|  |  |
| --- | --- |
| 이름 | 2171262 박소희 |
| 주제 | 쉽게 풀어쓴 자료구조 Chap 02. 순환 |
| 일시 | 2023. 03. 23 18:00 ~ 19:30 |

Chap 02 순환에 대해서 공부하였다.

먼저 순환의 대표적인 예시인 팩토리얼 문제를 풀어보았다.

//순환 팩토리얼 계산하기

#if 1

int factorial(int n)

{

if(n == 0) return 1; //종료조건:자기자신을 더이상 안 부르는 조건

else

return n\*factorial(n-1); //3\*2\*1...

}

int main(void)

{

printf("%d\n",factorial(5));

}

#endif

재귀함수 순환에 있어서 가장 중요한 것은 종료조건이다. 종료조건이 있어야 프로세서가 재귀함수의 끝을 인지하고 프로그램을 안정적으로 끝마칠 수 있다. 만약, 종료조건이 없다면 위 코드는 무한으로 함수를 호출하게 되고 프로세스도 계속 돌아가게 될 것이다.

위 코드에서 종료조건은 n == 0일 때 1을 리턴하면서 종료하는 조건이다.

팩토리얼의 특성상, n, n-1, n-2, …. 1, 0 순서로 줄어들면서 자기자신을 호출하게 되는데 n == 0이라면 자기자신을 더 이상 호출하지 않고 1을 반환하여 종료시키겠다는 뜻이다. 필자는 1을 반환하면 결과값이 1이 나오는 것이라고 생각했는데 재귀함수는 종료조건까지 자기자신을 호출했다가 반환시에는 호출의 역순으로 반환하기 때문에 제대로 된 숫자 120이 나온다는 것을 알게 되었다.

교재의 실습문제 13번을 풀어보았다.

#if 1

//문제 13번 1부터 n까지 더하기

int sum(int n)

{

if(n==1) return 1;

else return n+sum(n-1);

}

int main (void)

{

printf("%d\n",sum(5));

return 0;

}

#endif // 1

간단한 문제이며 1부터 N까지 더한 값을 출력하는 코드이다.

이 코드에서의 종료조건은 if(n == 1) return 1; 이라는 코드이다. N, n-1, n-2,…. 1이 되면 1을 반환하고 자기자신을 그만 부르겠다는 뜻이다. 이 코드 역시 마찬가지로 재귀함수는 자기자신을 종료조건까지 호출했다가 반환시에는 호출의 역순으로 값이 반환되기 때문에 1+2+3+4+5 의 값인 15가 반환된다.

교재 연습문제인 피보나치 문제도 풀어보았다.

#if 1

int fib(int n)

{

printf("fib(%d) is called\n",n);

if (n == 0) return 0;

if (n == 1) return 1;

return (fib(n-1)+fib(n-2));

}

int main(void)

{

printf("%d\n",fib(4));

return 0;

}

#endif // 1

함수 호출의 과정을 따라가보기 위해 print해보았다.

텍스트이(가) 표시된 사진

자동 생성된 설명

마찬가지로 자기자신을 계속 부르는 식으로 호출이 되다가 n 값이 1 또는 0의 값이 되었을 때 값을 호출의 역순으로 반환하면서 프로그램이 끝이 난다. 특이한 점은 반환 시에 fibo(n-1)+fibo(n-2)의 값을 반환한다는 점이다. 출력 결과를 확인해보면 fibo(n-1)이 종료조건을 만날 때까지 자기자신을 부른 뒤, fibo(n-2)가 동일한 방식으로 호출이 되고 2개의 값을 더한 값이 리턴이 된다는 것을 알 수 있다.

마지막으로 백준 25501 재귀의 귀재 문제를 풀어보았다.

#if 1

//백준 팰린트롬

//int c = 0;

char c[BUFSIZ];

idx = 0;

int recursion(const char \*s, int l, int r){

if(l >= r) //종료 조건

{

c[idx]++; return 1;

}

else if(s[l] != s[r]) //종료조건

{

c[idx]++; return 0;

}

else

{

c[idx]++; return recursion(s, l+1, r-1);

}

}

int isPalindrome(const char \*s){

//c = 0;

return recursion(s, 0, strlen(s)-1);

}

int main(){

int num = 0;

scanf("%d\n",&num);

char buff[BUFSIZ];

char siz[BUFSIZ];

for(int i = 0; i < num; i++, idx++)

{

gets(buff);

siz[i] = isPalindrome(buff);

}

for(int i = 0; i < num; i++)

{

printf("%d %d\n", siz[i], c[i]);

}

}

#endif // 1

recursion함수는 제공되어 있었고 필자가 해야 할 일은 recursion 함수 호출 횟수와 반환 값을 함께 출력하는 것이었다. palindrome이란 앞에서부터 읽은 문자열과 뒤에서 읽은 문자열이 동일한 문자열을 뜻한다. 예를 들어 AAA, ABBA, ABABA 등이다.

위 코드에서는 palindrome 문자열이면 1을 반환, 아니면 0을 반환한다.

recursion함수의 매개변수를 보면 첫번째 인자는 char 문자열, 두번째 인자는 0번째 인덱스, 세번째 인자는 문자열의 길이 -1의 값, 즉 마지막 인덱스 번호를 뜻한다.

함수 내부 로직을 살펴보면 자기자신을 호출할 때마다 요소 값이 같은지를 확인하고 다르면 바로 0을 리턴한다. 같으면 자기자신을 호출하는데 recursion(s, l+1, r-1)의 방식으로 호출하여 2개의 지역변수로 앞에서부터 인덱스를 늘려가는 동시에 뒤에서부터 인덱스를 줄여가는 방식이다. 여기서 문자의 값이 같으면 바로 종료하지 않은 이유는 문자열의 모든 값을 모두 유효성 검사를 하기 위해서이다. If(l>=r)의 의미는 인덱스 번호가 같다면 1을 리턴하면서 함수를 종료하겠다는 의미이다.

필자는 C언어를 공부하면서 이러한 대칭 문자열의 구조를 반복문을 활용해서 접근한 경험이 대다수였는데 재귀함수로 값을 도출할 수 있다는 점이 새롭게 다가왔던 것 같다.

메모리의 data영역에 전역변수 배열 c와 c의 인덱스를 나타내는 변수 idx를 정의하였다. stack영역에는 테스트 할 개수인 num, 문자열을 저장하는 배열 buff, palindrome 여부(1 또는 0의 값)을 저장하는 배열 siz를 정의하였다. 배열대신 heap공간에 동적할당을 통해 공간을 할당하고 프로그램 종료 직전에 남은 공간들을 소멸했다면 더욱 효율적인 메모리 관리가 될 것 같다.

