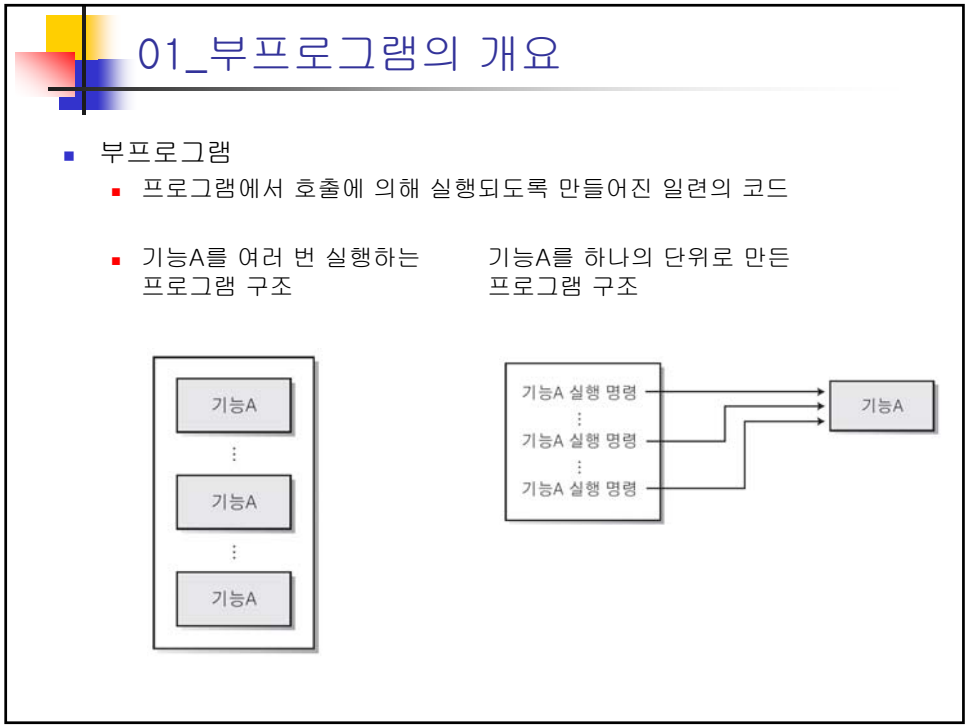
1. 부프로그램의 개요
2. 매개변수 전달 방식
3. 중복 부프로그램
4. 포괄 부프로그램
5. 매크로 함수와 인라인 함수
6. 부프로그램의 구현



## 01_부프로그램의 개요

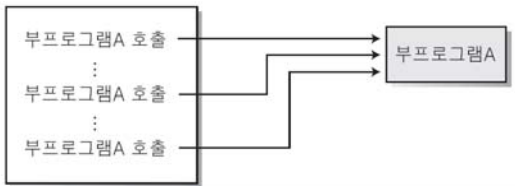
- 부프로그램
  - 프로그램에서 호출에 의해 실행되도록 만들어진 일련의 코드
  - 기능A를 여러 번 실행하는 프로그램 구조



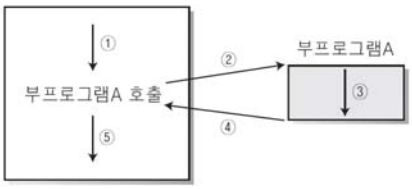

기능A를 하나의 단위로 만든 프로그램 구조

## 01_부프로그램의 개요

- 부프로그램이 있는 프로그램 구조




- 부프로그램이 있는 프로그램의 실행 순서




## 01_부프로그램의 개요

- 부프로그램 정의와 호출
  - 부프로그램 정의
    - 부프로그램이 실행할 내용을 기술한 일련의 코드로 머리부(header)와 본체(body)로 구성
  - 부프로그램 호출
    - 호출되는 부프로그램을 실행하라는 명령
  - 부프로그램 정의의 머리부
    - 부프로그램임을 나타내는 예약어, 부프로그램의 이름, 매개변수들의 이름과 타입, 반환 값의 타입 등을 기술
    - FORTRAN 부프로그램의 머리부 `SUBROUTINE SUB(K, L)`
    - Pascal 부프로그램의 머리부 `procedure sub(var k,l: integer);`
    - C 언어의 부프로그램 머리부 `int sub(int k, int l)`

## 01_부프로그램의 개요

- 부프로그램 정의
  - FORTRAN 부프로그램 정의 예

```
SUBROUTINE SUB(K, L)
IF(K .GT. L) THEN
  PRINT *, K
ELSE
  PRINT *, L
ENDIF
RETURN
END
```

C 부프로그램 정의 예

```
void sub(int k, int l)
{
   if(k > l)
      printf("%d\n", k);
   else
      printf("%d\n", l);
}
```

- 부프로그램 선언
  - 부프로그램이 정의되어 있다는 것을 컴파일러에게 알리는 역할
  - 부프로그램의 머리부는 제공하지만 부프로그램 몸체를 포함하진 않음

## 01_부프로그램의 개요

- C 언어 부프로그램 선언의 예

```
void sub(int, int);
```

- 부프로그램 호출
  - FORTRAN에서 SUB 부프로그램을 호출하는 예

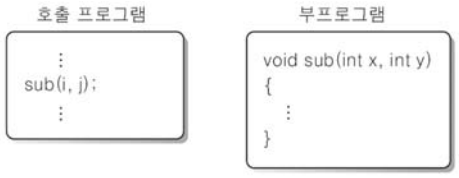
```
CALL SUB(I, J)
```

  - Pascal과 C에서 sub1 부프로그램을 호출하는 예

```
sub(i, j)
```

## 01_부프로그램의 개요

- 호출 프로그램에서 부프로그램에게 어떤 정보를 전달
  - C 예




    - 실 매개변수 : 호출 문장의 매개변수 i와 j
    - 형식 매개변수 : 부프로그램 정의 머리부의 매개변수 x와 y

- 위치 매개변수 방법
  - 위치에 따라 형식 매개변수와 실 매개변수를 대응하는 방법

## 01_부프로그램의 개요

- 키워드 매개변수 방법
  - 호출 문장에서 실 매개변수와 대응되는 형식 매개변수의 이름을 지정할 수 있는 방법
  - Ada

```
procedure sub(koreanScore, mathScore, engScore: integer) is
begin
  ⋮
end sub;
```

  - 호출 문장

```
sub(i=>mathScore, j=>engScore, k=>koreanScore);
```

- Ada는 위치 매개변수 방법과 키워드 매개변수 방법을 혼합
  - 위치 매개변수 방법을 적용한 i는 koreanScore와 대응되고, 키워드 매개변수 방법을 적용한 j는 engScore와, 그리고 k는 mathScore와 대응

```
sub(i, j=>engScore, k=>mathScore);
```

## 01_부프로그램의 개요

- C++, Ada 등의 디폴트 매개변수
  - Ada의 예

```
with TEXT_IO;
 use TEXT_IO;
 procedure default is
   package INT_IO is new TEXT_IO.INTEGER_IO (integer);
   use INT_IO;
   i, k: integer;
   procedure sub(koreanScore: integer; mathScore: integer := 60;
engScore: integer) is
   begin
     put("korean: "); put(koreanScore); NEW_LINE;
     put("mathematics: "); put(mathScore); NEW_LINE;
     put("english: "); put(engScore);
   end sub;
 begin
   i := 80; k := 70;
   sub(i, engScore=>k);
 end default;
```

## 01_부프로그램의 개요

- C++의 예

```
01 #include <iostream>
02 using std::cout;
03 using std::endl;
04
05 void sub(int koreanScore, int mathScore, int engScore=60)
06 {
07     cout << "korean: " << koreanScore << endl;
08     cout << "mathematics: " << mathScore << endl;
09     cout << "english: " << engScore << endl;
10 }
11
12 int main(void)
13 {
14     sub(80, 70, 90);
15     sub(70, 50);
16
17     return 0;
18 }
```

## 01_부프로그램의 개요

- 부프로그램의 종류
  - 프로시저 : 값을 반환하지는 않고 부작용(외부에서 정의된 값을 수정) 만듦
  - 함수 : 값을 반환하고 부작용은 없음
    - 프로시저와 함수를 별도로 구분하지 않기도 함
  - FORTRAN의 프로시저

```
    SUBROUTINE COPYSUB(A, B, N)
    INTEGER A(N), B(N)
    DO 10 I = 1, N
      A(I) = B(I)
10  CONTINUE
    RETURN
    END
```

    - 호출 : CALL COPYSUB(K, L, I)

## 01_부프로그램의 개요

  - FORTRAN의 함수

```
INTEGER FUNCTION ADD(A, B)
ADD = A + B
RETURN
END
```

    - 호출 : I = ADD(K, L)
  - Ada의 프로시저

```
procedure incre(x, y: in out integer) is
begin
  x := x + 1;
  y := y + 1;
end incre;
```

## 01_부프로그램의 개요

- Ada의 함수

```
function 함수이름(매개변수) return 반환값 타입 is
begin
   return 반환값;
end 함수이름;
```

  - 함수를 사용하는 Ada 예제

```
with TEXT_IO;
use TEXT_IO;
procedure func is
  package INT_IO is new TEXT_IO.INTEGER_IO (integer);
  use INT_IO;
  function add(num1, num2: integer) return integer is
  begin
    return num1 + num2;
  end add;
begin
  put(add(5, 7));
end func;
```

## 01_부프로그램의 개요

- C
  - 값을 반환할 때 함수 이름 왼쪽에 반환하는 값의 타입을 쓴다

```
int add(int x, int y)
{
  return x+y;
}
```

  - 값을 반환하지 않으려면 void 사용

## 02_매개변수 전달 방식

- 값 전달
  - 실 매개변수의 값을 형식 매개변수에 저장하고 형식 매개변수를 부프로그램의 지역 변수로 사용
  - 실 매개변수와 형식 매개변수는 별개의 변수이므로 형식 매개변수의 어떠한 변화도 실 매개변수에는 아무런 영향을 미치지 않음
  - C와 Java, C++는 값 전달이 기본 방식, Ada의 in 매개변수가 값 전달 방식
  - 값 전달 방식을 하는 C 예제

[호출프로그램]

```
        ...
01 a=10; b=20;
02 sub(a, b);
03 printf("%d %d", a, b);
        ...
```

[부프로그램]

```
01 void sub(int x, int y)
02 {
03    x=x+1; y=y+1;
04    printf("%d %d", x, y);
05 }
```

## 02_매개변수 전달 방식

- in 매개변수를 이용해서 값 전달을 보여주는 Ada 예
  - Ada in 매개변수의 특이점은 값을 배정할 수 없다는 점

```
with TEXT_IO;
use TEXT_IO;
procedure inparameter is
package INT_IO is new TEXT_IO.INTEGER_IO (integer);
use INT_IO;
a, b: integer;
procedure sub(x, y: in integer) is
begin
put(x); put(y);
end sub;
begin
a := 10; b := 20;
sub(a, b);
end inparameter;
```

## 02_매개변수 전달 방식

- 참조 전달
  - 실 매개변수의 주소를 형식 매개변수로 보내는 방식
  - 형식 매개변수는 실 매개변수의 별명(alias)이 되어 형식 매개변수의 어떠한 변화도 실 매개변수에 그대로 반영
  - FORTRAN, C++의 참조자 형식 매개변수를 사용하는 경우
  - 참조자를 이용해서 참조 전달을 보여주는 C++ 예

```
[호출프로그램]

        ...
01 a=10; b=20;
02 sub(a, b);
03 printf("%d %d", a, b);
        ...
```

```
[부프로그램]

01 void sub(int &x, int &y)
02 {
03    x=x+1; y=y+1;
04    printf("%d %d", x, y);
05 }
```

## 02_매개변수 전달 방식

- 참조 전달 효과를 내는 C 예제

```
[호출프로그램]

        ...
01 a=10; b=20;
02 sub(&a, &b);
03 printf("%d %d", a, b);
        ...
```

```
[부프로그램]

01 void sub(int *x, int *y)
02 {
03    *x=*x+1; *y=*y+1;
4    printf("%d %d", *x, *y);
05 }
```

## 02_매개변수 전달 방식

- 값-결과 전달
  - 부프로그램을 호출할 때 실 매개변수의 값을 형식 매개변수에 저장하고 동작하다가 부프로그램을 종료할 때 형식 매개변수의 값을 실 매개변수로 반환
  - ALGOL 68, Ada의 in out 매개변수
  - 값-결과 전달 방식을 하는 Ada 예제

```
[호출프로그램]

        ...
01 a=10; b=20;
02 sub(&a, &b);
03 put(a); put(b);
        ...
```

```
[부프로그램]

01 procedure sub(x, y:in out integer) is
02 begin
03    x:=x+1; y:=y+1;
4     put(x); put(y);
05 end sub;
```

## 02_매개변수 전달 방식

- **이름 전달**
  - 형식 매개변수의 이름이 사용될 때마다 대응되는 실 매개변수의 이름으로 대치하는 방식
  - ALGOL 60
  - 이름 전달 방식을 하는 ALGOL 60 예제

```
[호출프로그램]

        ...
01 a := 1;
02 n(1) := 1;
03 (n2) : = 2;
04 call sub(a, n(a));
        ...
```

```
[부프로그램]

01 procedure sub(x, y)
02    integer x, y;
03 begin
04    x := x +1;
05    y := y +1;
06 end
```

```
04 a := a+1;
05 n(a) := n(a)+1;
```

  - 이름 전달 방식을 사용하는 부프로그램 동작은 형식 매개변수의 이름을 실매개변수의 이름으로 대치하여 실행한 결과와 같음
  - 프로그램을 작성하기가 어렵고 구현이 어려워 최근 대부분의 언어에서는 사용되지 않음

## 03_중복 부프로그램

- 중복 부프로그램
  - 같은 이름을 갖는 두 개 이상의 부프로그램들을 의미
  - 단 모든 중복 부프로그램은 매개변수 내용이 달라야 함
  - C++, Java, Ada 등에서 중복 부프로그램 기능을 지원

  - 오류가 있는 C 예제

```
01  int add(int x, int y)
02  {
03  return x+y;
04  }
05  float add(float x, float y)
06  {
07      return x+y;
08  }
09  int main(void)
10  {
11      add(10, 20);
12      return 0;
13  }
```

함수의 이름이 같으므로 번역 시 오류 발생

## 03_중복 부프로그램

- 중복 부프로그램을 사용하는 C++ 예제

```
01  int add(int x, int y)
02  {
03      return x+y;
04  }
05  float add(float x, float y)
06  {
07      return x+y;
08  }
09  int main(void)
10  {
11      add(10, 20);
12      add(1.2, 3.4);
13      return 0;
14  }
```

함수의 이름이 같아도 매개변수의 내용이 다르면 다른 함수로 인식

## 03_중복 부프로그램

- 중복 부프로그램과 디폴트 매개변수를 함께 사용하여 오류가 있는 C++ 예제

```
01 int func(int x=0)  ➔ 디폴트 매개변수 사용
02 {
03    return x+10;
04 }
05 int func()          ➔ 매개변수 없음
06 {
07    return 0;
08 }
09 int main(void)
10 {
11    func();          1행과 5행의 함수 모두에게 해당되므로 오류 발생
12    return 0;
13 }
```

## 04_포괄 부프로그램

- 포괄 부프로그램
  - 다양한 타입의 매개변수를 허용하는 부프로그램
  - Ada 예를 통한 이해 : 두 예가 거의 유사 → 이때 포괄 부프로그램을 사용
    - 정수 타입의 두 데이터를 교환하는 Ada 부프로그램

```
procedure swapInt (x, y: in out integer) is
  temp: integer;
begin
  temp := x;
  x := y;
  y := temp;
end swap;
```

    - 부동 소수점 타입의 두 데이터를 교환하는 Ada 부프로그램

```
procedure swapFloat (x, y: in out float) is
  temp: float;
begin
  temp := x;
  x := y;
  y := temp;
end swap;
```

## 04_포괄 부프로그램

- Ada
  - generic이라는 예약어를 이용해서 포괄 부프로그램을 생성

```
generic
   type Element is private;
procedure exchange(x, y: in out Element);
procedure exchange (x, y: in out Element) is
   temp: Element;
begin
   temp := x;
   x := y;
   y := temp;
end swap;
```

  - integer, float, character 타입의 데이터를 교환하는 swap이라 불리는 3개의 부프로그램이 정의

```
procedure swap is new exchange (integer);
procedure swap is new exchange (float);
procedure swap is new exchange (character);
```

  - 호출

```
swap(a, b);
```

## 04_포괄 부프로그램

- 포괄 부프로그램을 사용하는 Ada 예제

```
01 with TEXT_IO;
02 use TEXT_IO;
03 procedure gene is
04     package INT_IO is new TEXT_IO.INTEGER_IO (integer);
05     use INT_IO;
06     a, b: integer;
07     c, d: character;
08     generic
09         type Element is private;
10     procedure exchange(x, y: in out Element);
11     procedure exchange(x, y: in out Element) is
12         temp: Element;
13     begin
14         temp := x;
15         x := y;
16         y := temp;
17     end exchange;
18     procedure swap is new exchange (integer);
19     procedure swap is new exchange (float);
20     procedure swap is new exchange (character);
```

## 04_포괄 부프로그램

```
21 begin
22      a := 10; b := 20;
23      swap(a, b);   ➔ integer 타입 데이터에 대한 교환 수행
24      put(a); put(" "); put(b); NEW_LINE;
25      c := 'i'; d := 'j';
26      swap(c, d);  ➔ character 타입 데이터에 대한 교환 수행
27      put(c); put(" "); put(d);
28 end gene;
```

## 05_매크로 함수와 인라인 함수

- 일반 함수 사용 시 단점
  - 호출과 반환으로 인한 메모리 할당 등 오버헤드가 발생
  - 짧은 길이의 C 함수를 이용하는 것은 장점보다는 비효율적인 면이 많음

```
int substraction(int x, int y)
{
   return x-y;
}
```

➔ 이때, 매크로 함수를 사용하는 편이 바람직

- 매크로 함수
  - 함수 이용으로 인한 오버헤드가 발생하지 않음
  - 매크로 함수의 정의 부분이 길면 프로그램의 크기가 커지게 되는 문제점이 있음
  - 짧은 길이인 경우에 매크로 함수를 이용하고 긴 경우에는 일반적인 함수를 이용하는 것이 바람직

## 05_매크로 함수와 인라인 함수

- 문제 발생 예로 살펴본 매크로 함수 정의

```
#define substraction (x, y)  x-y
```

  - 문제 발생!

예상하던 (a+ 1)-(b+ 10)과는 다른 결과

```
result = substraction(a+1, b+10);
```
⇒
```
result = a+1 - b+10;
```

    - 1. 해결

```
#define substraction (x, y)  (x)-(y)
```

함수를 정의할 때 괄호로 묶어야 함

    - 2. 여전히 문제점 남음

연산자 우선순위에 의해 (b)*2가 먼저 수행

```
result = substraction(a, b)*2;
```
⇒
```
result = (a)-(b)*2;
```

    - **3. 해결: 완성된 매크로 함수**

```
#define substraction (x, y) ((x)-(y))
```

정의 부분 전체를 괄호로 묶어야 함

## 05_매크로 함수와 인라인 함수

- C++의 인라인 함수
  - 일반함수처럼 정의하기 쉬움
  - 매크로 함수처럼 동작

```
inline int substraction(int x, int y)
{
   return x-y;
}
```

||

```
#define substraction (x, y) ((x)-(y))
```

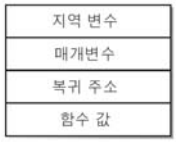
```
#define substraction (x, y) ((x)-(y))
```

## 06_부프로그램의 구현

- 정적 구조
  - FORTRAN 77
    - 메모리 할당이 적재 시간에 이루어지며 모든 변수들의 위치가 고정
    - 부프로그램은 중첩될 수 없고 재귀 호출도 허용되지 않음
    - COMMON 문을 이용해서 전역 변수를 선언할 수 있고, 그 외 변수들은 지역 변수
    - FORTRAN 77의 주프로그램과 2개의 부프로그램 A, B로 이루어진 프로그램의 메모리 구조를 나타낸 구조


## 06_부프로그램의 구현

- 활성 레코드
  - 지역 변수는 해당 주프로그램 또는 부프로그램에서 선언된 지역 변수를 의미
  - 매개변수는 해당 부프로그램의 매개변수를 의미
  - 복귀 주소는 부프로그램 실행이 종료되고 복귀해야 할 메모리 주소를 의미
  - 함수 값은 부프로그램이 함수인 경우 함수의 반환 값을 의미



## 06_부프로그램의 구현

- 스택 기반 구조
  - C, Pascal, Ada 등으로 재귀 호출을 허용
  - C는 부프로그램의 중첩을 허용하지 않는 반면 Pascal과 Ada는 중첩을 허용
  - 스택 기반 구조의 C 예제

```
01  void q(void)
02  {
03     ⋮
04  }
05  void p(void)
06  {
07     q();
08     ⋮
09  }
10  int main(void)
11  {
12     p();
13     ⋮
14  }
```

## 06_부프로그램의 구현

- 프로그램 실행 : 스택에 main 활성 레코드가 생성됨


- main에서 p를 호출하여 p가 활성화되면 스택에 p의 활성 레코드가 생성됨


- p에서 q를 호출하여 q가 활성화되면 스택에 q의 활성 레코드가 생성됨

## 06_부프로그램의 구현

- q의 실행 종료 : q의 활성 레코드는 스택에서 삭제됨

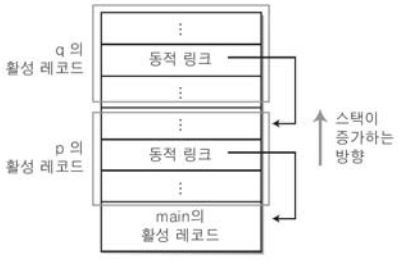

- p의 실행 종료 : p의 활성 레코드가 스택에서 삭제됨


- 스택 기반 구조의 활성 레코드
  - 임의의 활성 레코드를 스택에서 제거할 때, 제거 범위에 대한 정보 필요
  - 동적 링크가 이 역할을 담당
  - 호출자의 활성 레코드의 시작부를 가리킴


## 06_부프로그램의 구현

- 스택 기반의 C 예제에서 q가 활성화되었을 때의 스택 구조 살펴보기
  - q의 동적 링크: 자신의 호출자의 p의 활성 레코드의 시작부를 가리킴
  - p의 동적 링크: 자신의 호출자의 main의 활성 레코드의 시작부를 가리킴

```
01  void q(void)
02  {
03     ⋮
04  }
05  void p(void)
06  {
07     q();
08     ⋮
09  }
10  int main(void)
11  {
12     p();
13     ⋮
14  }
```




- 이 상태에서 q의 실행이 종료되면 활성 레코드의 동적 링크가 가리키는 부분까지 스택에서 삭제됨

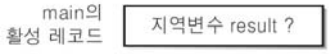
## 06_부프로그램의 구현

- 스택 기반 구조의 재귀 호출이 있는 C 예제

```
01  int factorial(int n)
02  {
03     if(n<=1)
04        return 1;
05     else
06        return n*factorial(n-1);
07  }
08  int main(void)
09  {
10     int result;
11     result = factorial(3);
12     return 0;
13  }
```

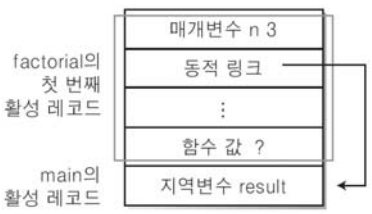
## 06_부프로그램의 구현

- 프로그램 실행 : 스택에 main 활성 레코드가 생성됨




- 11행에서 factorial(3)을 호출하면 스택에 factorial 의 활성 레코드가 생성됨
- 매개변수 n=3, 동적링크는 호출자의 main 의 활성 레코드를 가리킴
- 함수의 반환값은 정해지지 않음

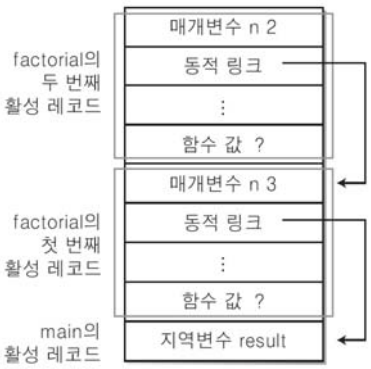
```
01  int factorial(int n)
02  {
03     if(n<=1)
04        return 1;
05     else
06        return n*factorial(n-1);
07  }
08  int main(void)
09  {
10     int result;
11     result = factorial(3);
12     return 0;
13  }
```



# 06_부프로그램의 구현

- 6행에서 factorial(2)를 재귀호출하면, 스택에 또 다른 factorial의 활성 레코드가 생성됨
- 매개변수 n=2, 동적 링크는 호출자인 factorial의 활성 레코드를 가리킴
- 함수의 반환값은 정해지지 않음

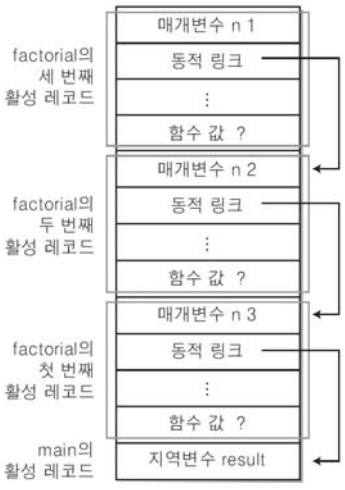
```
01 int factorial(int n)
02 {
03   if(n<=1)
04     return 1;
05   else
06     return n*factorial(n-1);
07 }
08 int main(void)
09 {
10   int result;
11   result = factorial(3);
12   return 0;
13 }
```




# 06_부프로그램의 구현

- 6행에서 factorial(1)을 재귀호출하면, 스택에 또 다른 factorial의 활성 레코드가 생성됨
- 매개변수 n=1, 동적 링크는 호출자인 factorial의 활성 레코드를 가리킴
- 함수의 반환값은 정해지지 않음

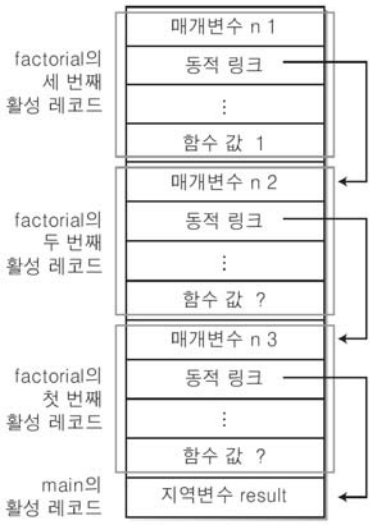
```
01 int factorial(int n)
02 {
03   if(n<=1)
04     return 1;
05   else
06     return n*factorial(n-1);
07 }
08 int main(void)
09 {
10   int result;
11   result = factorial(3);
12   return 0;
13 }
```



# 06_부프로그램의 구현

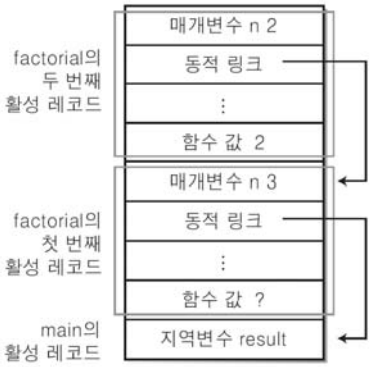
- 4행에서 함수의 반환값이 1로 정해지므로 활성 레코드의 함수 값이 1이 된다

```
01 int factorial(int n)
02 {
03    if(n<=1)
04       return 1;
05    else
06       return n*factorial(n-1);
07 }
08 int main(void)
09 {
10    int result;
11    result = factorial(3);
12    return 0;
13 }
```



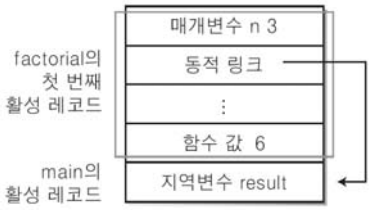

# 06_부프로그램의 구현

- 함수값 1을 factorial의 두번째 활성 레코드로 반환
- factorial(1)의 실행이 종료되면 동적 링크가 가리키는 부분까지 스택에서 삭제됨
- factorial 두번째 활성 레코드의 n인 2와 반환받은 1을 곱한 2가 함수값이 됨

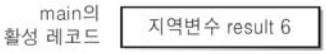


## 06_부프로그램의 구현

- 함수값 2를 factorial의 첫번째 활성 레코드로 반환
- factorial(2)의 실행이 종료되면 동적 링크가 가리키는 부분까지 스택에서 삭제
- factorial 첫번째 활성 레코드의 n인 3과 반환받은 2를 곱한 결과인 6이 함수값




- 함수값 6을 main의 활성 레코드로 반환
- factorial(3)의 실행이 종료되면 동적 링크가 가리키는 부분까지 스택에서 삭제됨
- main 활성 레코드의 지역변수 result가 6이 됨




## 06_부프로그램의 구현

- 동적 구조
  - APL과 같은 인터프리터 언어
  - 활성 레코드는 스택에 생성하여 동적 링크로 연결, 이때 활성 레코드에서 지역변수와 같은 각 항목은 이름과 포인터로 구성
  - 비지역 변수의 참조는 호출 순서에 기반하는 동적 영역 규칙을 따르므로 동적 링크를 따라가며 해당 이름에 대한 선언을 찾음
  - 정적 영역 규칙에 필요한 정적 링크는 불필요하므로 활성 레코드에 정적 링크 항목은 없음

## 요약

1. 부프로그램은 프로그램에서 호출에 의해 실행되도록 만들어진 일련의 코드를 의미한다.
2. 부프로그램 정의는 부프로그램이 실행할 내용을 기술한 일련의 코드로, 머리부와 본체로 구성된다.
3. 부프로그램 호출은 호출되는 부프로그램을 실행하라는 명령이다.
4. 매개변수란 호출 프로그램에서 부프로그램에게 어떤 정보를 전달하고자 할 때 사용하는 변수이다.
5. 부프로그램은 값을 반환하지는 않고 부작용을 만드는 프로시저와 값을 반환하고 부작용이 없는 함수로 구분한다.
6. 매개변수 전달 방식에는 값 전달, 참조 전달, 값-결과 전달, 이름 전달 등이 있다.
   - 값 전달 방식은 실 매개변수의 값을 형식 매개변수에 저장하고 형식 매개변수를 부프로그램의 지역 변수로 사용한다.
   - 참조 전달 방식은 실 매개변수의 주소를 형식 매개변수로 보내는 방식이다.
   - 값-결과 전달 방식은 부프로그램을 호출할 때 실 매개변수의 값을 형식 매개변수에 저장하고 동작하다가, 부프로그램을 종료할 때 형식 매개변수의 값을 실 매개변수로 반환한다.
   - 름 전달 방식은 형식 매개변수의 이름이 사용될 때마다 대응되는 실 매개변수의 이름으로 대치하는 방식이다.
7. 중복 부프로그램은 같은 이름을 갖는 두 개 이상의 부프로그램들을 의미하는데, 모든 중복 부프로그램은 매개변수 내용이 달라야 한다.
8. 포괄 부프로그램은 다양한 타입의 매개변수를 허용하는 부프로그램이다.

## 요약

9. 길이가 짧은 경우에는 일반적인 함수를 이용하는 것보다 매크로 함수와 인라인 함수를 이용하면 오버헤드가 줄게 된다.
10. 주프로그램과 각 부프로그램은 코드부와 활성 레코드로 구성되는데, 코드부는 프로그램 명령어들로 내용이 변하지 않고 활성 레코드는 지역 변수, 매개변수 등 단위 프로그램 실행에 필요한 정보이다.
11. 부프로그램이 메모리에 할당되는 구조는 크게 정적 구조, 스택 기반 구조, 동적 구조로 구분된다.