الباب الثالث

الإعادة (recursion):

فكرة الريكيرجين هي ان الفانكشن بتاعتي بتنادي على نفسها تاني و ده بيفيدني اكتر في تقسيم وتبسيط المشكلة الكبيرة بتاعتي لمجموعة مشاكل صغيرة و ده هيخلي مشكلتي ابسط في الحل.

في الباب ده هنتعرف على الريكيرجين بشكل مفصل شوية وهنعرف ايه هو ال call stack وازاي عملية ان فانكشن بتنادي نفسها اكتر من مرة الكمبيوتر بيتعامل معاها.

الريكيرجين:

دلوقتي لنفترض أنك عايز تكتب كود يعد تنازلي من رقم معين انت تحدده لحد ما يوصل لصفر فا الحل الي هيجي في بالك اول حاجة انك تستخدم loopسواء كانت while او for تلف لحد ما توصل ل 0 و ده ممكن يكون شكله (الكود ده هيعد من 20 لحد 1 يعني 0 مشه هتطبع عندنا ك output)

```
int n = 10;
while (n) {
    cout << n << endl;
    n--;
}
```

طيب احنا لو حبينا نكتب نفس الكود بس المرادي باستخدام الريكيرجين هيكون عامل ازاي ده هيكون شكله طيب عشان نفهم اكتر الكود هنا الكود بيعمل ايه

```
void recursiveCountdown(int n) {
    if (n == 0) {
        return;
    }
    cout << n<<endl;
    recursiveCountdown(n-1);

}
```

محتاجين نتعرف على كلمتين جداد ال base case وال recursive case

Base case:

هي الحالة الي عندها الكود بتاعنا هيقف زي الكوندشن بتاع اللوب و لو انا مقدرتش احدد ال base case بتاعتي صح فا ده معناه ان ال loop بتاعتي مش هتحلص و هخش في infinite loop.

Recursive case:

هي الحالة انا فيها هفضل مكمل شغل ومش هقف

في الحالة بتاعتي ال base case هي أنى أوصل لصفر اول ما أوصل لصفر فا انا عايز عملية ان الفانكشن عمالة تنادي نفسها تقف فا بعمل return عشان انهي ال loop دي و ال recursive case هي اني لسه

موصلتش للصفر فا طول ما انا موصلتش الصفر انا عايز انقص الرقم بتاعي واحد و اشوف لو هو مش صفر اطبعه لو صفر هقفل الloop.

طيب دلوقتي احنا بدأنا نفهم فكرة الريكرجين لكن ليه بنستخدمه مع ان مصلا في المثال الي احنا لسه شايفينه ال loop كانت أسهل والكل عارفها والكود بتاعها أسهل انه يتقري ويتفهم بس الريكيرجين مستخدم في الجوريزمات كتير و يستحسن انك كمبرمج تكون فاهمه عشان كده بيكون مهم انك تتعلمه.

دلوقتي هنشوف ازاي الكمبيوتر بيتعامل مع موضوع أنك عمال تنادي فانكشن كزا مرة جوا بعض او أصلا هو الكمبيوتر بيعمل ايه لما انت بتنادي فانكشن عادية من غير ريكيرجين و ده هياخدنا لل call stack قبل ما نقول ايه هو ال call stack هنعدي على مفهوم ال stackلوحده الأول.

:Stack

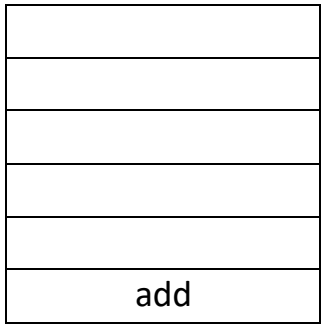
هو طريقة بخزن بيها البيانات بتاعتي (data structure) خلينا دلوقتي نقول ان معاك كتاب فا انت حطيته عالمكتب و بعد كده جبت كتاب جديد فا حطيته فوق الكتاب الي على المكتب و هكذا لحد ما بقا عندك 100 كتاب فوق بعض فا كده انت لو حبيت تاخد كتاب من ال100 كتاب بتوعك اجباري هتاخد الكتاب الي على الوش الي هو اخر كتاب انت حطيته و لو حبيت توصل لأول كتاب معاك هتحتاج تشيل كل الكتب الي فوقيه الأول و هو ده الstack لما بستعمل الstack بخضع لشوية شروط اول شرط اني متاحلي اعمل عملية من تلاتة فقط 1- اضيف عنصر جديد و بالتالي هيكون هو العنصر الاو عندي 2- امسح العنصر الاولا الي عندي 3- اروح استخدم العنصر الأول (الأول هنا معناها اخر عنصر انا ضيفته للstack)

الشرط التاني ان ترتيب خروج العناصر عندي من الstack هيكون عكس ترتيب دخولها بمعنى ان اخر عنصر دخل هو اول عنصر هيخرج.

ده كان شرخ مبسط للstack فا ايه هو الcall stack هو عمليا عبارة عن stack بيشيل كل ال functionsالي انت ناديت عليها.

:Call stack

دلوقتي نفترض انك كنت شغال على كود و ناديت فانكشن اسمه add و هنفترض ان ده ال stackبتاعنا فا ال stack حاليا هيكون شكله كده



في الوضع الطبيعي لو انت ناديت add و addمن جواها مش بتنادي على functionتانية فا الي هيحصل انها هتتحط في ال call stackو تتنفذ و بعد ما تخلص هتتحذف من ال call stack طب لو افترضنا ان add جواها بتنادي على function تانية اسمها display فا ساعتها ال call stack بتاعي هيكون عامل كده

display
add

الي هيحصل هنا ان اول ما addتنادي على displayالاولوية هتتنقل ل display عشان زي ما في ال stack بتعامل مع العنصر الي فوق طيب و add ايه الي هيحصلها هتقف عند اخر حاجة حصلت قبل ما تنادي على display لحد ما display تخلص و تتحذف من ال stack فا يبقى ده شكل ال stack بتاعنا

add

وهنا هترجع addتكمل شغل تاني من مكان ما وقفت لحد ما تخلص وتتحذف هي كمان من الstack

طب عايزين نشوف مثال العد التنازلي بتاعنا هيشتغل ازاي نفترض ان احنا هنعد المرادي من 5 لحد 1

شكل ال stack هيكون كده

recursiveCountdown(1)
recursiveCountdown(1)
recursiveCountdown(2)
recursiveCountdown(3)
recursiveCountdow(4)
recursiveCountdown(5)

هنا الي هيحصل ان اول مرة انت هتنادي على ال functionفا هتتزود في ال stackو تروح تطبع الرقم و بعدين تنادي نفسها تاني برقم اقل فتتنتقل الأولوية للcall الجديدة و هكذا لحد ما نوصل لل base caseفهنا تنتهي عندنا عملية الريكيرجن و تبدأ كل فانكشن اتنادى عليها تخلص شغلها و تتحذف من الstack

ملحوظات:

لو لاحظت ال functionبتاخد intاسمه n فا بالرغم من المتغير بالنسبة لكل مرة انت ناديت فيه ال function لسه اسمه n بس كل function عندها نسختها الخاصة من المتغير ده بحيث متأثرش عليه في function تانية يعني لو انت رحت غيرت قيمة ال n في مرة من المرات فا ده مش هيغير من قيمة ال n في بقيت ال functions يعني لو غيرنا في شكل ال function خليناها كده

```
void recursiveCountdown(int n) {
    if (n == 0) {
        return;
    }
    cout << n<<endl;
    recursiveCountdown(n-1);
    cout << n << endl;

}
```

```
5
4
3
2
1
1
2
3
4
5
```

هيكون ده شكل ال output

و ده بيوريك ان قيمة ال n

في كل كول متأثرتش بالي بعدها

الا بقي لو انت خاولت تعمل

التعديل ده بشكل صريح

زي أنك تعمل فانكشن

بترجع intو تساوي n

بال returnبتاع ال callالي بعدها مثلا

تاني نقطة التعديل الي عملناه فالكود ده مخدتش بالك من نقطة فيه؟

لو انا حبيت أخلى ال function دي تعد تصاعدي كل الي هحتاج اعمله اني اخلي ال callبتاع ال functionقبل الطباعة و هيكون ده شكل الكود

```
void recursiveCountdown(int n) {
    if (n == 0) {
        return;
    }
    recursiveCountdown(n-1);
    cout << n << endl;

}
```

ودي من المميزات القوية لي الريكيرجين انه اه طريقة كتابته مش واضحة اوي بس هو هيخليك انت كا مبرمج قادر انك تفهم و تستوعب اكتر ازاي الكمبيوتر بيتعامل مع الكود بتاعك

احر ملحوظة هسيب شوية functionsبسيطة في البروجيكت الموجود بتستعمل ال recursionفي عمل شوية حاجات بسيطة عشان تساعد اكتر في فهم الريكيرجين