함수와 재귀

01

함수

02

프로세스
메모리

03

변수

04

재귀와 꼬리재귀

함수란?

1) 프로그래밍에서 함수(function)란 하나의 특별한 목적의 작업을 수행하기 위해 독립적으로 설계된 프로그램 코드의 집합이다.

C언어에서 함수는 크게 표준 함수와 사용자 정의 함수로 구분할 수 있다.

-표준 함수 : C에서 기본적으로 제공하는 함수
-사용자 정의 함수 : 사용자가 직접 정의해서 사용하는 함수

함수가 왜 필요할까?

1) 함수를 사용하는 이유는 바로 반복적인 프로그래밍을 피할 수 있기 때문이다.
=>프로그램에서 특정 작업을 여러 번 반복해야 할 때는 해당 작업을 수행하는 함수를 작성하면 된다.
필요할 때마다 작성한 함수를 호출하면 해당 작업을 반복해서 수행할 수 있다.

2) 전체적인 코드의 가독성이 좋아집니다.
=>손쉽게 유지보수를 할 수 있습니다.

함수 정의 방법[1]

```
return-type function-name(argument declarations)

{
    declarations and statements
}
```

- return-type은 함수가 실행을 끝나고 리턴되는 값의 데이터형을 말한다.

- function-name은 함수의 이름이다.

(이는 프로그램이 함수를 호출할 때 필요한 식별자이다.)

- argument declarations는 인수들의 선언으로 구성되어 있다.

- declarations and statements는 다음과 같이 구성됩니다.
<0개 이상의 변수 선언문들, 작업을 수행하기 위한 0개 이상의 statement들>

-함수 정의시 {declarations and statements}를 묶어 몸체(body)라고도 합니다.

함수 정의 방법(2)

```
int sum(int x, int y)
{
        int res;
        res = x + y;
        return res;
}
```

1. 위 함수의 함수명은 sum이다.
2. 함수의 자료형이 int 이므로 결과값이 정수값을 반환한다는 것을 알 수 있다.
3. 함수에서 더하고자 하는 두 정수를 전달받기 위해 두 개의 정수 변수 x, y를 인수로 선언했다.

프로세스 메모리 영역(1)

1) 텍스트 영역 - 프로그램의 실행 코드가 존재하는 영역

2) 데이터 영역
- 전역 변수나 정적 변수(Static)등이 이곳에 기록.
- 데이터 영역 내에서도 읽기 전용 영역, 읽기/쓰기 영역, 초기화가 되지 않은 영역으로 구분

프로세스 메모리 영역(2)

3)스택 영역

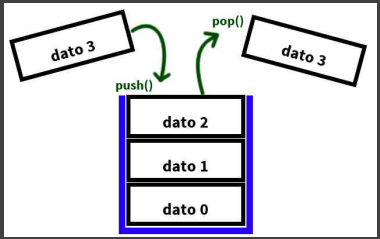
- 스택(stack) 영역은 함수의 호출과 관계되는 지역 변수와 매개변수가 저장되는 영역.

- 스택 영역은 함수의 호출과 함께 할당되며, 함수의 호출이 완료되면 소멸한다.

- 스택 영역에 저장되는 함수의 호출 정보를 스택 프레임이라고 한다.

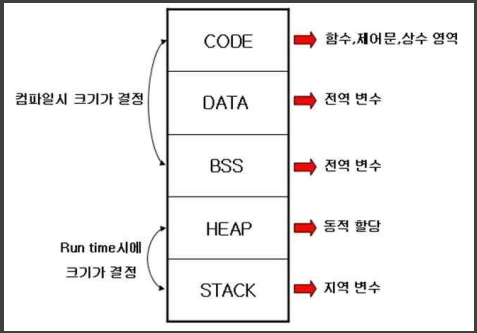
- 스택 영역은 푸시(push) 동작으로 데이터를 저장하고, 팝(pop) 동작으로 데이터를 인출한다.이러한 스택은 후입선출(LIFO, Last-In First-Out) 방식에 따라 동작하므로, 가장 늦게 저장된 데이터가 가장 먼저 인출된다.

- 스택 영역은 메모리의 높은 주소에서 낮은 주소의 방향으로 할당된다.



프로세스 메모리 영역(3)

4) 힙 영역:
메모리의 힙(heap) 영역은 사용자가 직접 관리할 수 있는 메모리 영역이다.
힙 영역은 사용자에 의해 메모리 공간이 동적으로 할당되고 해제된다.
힙 영역은 메모리의 낮은 주소에서 높은 주소의 방향으로 할당된다.



스택의 특징

-매우 빠른 액세스

-변수를 명시적으로 할당 해제 할 필요가 없다.

-공간은 CPU에 의해 효율적으로 관리된다.

- 메모리는 단편화되지 않는다.

-스택 크기 제한 (OS에 따라 다름)

-변수의 크기를 조정할 수 없다.

힙의 특징

-변수에 전역적으로 액세스 할 수 있습니다.

-메모리 크기 제한 없음

-(상대적으로) 느린 액세스

-효율적인 공간 사용을 보장하지 못하면 메모리 블록이 할당 된 후 시간이 지남에 따라 메모리가 조각화되어 해제될 수 있습니다.

-메모리를 관리해야합니다 (변수를 할당하고 해제하는 책임이 있습니다)

-변수는 C언어 realloc()

realloc () 함수는 이전에 예약된 기억장치 블록의 크기를 변경합니다

지역 변수란?

- 지역변수란 이름과 같이 한 지역에서만 사용할 수 있는 변수
- 지역이란 {}에 의해 만들어지는 영역을 의미함
- 지역변수들은 해당 지역을 벗어나면 소멸

전역변수란?

- 반대로 어느 지역에서나 사용할 수 있는 함수
- 괄호 밖에 서술
- 시작과 동시에 메모리 공간에 할당되어 프로그램이 끝나기 전까지 존재
- 별도의 값으로 초기화하지 않으면 0으로 초기화

지역 변수를 사용하는 이유!

- 전역변수 남용시 추후 유지보수 어려움(어느 함수에서 전역변수를 사용하는지 파악하기 어려움)
- 지역변수와 전역변수의 이름이 같다면 지역변수를 우선적으로 접근
- 꼭 필요한 경우가 아니라면 지역변수 사용 권장

Static

- 정적 변수
- 전역이나 함수 내부 모두에서 사용 가능
- 변수를 선언할 때 앞에 static을 넣어주는 것이 사용 방법
- 선언시 초기화하지 않아도 0으로 초기화

-프로그램 시작 시 할당되고 끝날 때 파괴
=> 전역으로 선언하면 해당 소스파일에서만, 함수 내부에서 선언하면 해당 함수에서만 사용 가능

함수 내부에서 선언하면 함수가 여러 번 실행되어도 선언 및 초기화는 한번만 일어남.

재귀란?

```
int factorial(int num){
  int result = 1;
  for(int i = 1 ; i <= num ; i++){
      result = result * i;
  }
  return result;
}
```

<반복문>

```
int factorial(int n){
    if(n == 1) return 1;
    return n * factorial(n - 1);
}
```

<재귀>

#반복과 재귀의 가장 큰 차이는, 재귀는 스스로를 호출한다는 점에 있다!

재귀와 반복의 차이점!

무한 반복이 발생하는 경우 재귀는 CPU 크래시를 초래할 수 있지만, 반복은 메모리가 부족할 때가 되면 멈춘다.

- 재귀

종결 조건을 명시할 때 실수를 하면, 무한 재귀 호출이 생길 수 있다.
예를들어 조건이 false가 되지 않아 계속 함수를 호출하게 되면 CPU 크래시가 발생할 수 있다.

- 반복

반복자 실수(반복자 배정이나 증가) 혹은 종료 조건을 실수하면 무한 루프가 발생한다.
시스템 에러가 발생할 수도, 발생하지 않을 수도 있지만 프로그램은 확실하게 중지된다.

꼬리재귀란?

```
int factorial(int n){
    if(n == 1) return 1;
    return n * factorial(n - 1);
}
```

<재귀>

```
int factorialTail(int n, int acc){
    if(n == 1) return acc;
    return factorialTail(n - 1, acc * n);
}
int factorial(int n){
    return factorialTail(n, 1);
}
```

<꼬리 재귀>

팩토리얼 함수가 두개로 분리됨을 확인할 수 있다.

재귀를 사용하는 이유?

- 알고리즘 자체가 재귀적으로 표현하기 자연스러울 때, 가독성이 좋다.
- 변수 사용을 줄여준다.

꼬리 재귀를 사용하는 이유?.

재귀로 구현을 하고 싶은데, 재귀가 가지는 오버헤드 때문에 꺼려질 수 있다.

하지만 컴파일러가 꼬리재귀를 반복문으로 최적화하는 것을 이용한다면
오버헤드를 피하면서 재귀문을 작성할 수 있다.

요점은 **재귀 호출 이후 추가적인 연산을 요구하지 않도록** 구현하는 것이다.

#오버헤드란?
어떤 처리를 하기 위해 들어가는 간접적인 처리 시간 · 메모리 등을 말한다.

꼬리재귀 최적화 예시

```
int factorialTail(int n, int acc){
    if(n == 1) return acc;
    return factorialTail(n - 1, acc * n);
}
int factorial(int n){
    return factorialTail(n, 1);
}
```

<꼬리 재귀>

```
int factorialTail(int n){
    int acc = 1;

    do{
            if(n == 1) return;
            acc = acc + n;
            n = n - 1;
            } while(true);
}
```

<컴파일러의 판단>

THANK YOU!

사진 출처

https://phaphaya.tistory.com/24

https://lignux.com/pilas-de-datos/