20203324 게임공학과 최종현

```
5
함수 호출
10

함수 호출
-5

함수 호출
15

15
함수 호출
5

함수 호출
-10

함수 호출
10
```

```cpp
#include <string>
using namespace std;

void print(int i) // 함수 정의, 선언
{
    cout << "함수 호출" << endl;
    cout << i << endl;
}

int main(int argc, char const* argv[])
{

    int a = 10;
    int b;
    cin >> b; // b 값 입력받음

    print(a); // a에 저장된 10의 값이 print함수로 감
    cout << endl;

    print(b=b-a); // b에 5를 입력하면 -5가 b에 저장된다.
    // 출력값은 -5이다.
    cout << endl;

    print(a = a - b);
    // 출력값은 15
    cout << endl;

    cout << a << endl;
    print(a - 10);
    // 출력값은 5
    cout << endl;

    print(b - 5);
    // 출력값 -10
    cout << endl;
    print(10);
    // 함수에 상수를 전달했기 때문에 출력값은 10이다.
```

```
cout << (a > 6) << endl; // 참과 거짓을 판별해서 0(false) 또는 1(true)을 반환함.
// true는 컴퓨터에서 어떤 연산의 결과가 참이면 1이라고 출력한다.1말고 다른 값도 다 참임
cout << (a - 11 > 6) << endl;
cout << (a - 11 < 6) << endl;
cout << ((a - b) >= 10) << endl; // ()는 연산자중에서 제일 먼저 연산을 해야한다.

int c, d;
int* ptr1 = &c;
int* ptr2 = &d;
cout << (ptr1 - ptr2) << endl;
// 포인터 연산도 가능하다.
// 결과값은 -4가 나온다.
// 왜 -4가 나오냐면 ptr2의 주소번지가 ptr1의 주소번지보다 더 높기 때문이다.
// 메모리 상에서 d가 c보다 높은 주소에 위치
// ptr1을 먼저 선언을 했는데 왜 4가 아니고 -4가 나오는지는 컴퓨터마다 다르다.
// 컴파일러가 변수를 메모리에 배치하는 방식, 순서, 정렬을 위한 간격 등이 다를 수 있음
//결과가 - 4라는 것은 d의 주소와 c의 주소 사이에 int 타입 요소 4개만큼의 공간 차이가 있고,
// c가 d보다 메모리 주소가 낮다는 것을 의미한다.
// 이 '4'라는 값은 컴파일러가 변수 c와 d를 메모리에 어떻게 배치했는지에
// 따라 결정된 상대적인 거리일 뿐, 절대적인 규칙은 아니다.
```

```
for (int i = 0; i < b; i++)
{
    print(a);
}

// 입력되는 숫자만큼 출력되는것은 동적 or 정적일까?
// 입력되는 숫자만큼 함수가 호출되는것은 동적으로 처리가 된다.
// 프로그램이 실행되는 도중에 값을 받아서 호출하기 때문에 동적으로 처리가 된다.
```

```
1
0
1
1
-8
5
함수 호출
5
함수 호출
5
함수 호출
5
함수 호출
5
함수 호출
5
```

```cpp
int i = 0; // 반복횟수를 정하기 위한 변수 선언
cin >> i;

do
{
    cin >> i;
    cout << "Hello World" << endl;
    // i++;
} while (i!=0); // 컴퓨터는 0을 false로 판단하기 때문에 i=0이면 i는 false이기때문에 한 번만 출력된다.
// 계속조건


while (i!=0)
{
    cin >> i;
    cout << i << endl;
    cout << "Hello Wrold!" << endl;
}

int i1 = 0;
for (; i1 < 5; i1++)
{
    cout << "Hello World!" << endl;
}
cout << i1 << endl;
// 초기화칸은 비워둘 수 있다. 왜냐하면 루프 시작에 필요한 변수 i1의 초기화(int i1 = 0;)가
// 이미 for문 바깥에서 이루어졌기 때문이다.
// 그래서 for문 자체에서는 별도의 초기화를 할 필요가 없는 거다.
// 세미콜론은 반드시 있어야 한다.
```

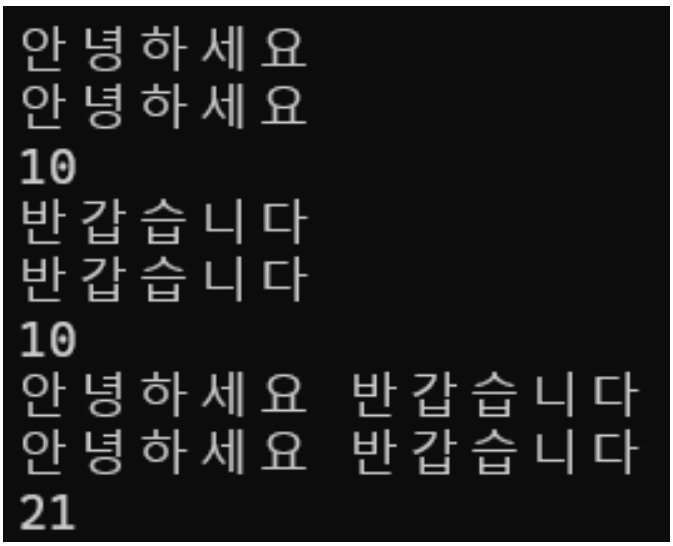
```
1
2
Hello World
3
Hello World
4
Hello World
0
Hello World
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
5
```

```cpp
// 첫 번째 do while , while 비교
// string을 통해 입력을 받음, 입력된 문자열의 길이가 15이상이면 프로그램이 종료가 되게끔 작성

string str;

do
{
    getline(cin, str);
    cout << str << endl;
    cout << str.length() << endl;
} while (str.length()<=15);


while (str.length() <= 15)
{
    getline(cin,str);
    cout << str << endl;
    cout << str.length() << endl;
}
```



```cpp
// 두 번째 do while , while 비교

// do while은 한 번은 무조건 실행해야 한다.
// 처음부터 조건을 거짓으로 설정을 하고 결과를 확인하면
int i2 = 100;
do
{
    i2++;
} while (i2<100);
cout <<"i2 : "<< i2 << endl;
// 결과값은 i는 101이 나온다.
int ii = 100;
while (ii < 100)
{
    ii++;
}
cout <<"ii :"<< ii << endl;
// ii는 100이 출력되게 된다.
```

```
i2 : 101
ii :100
```

```cpp
// 세 번째 do while , while 비교
// 입력을 받을 때는 do while은 별도로 초기화를 하지 않아도 된다.
// 하지만, while은 먼저 조건을 검사하기 때문에 0으로 초기화 하지 않아야하는
// 번거로움이 존재한다.
// 사용자 입력 값에 따라 반복 여부 결정 (0 입력 시 종료)
int input;
do {
    cout << "숫자 입력: ";
    cin >> input; // 먼저 입력을 받음
    cout << "입력값: " << input << endl;
} while (input!= 0); // 입력값이 0인지 검사
cout << input<< endl;

cout << endl;

int input1 = -1; // while 루프에 진입하기 위해 0이 아닌 값으로 초기화
// 또는 입력을 루프 전에 한 번 받아야 함
// 만약 input_while을 0으로 초기화했다면 아래 루프는 실행되지 않음.
// 만약 while을 사용하려면 보통 이렇게 먼저 입력받는 경우가 많음
// cin >> input_while;

while (input1!= 0) { // 먼저 조건을 검사
    cout << "숫자 입력: ";
    cin >> input1;
    cout << "입력값: " << input1 << endl;
}
cout <<input1 << endl;
```

```
숫자 입력: 123
입력값: 123
숫자 입력: 123
입력값: 123
숫자 입력: 0
입력값: 0
0

숫자 입력: 0
입력값: 0
0
```

```cpp
// 네 번째 do while , while 비교


int* ptr = nullptr; // 포인터가 아무것도 가리키지 않음
int counter = 0;
do {
    // 일단 루프 안으로 들어와서 실행!
    cout << "do-while 루프 안 실행 횟수: " << ++counter<< endl;
    // 여기서 ptr이 가리키는 값을 사용하려고 하면 위험
    // 이 예제에서는 그냥 루프가 도는지만 확인.

    // 루프 종료를 위해 조건을 바꾸지 않음 (어차피 ptr!= nullptr 조건은 거짓)
} while (ptr!= nullptr); // 조건을 나중에 검사 (nullptr != nullptr -> 거짓)

cout << counter << "번 실행됨." << endl;

cout << endl;


int* ptr5 = nullptr; // 포인터가 아무것도 가리키지 않음
int counter2 = 0;
// 조건을 먼저 검사(nullptr != nullptr -> 거짓)
while (ptr5!= nullptr) {
    // 조건이 처음부터 거짓이므로 이 안은 절대 실행되지 않음.
    cout << "while 루프 안 실행 횟수: " << ++counter2<< endl;
}
cout << counter2 << "번 실행됨." << endl;
```

```
do-while 루프 안 실행 횟수: 1
1번 실행됨.

0번 실행됨.
```

```cpp
// 다섯 번째 do while, while 비교

int arr[] = { 1,2,3,4,5 };
int *num = arr;
int count = 0;
do
{
    ++count;
    cout << "*num이 가리키는 값 : " << *num << endl;
    // num이 가리키는 값(주소값 X)
    num++;
    // 주소 1 증가
    // 실행횟수 체크
} while (*num == 0);
cout << count <<"번 실행"<< endl;
// 무조건 한 번 실행을 했기 때문에 실행 횟수가 1번

cout << endl;

int* num2 = arr;
int count2 = 0;
while (*num2 == 0)
{
    ++count2;
    cout << "*num이 가리키는 값 :" << *num2 << endl;
    num2++;
}
cout << count2 << "번 실행" << endl;
// 조건이 애초에 거짓이기 때문에 실행 횟수가 0번
```

```
*num이 가리키는 값 : 1
1번 실행

0번 실행
```

```cpp
// 9번
int input1;
int count = 0;
do {
    ++count;
    cout << "정수를 입력하시오: ";
    cin >> input1;
    cout << "입력값: " << input1 << endl;
} while (input1 != 0);
cout << count << endl;
// 반복되는 문장은 정수를 입력하시오 와 입력값 두가지 이다.
// 여기서 반복되는 횟수를 추론하는 코드를 작성해보겠다.
// 선 증가를 한 이유는 do while은 무조건 한 번 실행이 되기 때문에
// 선 증가를 사용했다.

// 10번
int count1 =0, num;
cout << "반복 횟수를 입력하시오: ";
cin >> num;

while (count1 < num-2) {
    cout << "repeat (" << count1 << ")" << endl;
    count1++;
}
// 입력받은 값에서 2를 빼면 된다.
```

```
정수를 입력하시오: 10
입력값: 10
정수를 입력하시오: 0
입력값: 0
2
반복 횟수를 입력하시오: 5
repeat (0)
repeat (1)
repeat (2)
```

```cpp
// 13번
for (int i = 0; i < 10; i++) {

    cout << i%4 << ", ";
}
cout << endl;

// 예상 결과가 0,1,2,3,0,1,2,3,0,1 이 나오게 하려면
// i를 4로 나눈 나머지로 계산하면 된다.

// 14번
int sum;
sum = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10;  // 1 ~ 10 까지의 정수 더하기

cout << "(1): 1+2+3+4+5+6+7+8+9+10 = " << sum << endl;

sum = 0;
for (int i = 1; i <= 10; i++) {

    sum += i;
}
cout << "(2): 1+2+3+4+5+6+7+8+9+10 = " << sum << endl;

// sum변수에 i가 증가하는 값을 계속해서 더해주면 된다.
```

```
0, 1, 2, 4, 5, 7, 8, 9,
0, 1, 2,
0, 1, 2, 3, 0, 1, 2, 3, 0, 1,
(1): 1+2+3+4+5+6+7+8+9+10 = 55
(2): 1+2+3+4+5+6+7+8+9+10 = 55
```

```cpp
int sum1 = 1;
for (int i = 1; i <= 16; i++) {

    sum1 *= i;
}
cout << sum1 << endl;

double sum2 = 1;
for (int i = 1; i <= 17; i++) {

    sum2 *= i;
}
cout << sum2 << endl;

// sum2에서는 int자료형을 사용할시 저장가능한값을 넘어버려 -가 나온다.
```

```
2004189184
3.55687e+14
```

do while, while , for을 어느 상황에서 써야할까?

for 루프는 반복 횟수가 비교적 명확하거나, 일정한 범위 내에서 값을 증가/감소시키며 반복할 때 가장 많이 사용된다. 초기화, 조건 검사, 값의 증감(업데이트) 부분이 명확하게 구분되어 있어 코드를 이해하기 쉽다. 특히 배열의 모든 요소를 순회하거나, 0부터 특정 숫자까지 반복하는 등의 작업에 매우 유용함.

while 루프는 특정 조건이 참(true)인 동안 계속 반복하고 싶을 때 사용한다. 반복 횟수를 미리 알기 어렵고, 오직 어떤 상황(조건)이 만족되는 동안만 작업을 계속해야 할 때 유용하다. 초기화는 루프 시작 전에, 값의 증감(업데이트)은 루프 본문 안에서 직접 처리해야 한다.

do-while 루프는 while 루프와 비슷하지만, 조건을 검사하기 전에 루프 본문을 최소 한 번은 반드시 실행한다는 점이 가장 큰 특징이다. 따라서 "일단 한번 실행하고 나서, 조건을 보고 계속할지 결정"해야 하는 상황에 적합하다.