06. 자료구조와 알고리즘

01. 재귀

학습목표

- 재귀 함수를 제작할 수 있다.

개요

문제는 작으면 작을수록 해결하기가 편하다. 그래서 어떤 문제를 해결할 때는 충분히 해결할 수 있을 정도로 작게 나눈 후 접근하는 것이 일반적이다. 이번 시간에는 재귀를 통해 **분할 정복법**(Divide & Conquer)에 대해 알아보도록 하자.

재귀

재귀(Recursion)는 함수가 자기 자신을 호출하는 것을 말한다. 재귀 호출을 사용하는 이유는 문제에 따라 전체를 한 번에 해결하기보다 같은 유형의 하위 작업으로 분할하여 작은 문제부터 해결하는 방법이 효율적이기 때문이다. 다시 말해 복잡한 알고리즘을 단순하고 알기 쉽게 표현할 수 있다. 그뿐만 아니라 알고리즘 자체만으로는 얼마나 많은 단계를 깊이 들어가야 하는지 알 수 없을 때에도 사용할 수 있다. 재귀의 구조는 재귀를 중단시키는 **기저 조건**(Base Case)과 기저 조건으로 수렴하게 되는 **재귀 조건**(Recursive Case)으로 구성된다.

예시를 살펴보자. 어떤 디렉토리가 갖고 있는 모든 파일 및 디렉토리를 지우는 함수를 구현한다고 해보자. 디렉토리에는 하위 디렉토리가 있을 수 있고, 그 하위 디렉토리에는 또 다른 하위 디렉토리가 있을 수 있으므로 같은 유형의 작업으로 분할할 수 있다.

```
void RemoveAll(directory):
```

```
// 재귀 조건
for childDirectory in childDirectories:
    RemoveAll(childDirectory)

// 기저 조건
모든 파일을 지운다;
```

다만 재귀는 결국 함수를 호출하는 것이므로 함수 호출에 따른 **오버헤드**(Overhead)*가 있다. 또, 재귀가 너무 깊어지면 그만큼 호출 스택의 공간을 더 많이 사용하게 되고, 끝내는 **스택 오버플로**(Stack Overflow)**가 발생할 수 있다.

*오버헤드 : 어떤 처리를 하기 위해 들어가는 간접적인 처리 시간 · 메모리 등을 말한다.

**스택 오버플로 : 스택이 가용할 수 있는 공간을 벗어나는 것을 말한다.

반복문과의 변환

모든 반복문은 재귀로 변환할 수 있다.

```
// 1부터 10까지 출력하고 싶다고 할 때

// #1. 반복문
for (int i = 1; i <= 10; ++i)
{
  Console.WriteLine(i);
}

// #2. 재귀
void DoRecursive(int i) // 반복 변수가 매개변수가 되었다.
{
  // 기저 조건
  // for문의 조건식을 탈출 조건으로 바꿔놓았다는 것을 볼 수 있다.
```

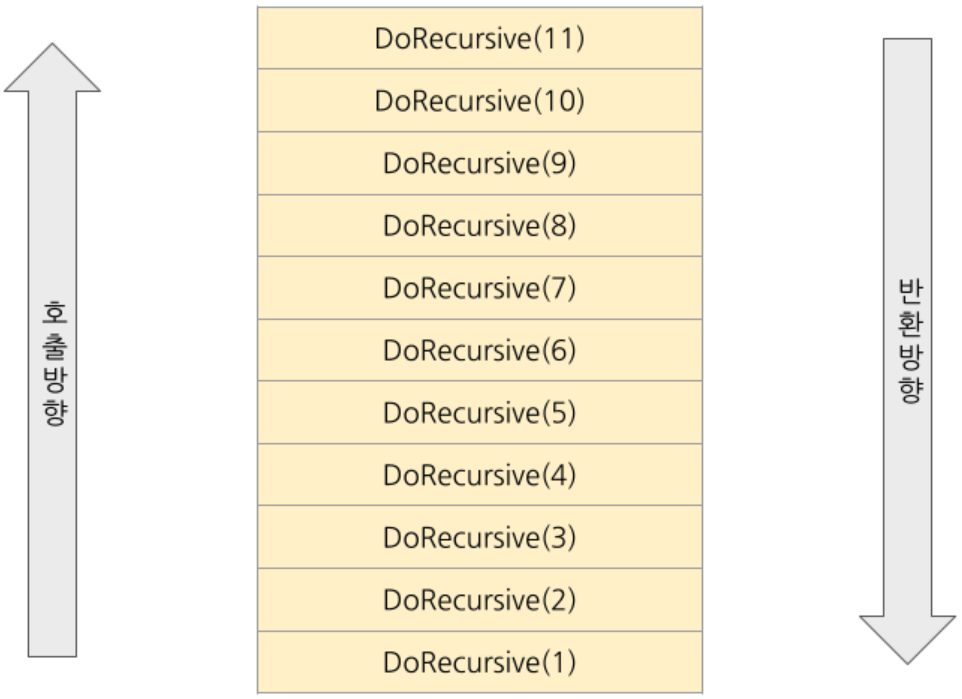
```
    if (i > 10)
    {
        return;
    }

    // for문에 바디에 있던 구문이다.
    Console.WriteLine(i);

    // 재귀 조건
    // 루프식에 있는 ++i 대신 i + 1을 해 다시 자기 자신을 호출한다.
    DoRecursive(i + 1);
}

// 반복문에서 1부터 시작했기 때문에 1을 호출한다.
DoRecursive(1);
```

함수의 동작을 그림으로 나타내면 아래와 같다.



재귀 함수의 동작

반대로 모든 재귀를 반복문으로 변환할 수 있는 것은 아니다. 반복문으로 변환이 불가능한 재귀도 존재한다. 실습 문제 6번의 경우가 그러하다.

재귀를 이용해 10진수를 2진수로 출력해보기

재귀를 이용해 입력 받은 10진수를 2진수로 출력해보자.

```
class Program
{
  static void Main()
  {
    // 10진수를 입력 받는다.
    int decimalNumber = int.Parse(Console.ReadLine());


    // 2진수로 출력한다.
```

```
    PrintToBinary(decimalNumber);
  }

  static void PrintToBinary(int number)
  {
    // 주어진 번호가 1이면 끝나게 된다.
    if (number == 1)
    {
      Console.Write(1);

      return;
    }

    // 2로 나눈 몫과 나머지를 구한다.
    int quo = number / 2;
    int rem = number % 2;

    // 몫에 대해서는 다시 한 번 과정을 진행해야 한다.
    PrintToBinary(quo);

    // 나머지를 출력한다.
    Console.Write(rem);
  }
}
```

동적 계획법

동적 계획법(Dynamic Programming)은 해결했던 문제에 대한 해답을 저장해두고 이후에 다시 활용하는 것이다. 즉, 재귀에 룩업 테이블을 결합한 형태라고 할 수 있다. 다음은 피보나치 수열을 재귀와 동적 계획법으로 푼 코드의 차이점이다.

```
class Program
{
  static void Main()
  {
    // 매우 오래걸린다. 이전에 계산했던 값을 다시 계산하기 때문이다.
    FibonacciRecursive(20);

    // 점화식에 따라 1번 항과 2번 항의 결과를 저장한다.
    _memo[1] = 1;
    _memo[2] = 1;

    // 이전에 계산한 값을 저장해두었기 때문에 훨씬 빠르다.
    FibonacciDp(20);
  }

  // 피보나치 수열을 재귀적으로 구현한 것이다.
  static void FibonacciRecursive(int n)
  {
    if (n <= 1)
    {
      return n;
    }
```

```
    return FibonacciRecursive(n - 1) + FibonacciRecursive(n - 2);
  }

  // 룩업 테이블을 만들어 둔다.
  // 이 룩업 테이블에는 이전에 해결했던 문제의 해답이 저장되어 있다.
  // 즉, _memo[5]에는 피보나치 수열의 5번 항이 저장되어 있다.
  // 이를 메모이제이션(Memoization)이라고 한다.
  static int[] _memo = new int[100]; // 단, 이 경우 99번까지의 항만 저장할 수 있다.

  static void FibonacciDp(int n)
  {
    // 이전에 해결했던 문제라면 저장된 결과를 사용한다.
    if (_memo[n] != 0)
    {
      return _memo[n];
    }

    // 해결하지 못한 문제라면 계산 결과를 저장한다.
    _memo[n] = FibonacciDp(n - 1) + FibonacciDp(n - 2);

    return _memo[n];
  }
}
```

더해보기

1. 아래의 문제를 풀어보세요.
    a. 10872번: 팩토리얼
    b. 10870번: 피보나치 수 5
    c. 17478번: 재귀함수가 뭔가요?
    d. 24060번: 알고리즘 수업 - 병합 정렬 1
    e. 11729번: 하노이 탑 이동 순서
    f. 2447번: 별 찍기 - 10
    g. 24416번: 알고리즘 수업 - 피보나치 수 1
    h. 9184번: 신나는 함수 실행
2. 병합 정렬(Merge Sort)은 분할정복법을 이용한 대표적인 알고리즘입니다. 여기를 참고하여 병합 정렬을 공부하고 아래의 코드 베이스를 완성시켜보세요.
    a. 코드를 따라치기보다 전체적인 개념을 이해한 후 구현해보려고 해보세요.

```
public class Program
{
  public static void Main()
  {
    int[] numbers = { 3, 7, 8, 5, 2, 1, 9, 5, 4 };
    MergeSort(numbers);

    foreach (var number in numbers)
    {
      Console.Write($"{number} ");
    }
  }

  public static void MergeSort(int[] numbers)
```

```
  {
    // TODO: 병합 정렬 구현하기
  }
}
```

참고자료

- C로 배우는 쉬운 자료구조 | 이지영 - 모바일교보문고