6

반복문 (for, while)

안녕하세요, 여러분. C 언어 공부는 잘 되가고 있나요? 제 사이트의 강좌 만을 보고도 충분히 C 언에 대한 깊은 학습이 이루어 질 수 있다고 생각하지만 사이트의 강좌 업데이트 속도가 느리니 답답하신 분들도 많을 것 입니다. 그러한 분들은 특별히 C 언어 관련 책을 서점에서 구매하여 읽어보는 것을 추천합니다.

물론 학원을 다녀도 되지만, 시중에 C 언어에 관련한 훌륭한 학습서 (흔히 가장 많이 추천하는 것으로는 열혈강의 C 프로그래밍, A Book On C, Teach Yourself C, 등등) 들이 많이 있으므로 굳이 학원을 다닐 필요가 없을 것이라 생각합니다. 또한 인터넷을 통해서도 C 언어에 관해 많은 정보를 알 수 있으니 이점 유의하시기 바랍니다.

3일전에, Psi 는 친구로 부터 1 부터 100 까지의 합을 구해달라는 요청을 받았습니다. Psi 는 수학자 가우스 처럼 똑똑하지가 못하기에, 등차수열의 합을 구하는 방법을 알지 못했습니다. 하지만 Psi 는 이 블로그에서 C 언어를 통해 계산하는 법을 알았으므로 이를 이용하기로 하였습니다. 그래서, 그는 다음과 같이 30분 동안 열심히 타이핑 하여 아래와 같은 프로그램을 만들었습니다.

```c
#include <stdio.h>
int main() {
  printf("%d", 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 + 11 + 12 + 13 + 14 + 15 +
                16 + 17 + 18 + 19 + 20 + 21 + 22 + 23 + 24 + 25 + 26 + 27 +
                28 + 29 + 30 + 31 + 32 + 33 + 34 + 35 + 36 + 37 + 38 + 39 +
                40 + 41 + 42 + 43 + 44 + 45 + 46 + 47 + 48 + 49 + 50 + 51 +
                52 + 53 + 54 + 55 + 56 + 57 + 58 + 59 + 60 + 61 + 62 + 63 +
                64 + 65 + 66 + 67 + 68 + 69 + 70 + 71 + 72 + 73 + 74 + 75 +
                76 + 77 + 78 + 79 + 80 + 81 + 82 + 83 + 84 + 85 + 86 + 87 +
                88 + 89 + 90 + 91 + 92 + 93 + 94 + 95 + 96 + 97 + 98 + 99 +
                100);
  return 0;
}
```

그리고, 무사히 계산 결과인 5050을 구해서 친구에게 알려주었습니다. 그런데, 이게 웬일입니까? 그 친구가 갑자기 1 부터 10000까지의 숫자의 합을 계산해 달라고 요청하는 것 아니겠습니까? 위 1

부터 100 까지 쓰는 것도 힘들어 죽겠는데 10000 까지 라니. Psi 는 눈 앞이 캄캄하였습니다. 적어도, 이 강좌를 보기 전 까지는 말이죠.

for 문 (for statement)

여러분은 컴퓨터가 왜 생겨났는지 아십니까? 물론 여러가지 이유가 있겠지만 그 중 제일 중요한 이유는 바로 계산 입니다.최초의 컴퓨터라고 알려진 ENIAC (물론 이에 대해 의견이 분분 하지만 가장 일반적으로 최초의 컴퓨터는 ENIAC 이나 영국의콜로서스 둘 중 하나이네요) 은 탄도의 발사표를 계산하는 역할을 하였습니다. 그렇다면 두 번째로 중요한 컴퓨터의 존재 이유는 무엇일까요? 바로, 노가다 - 즉 반복 연산 입니다.

예를 들어서, 1 부터 100 까지 곱한다고 칩시다. 인간은 지능이 있으므로 충분한 시간만 주어진다면 이를 수행할 수 있습니다. 단, 엄청난 짜증을 내겠지요. 그리고 '도대체 이런 계산을 내가 왜 하나?' 라는 생각도 들어 계산을 하다 말고도망갈 수 도 있습니다. 하지만 컴퓨터의 경우 그렇지 않습니다. 우리가 어떤 생-노가다 성 일을 시켜도 묵묵히 자기 일만합니다. 아무리 지겨운 연산 이라도 전기 조금 더 달라는 요구, 조금 쉬게 해달라는 요구도 없이 묵묵히 자기 일 만 할뿐이지요. 이 것이 바로 우리가 컴퓨터를 쓰는 두 번째 이유 입니다.

따라서, 이번 강좌에서는 반복문에 대해 중점적으로 알아보도록 하겠습니다. 반복문은 컴퓨터 상에서 상당히 많이 쓰이므로 반드시 이해하시기 바랍니다. 일단, C 언어에서 사용할 수 있는 반복문은 여러 종류가 먼저 있습니다만, 가장 먼저 널리 쓰이는 for 문에 대해 알아 보도록 하겠습니다.

```c
#include <stdio.h>
int main() {
  int i;
  for (i = 0; i < 20; i++) {
    printf("숫자 : %d \n", i);
  }

  return 0;
}
```

위 소스를 성공적으로 컴파일 하였다면,

실행 결과

```
숫자 : 0
숫자 : 1
숫자 : 2
숫자 : 3
숫자 : 4
```

```
숫자 : 5
숫자 : 6
숫자 : 7
숫자 : 8
숫자 : 9
숫자 : 10
숫자 : 11
숫자 : 12
숫자 : 13
숫자 : 14
숫자 : 15
숫자 : 16
숫자 : 17
숫자 : 18
숫자 : 19
```

와 같이 나옵니다.for 문은 다음과 같은 기본 구조를 가지고 있습니다.

```
for (/* 초기식 */; /* 조건식 */; /* 증감식 */) {
  // 명령1;
  // 명령2;
  // ....
}
```

일단, 각 부분의 역할이 무엇인지 알아 보도록 하죠. 초기식에서는 제어변수가 초기화 됩니다. 이 말은 즉슨, for 문은 반복문이고, 반복문은 얼마나 반복을 해야 될 지 알아야 합니다. 만약 반복문이 끊이지 않고 반복한다면 CPU 사용률을 100%로 끌어 올려 전력 낭비일 뿐 만이 아니라 코드 뒷부분이 실행되지 않아 여러 오류들이 발생될 수 있습니다.

따라서, C 언어에서는 반복문이 얼마나 반복해야 할 지를 알기 위해 '제어변수' 라는 것을 도입하였습니다. for 문으로 하여금 제어변수가 특정한 조건을 만족할 때 에만 반복을 계속하게 한다는 것입니다. 제어변수의 초기값은 for 문의 '초기식' 부분에서 지정 됩니다. 예를 들어서 내가 i 를 제어변수로 이용하다면 초기식에 i = 4; 가 되면 처음 i 의 값을 4 로 한다는 뜻이지요.

이제, for 문이 조건식을 봅니다. 조건식은 우리의 제어 변수인 i 가 만족해야 될 특정한 조건이 있습니다. 예를들어, i 의 값은 언제나 10 미만이라 던지 (i < 10), i 는 언제나 1 이상 이라던지 (i >= 1). for 문은 이러한 조건식이 참 일때에만 그 일을 수행합니다. 여기서 '그 일' 은 중괄호 속의 명령들을 실행한다는 뜻이지요.

마지막으로 증감식은 "1 회 실행 시 i 의 값을 어떻게 만들어야 되냐? " 가 나타나 있습니다. 예를 들어서 증감식에 i ++ 이 써 있다면 한 번 실행 할 때 마다, i 의 값을 1 증가 시킵니다.

마찬가지로 증감식 부분에 `i -= 2` 라면 한 번 실행 할 때 마다 2 씩 감소하겠네요.

매번 실행 할 때 마다, `for` 문의 증감식이 실행 되고, 그다음에 조건식을 체크 합니다. 만약에 조건식이 `i < 10` 이였고, `i` 의 값은 9 였다고 칩시다. 또한 증감식이 `i ++` 이였다면, 명령들을 실행 한 후, 증감식이 실행되어 `i` 의 값은 10 이 됩니다. 따라서, 조건식이 거짓이 되어 `for` 문을 빠져 나갑니다.

그렇다면 위의 소스 코드는 어떨까요?

```
for (i = 0; i < 20; i++) {
  printf("숫자 : %d \n", i);
}
```

우리가 컴퓨터라면, 일단 컴퓨터는 `for` 문을 보고,

> 음, `i` 의 값을 0 으로 해야 겠다. (초기식)

`for` 문에 `i < 20` 으로 되어 있으므로 (조건식)

> `i < 20` 이 맞나? 맞네.. 그럼 중괄호 속의 내용을 실행해야지. 숫자 0 출력!

또한, `for` 문에 `i++` 로 되어 있으므로 (증감식)

> 이제 `i` 의 값을 1 증가 시켜야 겠다.

따라서, `i` 의 값은 1 이 된다.

> i 의 값이 20 미만 인가? 어, 맞네. 그러면 한 번 더 실행 해야 겠다. (조건식) 숫자 1 출력

..... (생략) .....

> 이제 `i` 의 값을 1 증가 시켜야 겠다 (증감식)

20 번의 실행 후, `i` 의 값이 마침내 20 이 되었다.

> i 의 값이 20 미만 인가? 어? 아니잖아. (조건식) 그러면 이제 `for` 문을 빠져 나가야지

하며, 더이상 중괄호 속의 내용을 실행하지 않는다. 숫자 *20* 이 출력 되지 않는다.

`for` 문은 의외로 간단 합니다. 단지 기억 하실 것은 **`for` 문은 `{}` 안에 작업들을 조건식이 성립할 동안 반복해주는 것이고, 매 반복마다 증감식을 실행한다** 라고 이해 하시면 되겠습니다.

```c
/* 1 부터 19 까지의 합*/
#include <stdio.h>
int main() {
  int i, sum = 0;
  for (i = 0; i < 20; ++i) {
    sum = sum + i;
  }
  printf("1 부터 19 까지의 합 : %d", sum);

  return 0;
}
```

위 소스를 성공적으로 컴파일 했다면

실행 결과

```
1 부터 19 까지의 합 : 190
```

와 같이 나옵니다.

만약 위 결과를 믿지 못하는 사람들은 직접 계산기로 더하거나, 등차수열의 합 공식을 이용하여 직접 셈하셔 보시기 바랍니다. 아마, 독자 여러분들의 컴퓨터가 비정상이 아니라면, 아니면 당신의 눈이 잘못되지 않는 한 위 결과는 190 으로 나올 것 입니다.

일단, 위 프로그램의 핵심부분은 아래와 같습니다.

```c
for (i = 0; i < 20; ++i) {
  sum = sum + i;
}
```

for 문을 살펴보자면, 위 for 문은 총 20 회 실행되며 i 는 0 부터 19 까지의 값을 가집니다. 이 때 주목해야 할 부분은 바로

```c
sum = sum + i;
```

이 부분이죠. sum 이라는 변수에 i 의 값이 계속 더해집니다. 아시다 싶이 여러분은 sum = sum + i 라는 식의 뜻이 0 = i 라는 괴상한 방정식이 아니라 '=' 를 '대입 연산자' 로 생각하여 'sum 이란 변수에 sum + i 의 값을 집어 넣는다' 라는 의미가 됩니다. 즉, 위 상태로 for 문을 실행하게 되면 sum 에 0 부터 19 까지의 값이 더해지게 됩니다.

위 for 문을 보통 수식으로 풀어쓰면 아래와 같이 됩니다.

```
sum = 0;  // 초기 조건
sum = sum + 0;
sum = sum + 1;  // sum = 1;
sum = sum + 2;  // sum = 3;
sum = sum + 3;  // sum = 6;
// ....
sum = sum + 19;  // sum = 190;
```

이 되는 것이지요. 그렇다면 이제 Psi 의 고충을 풀어줄 시간이 왔네요. 1 부터 10000 까지의 합은 어떻게 구할까요? 그야 간단합니다. 단지 조건식만 약간 수정해 주면 됩니다. 한가지 걱정할 부분은 만약 10000 까지의 합이 `int` 자료형의 범위보다 크면 안되는데, 다행히도 크지 않으므로 그냥 계산하시면 됩니다. 이는 아래와 같습니다.

```c
#include <stdio.h>
int main() {
  int i, sum = 0;
  for (i = 0; i <= 10000; ++i) {
    sum = sum + i;
  }
  printf("1 부터 10000 까지의 합 : %d \n", sum);

  return 0;
}
```

그 결과는

실행 결과

```
1 부터 10000 까지의 합 : 50005000
```

와 같네요. 결국 Psi 는 친구와의 우정을 지킬 수 있었습니다. ㅎㅎ

```c
/* for 문 응용 */
#include <stdio.h>
int main() {
  int i;
  int subject, score;
  double sum_score = 0;

  printf("몇 개의 과목 점수를 입력 받을 것인가요?");
  scanf("%d", &subject);

  printf("\n 각 과목의 점수를 입력해 주세요 \n");
  for (i = 1; i <= subject; i++) {
```

```
    printf("과목 %d : ", i);
    scanf("%d", &score);
    sum_score = sum_score + score;
  }

  printf("전체 과목의 평균은 : %.2f \n", sum_score / subject);

  return 0;
}
```

위 소스를 성공적으로 컴파일 하였다면

실행 결과

```
몇 개의 과목 점수를 입력 받을 것인가요?4


 각 과목의 점수를 입력해 주세요
과목 1 : 100
과목 2 : 99
과목 3 : 89
과목 4 : 76
전체 과목의 평균은 : 91.00
```

음, 여러 과목을 입력해 보면서 실제 시험 성적 평균을 내보시기 바랍니다. 아무튼, 위 소스를 살펴봅시다. 일단, 가장 중요한 부분인 `for` 문 부분 부터 보자면...

```
for (i = 1; i <= subject; i++) {
  printf("과목 %d : ", i);
  scanf("%d", &score);
  sum_score = sum_score + score;
}
```

`for` 문을 살펴보면 `i` 의 값이 1 에서 `subject` 까지 1 씩 증가하면서 돌아가네요. 이 말은 즉슨, `for` 문 안의 내용이 `subject` 번 실행된다는 뜻입니다. (즉, `subject` 가 3 이라면, `i` 의 값이 1 부터 3 까지 1 씩 증가하면서 돌아가므로 `1,2,3.` 즉 3 번 `for` 문 속 내용이 실행됩니다)

이 때,

```
printf("과목 %d : ", i);
scanf("%d", &score);
```

위 부분에서 각 과목의 점수를 입력받고, 그 입력받은 점수를 `score` 라는 변수에 저장하게 되죠.

```
sum_score = sum_score + score;
```

그리고, 그 입력받은 score 를 sum_score 에 더하게 됩니다. 다시말해, for 문이 모두 돌아가고 나면 sum_score 에는 입력받은 과목들의 점수의 합이 들어가게 됩니다. 따라서,

```
printf("전체 과목의 평균은 : %.2f \n", sum_score / subject);
```

평균은 총점을 과목 수로 나눈 것이므로 sum_score / subject 가 우리가 구하고 싶은 전체 과목 평균이 되겠군요.

break 문

```
/* break! */
#include <stdio.h>
int main() {
  int usranswer;

  printf("컴퓨터가 생각한 숫자를 맞추어 보세요! \n");

  for (;;) {
    scanf("%d", &usranswer);
    if (usranswer == 3) {
      printf("맞추셨군요! \n");
      break;
    } else {
      printf("틀렸어요! \n");
    }
  }

  return 0;
}
```

성공적으로 실행했다면 아래와 같이 나오게 됩니다.

실행 결과

```
컴퓨터가 생각한 숫자를 맞추어 보세요!
5
틀렸어요!
6
틀렸어요!
```

```
7
틀렸어요!
8
틀렸어요!
3
맞추셨군요!
```

3 이 입력 될 때 까지 계속 물어보다가 3 을 입력하게 되면 프로그램이 종료됩니다 3 이 입력 될 때 까지 계속 물어보다가 3 을 입력하게 되면 프로그램이 종료됩니다.

```
for (;;) {
  scanf("%d", &usranswer);
  if (usranswer == 3) {
    printf("맞추셨군요! \n");
    break;
  } else {
    printf("틀렸어요! \n");
  }
}
```

일단, for 문을 잠시 살펴 봅시다. 그런데, 한 가지 이상한 점이 있죠? 초기식, 조건식, 증감식이 모두 없습니다! 그렇다면 이 for 문은 얼마나 실행 되야 되는 것인가요? 답은, for 문의 조건식이 명시되지 않는다면 항상 **참** 이라 인식 되기 때문에 이 for 문은 언제나 참이됩니다. 다시 말해, 무한히 중괄호 속의 내용을 실행한다는 것이지요. 그래서, 만약

```
#include <stdio.h>
int main() {
  for (;;) {
    printf("a");
  }
  return 0;
}
```

와 같은 프로그램을 만든다면, for 문이 무한번 실행되므로 (프로그램 자체가 강제적으로 종료되기 전 까지)

실행 결과

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

와 같이 나오게 됩니다.

```
scanf("%d", &usranswer);
if (usranswer == 3) {
  printf("맞추셨군요! \n");
  break;
} else {
  printf("틀렸어요! \n");
}
```

아무튼, scanf 를 통해 usranswer 에 사용자가 입력한 수를 저장합니다. 그리고 if 문을 통해 비교하지요. 과연 컴퓨터가 생각한 3 과 같은지... 만약 같다면 '맞추셨군요!' 가 출력이 됩니다. 그리고,프로그램이 종료되죠. 즉, for 문을 빠져나갑니다. 그런데, 맞추지 못하면 for 문은 계속 돌고 돌게 됩니다. 우리가 맞출 때 까지요. 그렇다면, 위 소스에서 for 문을 빠져나가게 하는 부분은 무엇일까요. 아마 짐작했던 대로, break 가 for 문을 탈출 시킵니다.

break 는 for 문에 조건식에 상관 없이 실행이 되기만 하면 for 문을 그대로 탈출 시켜 버립니다. 이 말은 즉슨 break 아래의 어떠한 것들도 실행이 되지 않는 다는 것이지요.

```
#include <stdio.h>
int main() {
  for (;;) {
    break;
    printf("a");
  }
  return 0;
}
```

따라서, 위와 같은 프로그램을 만들었을 때, break 문을 만나자 마자 for 문 밖으로 탈출 시키므로 a 는 출력이 되지 않고 프로그램은 종료됩니다. 반면에

```
#include <stdio.h>
int main() {
  for (;;) {
    printf("a");
    break;
  }
  return 0;
}
```

위와 같이 break 앞에 printf("a"); 가 있다면 a 가 출력이 되고 for 문을 빠져나가 종료가 되는 것이지요.

사실, 무한 for 문은 생소하기는 해도 많은 곳에서 쓰이고 있습니다. 예를 들어 어떤 게임에서

```
for (;;) {
  // 게임;
  if (/* 유저 사망 */) {
    if (/* 게임 다시 안할래요 */) {
      break;
    }
  }
  // 게임 재시작;
}
```

와 같이 쓰일 수 있습니다.

continue 문

continue 문은 break 문과 비슷하지만서도 하는 일은 완전히 다릅니다. continue 는 break 와는 달리 for 문을 빠져 나가지 않고, 그냥 패스 해주는 것입니다. 아래 예제를 봅시다.

```
/* 5 의 배수를 제외한 숫자 출력 */
#include <stdio.h>
int main() {
  int i;

  for (i = 0; i < 100; i++) {
    if (i % 5 == 0) continue;

    printf("%d ", i);
  }

  return 0;
}
```

성공적으로 실행하면

실행 결과

```
1 2 3 4 6 7 8 9 11 12 13 14 16 17 18 19 21 22 23 24 26 27 28 29 31
↪ 32 33 34 36 37 38 39 41 42 43 44 46 47 48 49 51 52 53 54 56 57
↪ 58 59 61 62 63 64 66 67 68 69 71 72 73 74 76 77 78 79 81 82 83
↪ 84 86 87 88 89 91 92 93 94 96 97 98 99
```

와 같이 나오게 됩니다. 보시다 싶이, 5 의 배수를 제외한 0 이상, 100 미만의 모든 수들이 출력되었습니다.

```c
for (i = 0; i < 100; i++) {
  if (i % 5 == 0) continue;

  printf("%d ", i);
}
```

일단, for 문을 살펴보면 i 가 0 부터 100 미만의 값을 가지게 됩니다. 이 때, if 문을 살펴 보면 i 를 5 로 나눈 나머지 (i % 5) 가 0 일 때 (== 0), continue 를 실행함을 볼 수 있습니다.

continue 는 break 문 처럼 아래 모든 내용을 무시한다는 점에서 동일하지만, break 문은 루프를 빠져나가는데 반면 continue 는 다시 조건 점검부로 점프하게 됩니다. continue 는 마치 카드게임에서 스킵과 같은 역할을 하게 됩니다. (break 문이 카드게임에서 퇴출 되는 것이라면...)

따라서, i 의 값이 5 의 배수인 경우에만 printf("%d", i) 가 실행이 되지 않게 되는 것이지요.

문득 for 문을 배우면서 이러한 생각은 들지 않았나요? if 문 안에 if 문을 넣을 수 있는 것 처럼 for 문 안에도 for 문을 넣을 수 있을까? 네, 물론 넣을 수 있습니다. 아래 예제를 참조하세요.

```c
/* 구구단 */
#include <stdio.h>
int main() {
  int i, j;

  for (i = 1; i < 10; i++) {
    for (j = 1; j < 10; j++) {
      printf(" %d x %d = %d \n", i, j, i * j);
    }
  }

  return 0;
}
```

성공적으로 컴파일 하였다면

실행 결과

```
.... (생략) ...
 7 x 8 = 56
 7 x 9 = 63
 8 x 1 = 8
 8 x 2 = 16
 8 x 3 = 24
 8 x 4 = 32
```

```
  8 x 5 = 40
  8 x 6 = 48
  8 x 7 = 56
  8 x 8 = 64
  8 x 9 = 72
  9 x 1 = 9
  9 x 2 = 18
  9 x 3 = 27
  9 x 4 = 36
  9 x 5 = 45
  9 x 6 = 54
  9 x 7 = 63
  9 x 8 = 72
  9 x 9 = 81
```

와 같이 근사한 구구단 표가 출력됩니다.

```
for (i = 1; i < 10; i++) {
  for (j = 1; j < 10; j++) {
    printf(" %d x %d = %d \n", i, j, i * j);
  }
}
```

위 코드에서 구구단 표를 출력하는 부분은 바로 위 부분입니다. `for` 문이 2 개나 사용되어 있는 꼴이지요. 그런데 사실 돌아가는 원리는 간단합니다. 일단, 처음에 `i` 에 1 이 들어 가게 되죠. 그런 다음에

```
for (j = 1; j < 10; j++) {
  printf(" %d x %d = %d \n", i, j, i * j);
}
```

이 부분이 열심히 실행됩니다. 물론 위 부분이 열심히 실행되는 동안 `i` 의 값은 변하지 않고 (계속 1 로 남는다), `j` 의 값만 1 부터 9 까지 변하여 구구단의 `1 x 1` ~ `1 x 9` 까지 출력하게 되는 것이지요. 위 `for` 문이 끝나면, 다시

```
for (i = 1; i < 10; i++)
```

이 부분이 실행되어 `i` 의 값이 1 증가합니다. 즉, `i` 는 2가 되는 것이지요. 이제 다시

```
for (j = 1; j < 10; j++) {
  printf(" %d x %d = %d \n", i, j, i * j);
}
```

가 실행되어 2 x 1 ~ 2 x 9 까지 출력되게 됩니다. 마찬가지 방법으로 i 의 값이 9 가 될 때까지 실행한 뒤 i 의 값이 10 이 되면 for 문을 완전히 빠져 나와 실행이 종료 됩니다.

```
/* 다음 소스만 보고 무슨 숫자가 출력될 지 맞추어 보세요~~ */
#include <stdio.h>
int main() {
  int i, j;

  for (i = 1; i < 10; i++) {
    for (j = 1; j < i; j++) {
      printf("%d ", j);
    }
  }

  return 0;
}
```

성공적으로 컴파일 하였다면

실행 결과

```
1 1 2 1 2 3 1 2 3 4 1 2 3 4 5 1 2 3 4 5 6 1 2 3 4 5 6 7 1 2 3 4 5
↪  6 7 8 %
```

가 나오게 됩니다. 아마 위에서 for 문에 대해 잘 이해하신 분들은 금방 이해 할 수 있겠지요.

```
for (i = 1; i < 10; i++) {
  for (j = 1; j < i; j++) {
    printf("%d ", j);
  }
}
```

이 부분에서 i 가 1 이면, j 가 출력되지 않고, i 가 2 가 되면 j 가 1 부터 1 까지, i 가 3 이 되면 j 는 1 부터 2 까지 순차적으로 출력되어 i 가 9 일 때, j 는 1 부터 8 까지 출력되어 위와 같은 모습을 보이게 됩니다. 어때요? 간단하지요?

while 문

아마 이 쯤 하셨다면 for 문에 대해 질렸을 것 같으니 for 문과 비스므리하면서도 다른 반복문인 while 문에 대해 살펴 보도록 해봅시다.

```c
/* while 문 */
#include <stdio.h>
int main() {
  int i = 1, sum = 0;

  while (i <= 100) {
    sum += i;
    i++;
  }

  printf("1 부터 100 까지의 합 : %d \n", sum);

  return 0;
}
```

성공적으로 컴파일 하였다면

실행 결과

```
1 부터 100 까지의 합 : 5050
```

와 같이 1 부터 100 까지 숫자들의 합이 출력됩니다.

while 문은 위의 예제에서도 알 수 있듯이 for 문과는 달리 구조가 사뭇 단순합니다. while 문의 기본 구조는 아래와 같습니다.

```c
while (/* 조건식 */) {
  // 명령1;
  // 명령2;
  // ...
}
```

for 문 처럼 '조건식' 에는 이 while 문을 계속 돌게 할 조건이 들어갑니다. 예를 들어서 조건식에 i <= 100 이 들어간다면 i 가 100 이하 일 때 만 조건이 성립하므로 i 가 100 이하일 때 까지 while 문이 계속 돌아가게 됩니다.

```c
while (i <= 100) {
  sum += i;
```

```
  i++;
}
```

위 경우, `i` 의 값이 100 이하 인 지 검사한 다음에 (`i <= 100`), `sum` 에 `i` 를 더하고 (`sum += i`), `i` 의 값을 증가한 뒤 (`i++`), 다시 처음으로 돌아가게 됩니다. 이 때, `while` 문의 특징이 바로 시작부터 조건식을 검사한다는 것입니다. (이는 `for` 문과 동일합니다.)

따라서, 만약 `i < 1` 이 조건식이라면 `while` 문 내부의 내용은 하나도 실행되지 않고 종료되게 됩니다.

do-while 문

```c
#include <stdio.h>
int main() {
  int i = 1, sum = 0;

  do {
    sum += i;
    i++;
  } while (i < 1);

  printf(" sum : %d \n", sum);
  return 0;
}
```

성공적으로 컴파일 하였다면

실행 결과

```
 sum : 1
```

와 같이 나오게 됩니다.

`do-while` 문의 구조는 아래와 같습니다.

```c
do {
  // 명령1;
  // 명령2;
  // ...
} while (/* 조건식 */);
```

`do - while` 문은 사실 `while` 문과 거의 비슷합니다. 한 가지 차이점은 앞서 말했듯이 `while` 문은 명령을 실행하기 전에 조건식이 참 인지 먼저 검사 합니다. 따라서, 조건식이 처음부터 참이 아니라면 `while` 문 안의 내용은 결코 실행 될 수 없겠지요.

그런데, `do - while` 은 먼저 명령을 실행 한 뒤에 조건식을 검사합니다. 따라서, 처음부터 조건식이 참이 아니라도 명령을 먼저 실행한 다음 조건식을 검사하기 때문에 최소한 한 번은 실행되게 됩니다.

```
do {
  sum += i;
  i++;
} while (i < 1);
```

따라서, 위 경우 `i` 가 1 로 `i < 1` 이 였지만 조건식을 나중에 검사하기 때문에 일단 `sum + = i;` 와 `i ++` 을 실행 한 다음에 `i < 1` 이 검사되어 `sum` 의 값이 1 이 출력될 수 있었던 것이지요. 어때요, 간단하죠?

그렇다면 이제 반복문에 대해 대충 감을 잡았을 것으로 기대합니다. 하지만 사실 반복문을 익숙하게 사용할 때 까지 많은 연습이 필요하기 때문에 제가 아래 반복문 사용법을 연습할 수 있는 몇 개 문제들을 준비하였습니다. 처음에는 문법 자체가 어색하므로 시간이 좀 걸리겠지만, 스스로 해보시길 바랍니다!

생각 해보기

문제 1 (난이도 : 中)

`N` 줄인 삼각형을 출력한다. 단, 사용자로 부터 임의의 `N` 을 입력 받는다. 아래는 `N = 3` 일 때의 출력 예시 이다.

```
* *** *****
```

문제 2 (난이도 : 中上)

위와 동일한 형태를 취하되, 역 삼각형을 출력한다. 아래는 `N = 3` 일 때의 출력 예시 이다.

```
***** *** *
```

문제 3 (난이도 : 下)

1000 이하의 3 또는 5 의 배수인 자연수들의 합을 구한다.

문제 4 (난이도 : 中)

1000000 이하의 피보나치 수열 (`N` 번째 항이 `N - 1` 번째 항과 `N - 2` 번째 항으로 표현되는 수열, 시작은 1,1,2,3,5,8,...) 의 짝수 항들의 합을 구한다

문제 5 (난이도 : 下)

사용자로 부터 `N` 값을 입력 받고 1 부터 `N` 까지의 곱을 출력한다.

문제 6 (난이도 : 中)

다음 식을 만족하는 자연수 `a,b,c` 의 개수를 구하여라

i) $a + b + c = 2000$ ii) $a > b > c$, a,b,c 는 모두 자연수

문제 7 (난이도 : 中上)

임의의 자연수 `N` 을 입력 받아 `N` 을 소인수 분해 한 결과를 출력하여라. 예를 들어서 `N = 18` 일 경우

```
N = 1818 = 2 * 3 * 3
```

문제 8 (난이도 : 上)

문제 7 에서 만든 프로그램의 속도를 향상 시킬 수 있는 방법은 없을까? 큰 수를 빠르게 소인수분해 할 수 있는 방법들을 찾아 프로그램에 적용시켜 보아라. 예를 들어서 `N` 의 제곱근 이하의 정수들만 처리한다던지, `Lucas- Lehmer` 판정법을 이용해 소수인지 아닌지 판정한다던지 등등..

(참조 : * 표가 붙은 문제들은 http://projecteuler.net/index.php?section=problems 에서 가져온 것들)

뭘 배웠지?

- `for` 문과 `while` 문의 사용법을 이해하고 사용할 수 있습니다.
- `break` 문을 사용하면 가장 가까운 루프에서 빠져나갈 수 있습니다.
- `continue` 를 사용하면, 아래 내용을 실행하지 않고 다음 루프를 실행합니다.

리눅스에서 C 프로그래밍 하기