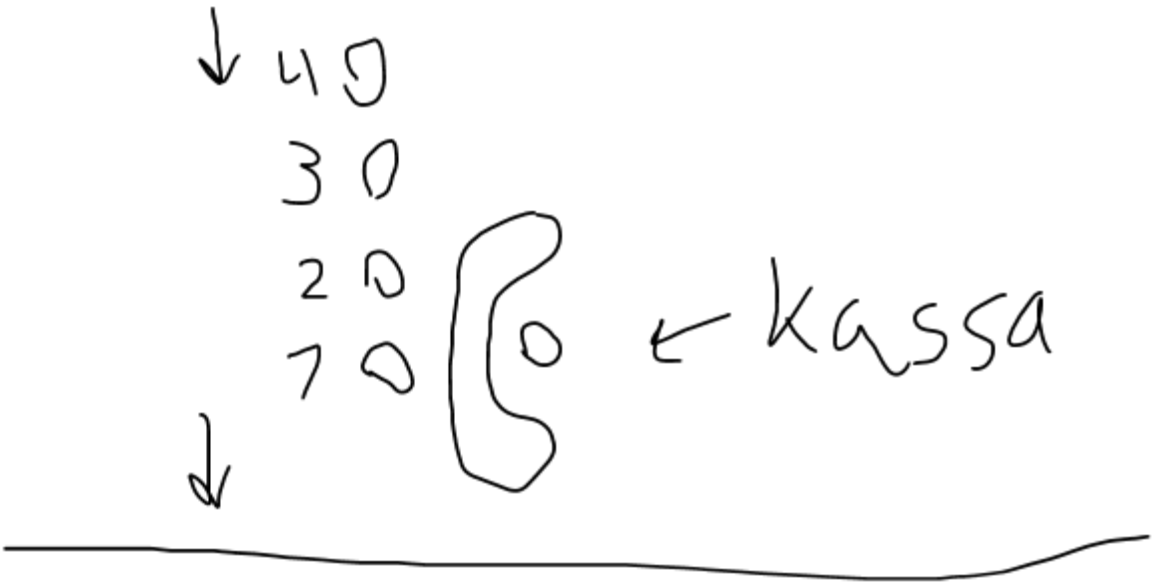
Fråga 3.

x och y refererar till samma ställa på Heapen och därför blir x.MyValue 4 när y.MyValue sätts som 4.

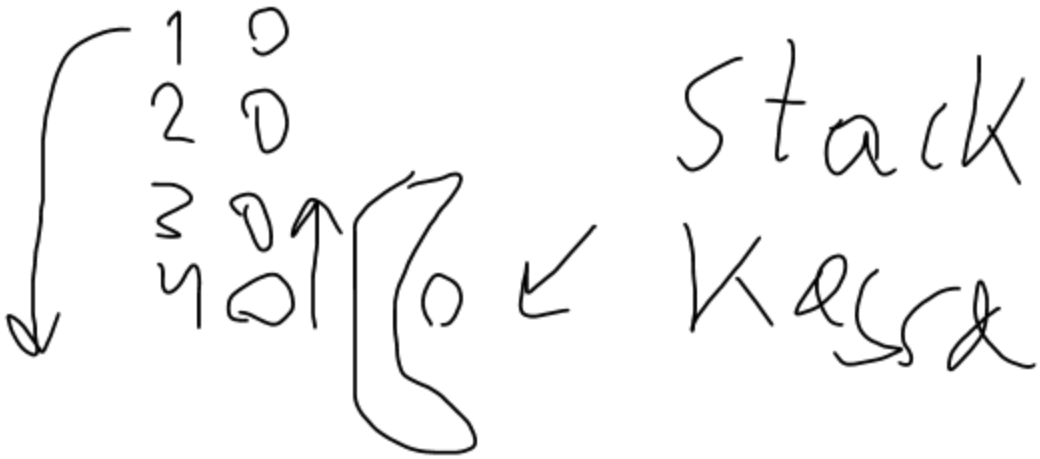


ICA KÖ

1. Kassan öppnar
2. Kalle ställer sig i kön
3. Greta ställer sig i kön
4. Kalle blir expedierad och lämnar kön
5. Stina ställer sig i kön
6. Stina blir expedierad och lämnar kön
7. Olle ställer sig i kön



Att använda en stack gör kön ganska konstig. Den som är först till kassan borde inte lämna sist. Slutar det inte komma folk in i kön får den första personen aldrig lämna.

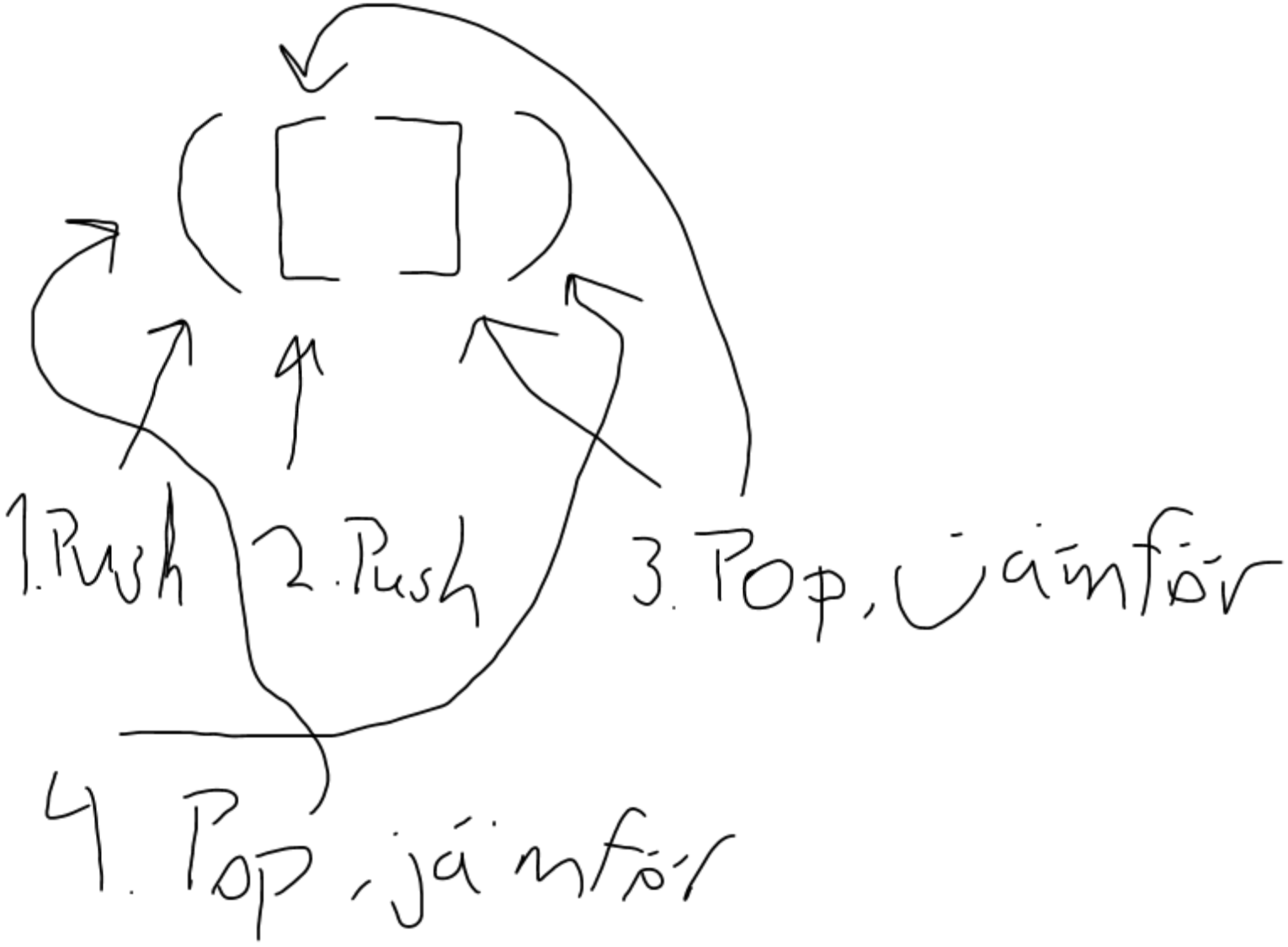


CheckParenthesis()

Om användaren matar in en sträng med parenteser ska programmet kolla och jämföra dem. Först är det värt att se vilket håll första parentesen är riktig åt. Ser den ut så här,) kan man direkt avsluta och säga att den är fel.

{[]}

På parenteserna ovan startar man med att hitta den första, i det här fallet {. Sedan hittar men den andra [och dem båda sparar man med hjälp av Push(). När dem kommer en parentes åt andra hållet som] jämför man den med den senaste inlagda parentesen. Detta med hjälp av Pop(). Om dem stämmer överens kan man fortsätta och leta efter fler par, annars säger man att det vart fel.



Rekursion

Vet inte om jag är helt 100 på hur det fungerar men förklarar vad jag tror och kommit fram till.

```
static int RecursiveOdd(int n)
{
    if (n == 1)
    {
        return 1;
    }
    return (RecursiveOdd(n - 1) + 2);
}
```

Om man till exempel matar in talet 3 i n så kommer RecursiveOdd anropa sig själv och minska n med 1 tills n är 1. Efter det returneras 1 först och sen returneras alla andra (RecursiveOdd(n – 1) + 2) där deras värden läggs in i (n-1) parentesen. Alltså den andra returneringen blir (1) + 2 = 3 och den sista returneringen blir (3) + 2 = 5. Hur många gånger detta sker har att göra med hur många gånger n – 1 behövde ske innan n blev 1.

Med denna logik bör RecursiveOdd(5) returnera talet 9. Först returneras 1 sedan (1) + 2 = 3, (3) + 2 = 5, (5) + 2 = 7, (7) + 2 = 9.

int 3: if = false → 3-1 → if = false → 2-1 → if = true, return 1 → return 1+2 (3) → return 3+2 (5)

Iteration

Om man vill ha till exempel femte positiva talet skriver man InterativeEven(5), därefter körs koden. Först skapar programmet int result och sätter den till 2, det första talet. For loopen börjar sedan köras på ett liknande sätt som med rekursionen fast lite simplare.

int 5: result = 2 → i (0) < 5 – 1, 2 += 2 → i (1) < 5 – 1, 4 += 2 → i (2) < 5 – 1, 6 += 2 → i (3) < 5 – 1, 8 += 2 → i (4) < 5 – 1, 8 += 2 → return 10;

FRÅGA: Interation och Rekursion

Jag antar att iteration är bättre att använda för minnet. När jag körde rekursion exemplet lyckades jag få ett litet fel som hette ”stack overflow” eftersom rekursionen kallar på sig själv läggs det till mer och mer minne i stacken och om en kodare gör fel och lyckas göra loopen infinite så tar minnet slut.