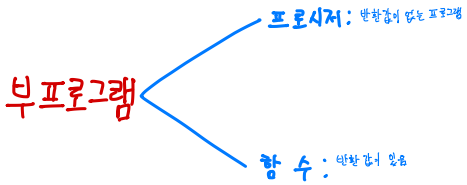
Subprogram



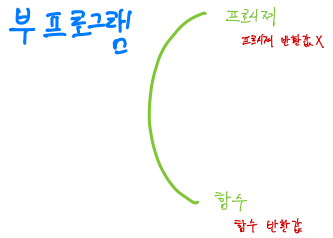
목차

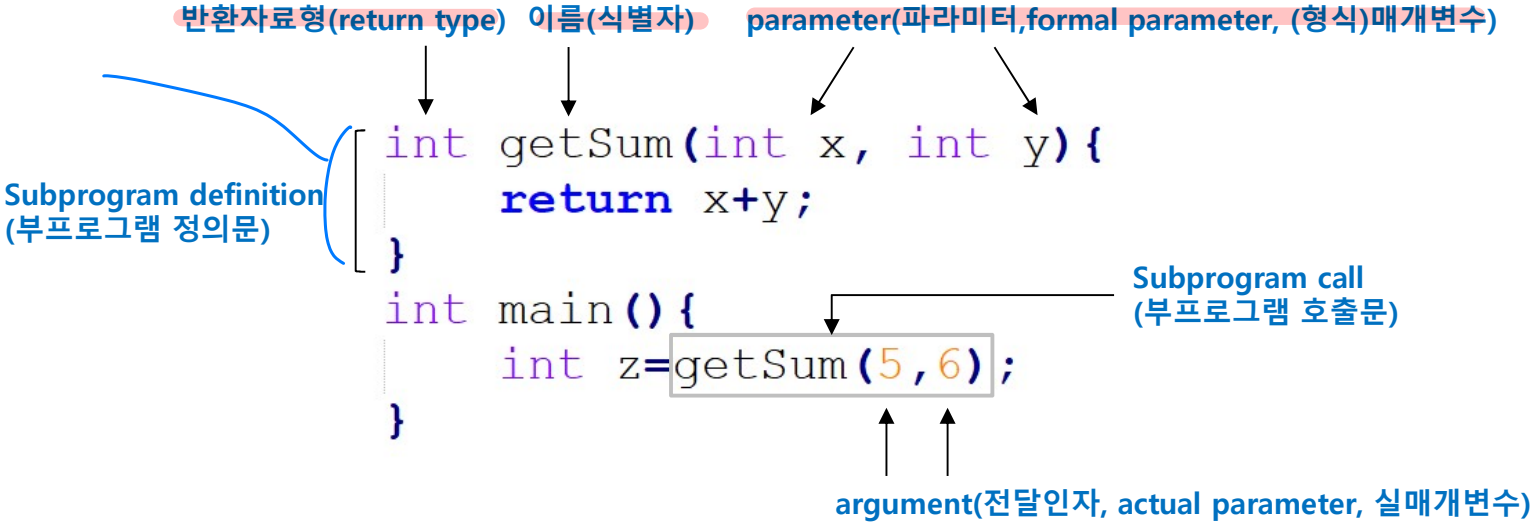
- 부프로그램
- 프로시저 vs. 함수
- 매개변수 전달
  - Call-by-value
  - Call-by-reference
  - Call-by-value-result
  - Call-by-name

부프로그램

개념

- 부프로그램(subprogram)은 프로그램 내 호출 가능한 단위(callable unit)로, 값(value)을 반환하는 함수(function)와 값을 반환하지 않는 프로시저(procedure)의 두 카테고리로 구분됨





Procedures vs. functions

```
-- gnatmake pgm4.adb
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
procedure pgm4 is

   procedure printTotal(x, y:Integer) is
   begin
      Put(x+y);
   end;

   function getTotal(x:Integer; y:Integer) return Integer is
   begin
      return x+y;
   end;

   z : Integer;
begin
   printTotal(3,4);
   z := getTotal(5,6);
   Put(Z);
end;
```

```
! gfortran pgm4.f -ffree-form -o pgm4.exe
program pgm4
   Integer z
   Integer getTotal
   call printTotal(3,4)
   z = getTotal(5,6)
   print *, z
end

subroutine printTotal(x, y)
   Integer x, y
   print *, x+y
end

integer function getTotal(x, y)
   Integer x, y
   total = x+y
end
```

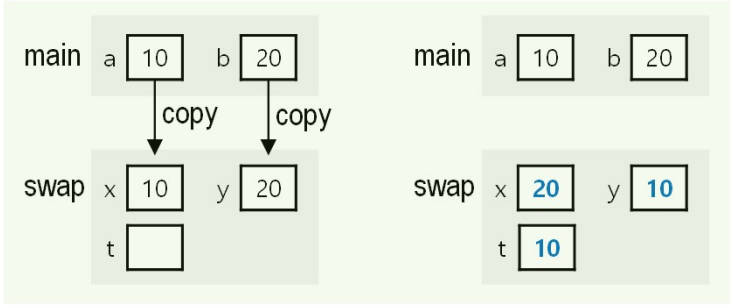
```
// gcc -o pgm4.exe pgm4.c
#include <stdio.h>
int getSum(int x, int y){
   return x+y;
}
void printSum(int x, int y){
   printf("%d\n", x+y);
}
int main(){
   printSum(3,4);
   int z=getSum(5,6);
   printf("%d\n", z);
}
```

파라미터 전달 방법 예 (1/2)

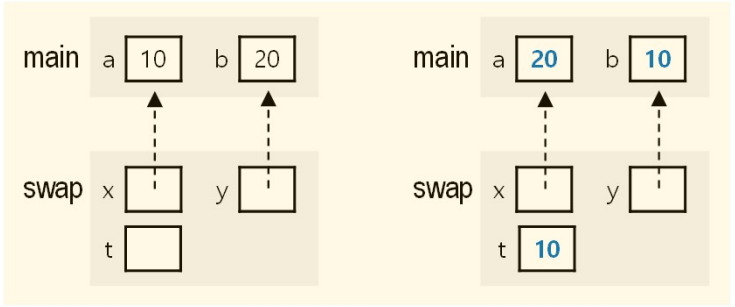
Reference: 창병모. (2021). 프로그래밍 언어론 : 원리와 실제

```
void swap(int x, int y){
   int t=x;
   x=y;
   y=t;
}
int main() {
   int a=10, b=20;
   swap(a, b);
   return 0;
}
```

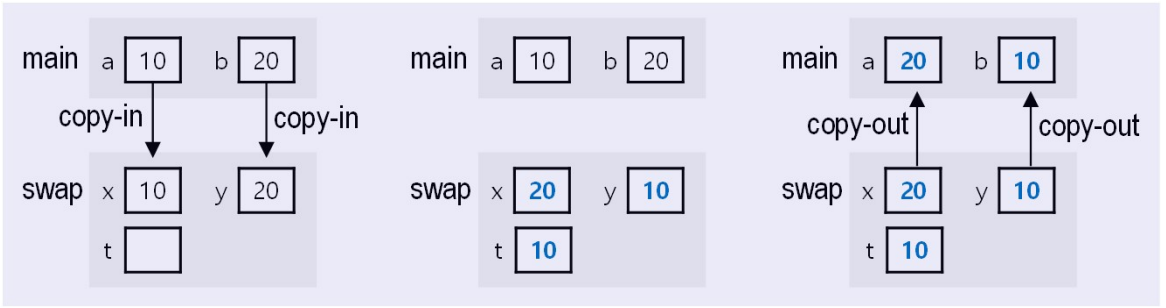
Call-by-value
- 호출 시: argument 값(**value**)을 parameter에 전달 (argument가 수식인 경우 계산된(evaluated) 값을 전달)



Call-by-reference
- 호출 시: argument에 대한 참조(**reference**)를 parameter에 전달



Call-by-value-result
- 호출 시: argument 값(**value**)을 parameter에 전달(복사)
- 리턴 시: parameter 값(**result**)을 argument에 전달(복사)



3가지의 차이

Call-by-Value

함수 호출시 인자 값이 파라미터에 복사된다.

함수 내에서 값을 변경해도 원본 값에 영향X

Call-by-Reference

함수 호출시 메모리 주소가 매개변수에 전달된다.

함수에서 값을 변경하면 원본값도 영향을 줌

Call-by-Result

함수 호출시 인자값을 복사해서 매개변수에 전달

처리만 실행후 매개변수값을 다시 원래변수로 복사

ex) a=10, b=20

swap(a, b) 호출

- ab의 값이 복사되어 전달
- 함수가 종료된후 x=20, b=10 을 a, b로 복사
- 최종적으로 a=20, b=10 이 됨

파라미터 전달 방법 예 (2/2)

Reference: 박두순. (2016). (내공 있는 프로그래머로 길러주는)컴파일러의 이해

```
#include <stdio.h>
void p(int x, int y){
    ++x;
    ++y;
}
int main(){
    int a=1;
    p(a, a);
    printf("a=%d₩n", a);
    return 0;
}
```

- 위 코드가 Call-by-value로 동작한다고 가정하면, 실행결과는 a=1
- 위 코드가 Call-by-reference로 동작한다고 가정하면, 실행결과는 a=3
- 위 코드가 Call-by-value-result로 동작한다고 가정하면, 실행결과는 a=2

Call-by-value-result

copy-in , copy-out

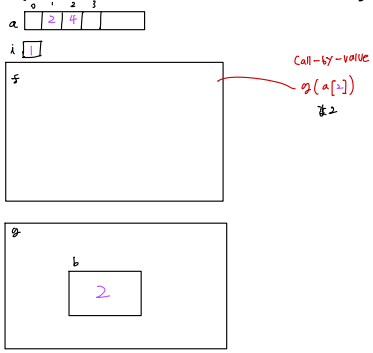
p(a, a)

```
#include <stdio.h>
int a[4], i;
void g(int b){
    a[1]=3;
    i=2;
    printf("%d", b);
}
void f(){
    i=1;
    a[1]=2;
    a[2]=4;
    g(a[i]);
}
int main(){
    f();
    return 0;
}
```

Call-by-name

- 부프로그램 실행 중 parameter 사용될 때마다 대응하는 argument가 계산된(evaluated) 후 사용됨

printf("%d", a[i]); // call-by-name



- 위 코드가 Call-by-reference로 동작한다고 가정하면, 실행결과는 3
- 위 코드가 Call-by-value로 동작한다고 가정하면, 실행결과는 2
- 위 코드가 Call-by-name으로 동작한다고 가정하면, 실행결과는 4

Parameter passing (매개변수 전달)

Call-by-value(값 전달)

- 호출 시 argument의 값(argument가 수식인 경우 계산된(evaluated) 값)을 대응하는 parameter에 전달
- 빠르지만, 복사 방식 전달 시 복사량 많으면 비효율

Call-by-reference(참조 전달)

- 호출 시 argument의 참조(주소)를 대응하는 parameter에 전달(argument는 할당된 저장소가 있는 변수일 것)
- 부프로그램 실행 중 parameter 접근 시 대응하는 argument에 접근하게 됨
- Argument 값의 양에 무관하게 빠르지만, call-by-value에 비해 parameter 값 접근 소요 시간 지연 및 alias로 인한 collision 발생 위험

Call-by-value-result(copy-in copy-out, 값-결과 전달)

- ① 호출 시 argument의 값을 대응하는 parameter에 전달, ② 부프로그램 종료 시 parameter 값(결과값)을 대응하는 argument로 전달
- Call-by-reference에서의 alias로 인한 collision 문제 해결 가능

Call-by-name(이름 전달)

- 부프로그램 실행 중 parameter가 사용될 때마다 대응하는 argument가 매번 계산된(evaluated) 후 사용됨
- 부프로그램 실행 중 미사용 parameter는 대응하는 argument가 계산되지 않음

나중에 질문하기

Call-by-value

Call-by-value(값 전달)

- 호출 시 argument의 값(argument가 수식인 경우 계산된 값)을 대응하는 parameter에 전달
- 빠르지만, 복사 방식 전달 시 복사량 많으면 비효율
- 예) C언어

```c
// gcc pgm5.c -o pgm5.exe
#include <stdio.h>
int f(int x){
    printf("x=%d\n",x);
    x=2*x+1;
    printf("x=%d\n",x);
    return x;
}
int main(){
    int a=5;
    printf("a=%d\n",a);
    printf("%d\n", f(a));
    printf("%d\n", f(a+1));
    printf("a=%d\n",a);
}
```

```
a=5
x=5
x=11
11
x=6
x=13
13
a=5
```

- f(a) 호출 시 a의 값 5가 x에 전달됨
- f(a+1) 호출 시 a+1의 값 6이 x에 전달됨
- f() 내에서 x의 값이 변경되지만 a에는 영향 없음

```c
// gcc pgm6.c -o pgm6.exe
#include <stdio.h>
void swap(int x, int y){
    printf("x=%d, y=%d\n",x,y);
    int temp=x;
    x=y;
    y=temp;
    printf("x=%d, y=%d\n",x,y);
}
int main(){
    int a=3, b=4;
    printf("a=%d, b=%d\n",a,b);
    swap(a,b);
    printf("a=%d, b=%d\n",a,b);
}
```

```
a=3, b=4
x=3, y=4
x=4, y=3
a=3, b=4
```

- swap(a, b) 호출 시 a, b의 값 3, 4가 x, y에 각각 전달됨
- swap() 함수 내에서 x와 y의 값은 교환됨
- Call-by-value가 사용되므로 위 swap() 함수를 통해 a, b의 값은 교환되지 않음

```c
// gcc pgm66.c -o pgm66.exe
#include <stdio.h>
void swap(int *x, int *y){
    printf("x=%d, y=%d\n",*x,*y);
    int temp=*x;
    *x=*y;
    *y=temp;
    printf("x=%d, y=%d\n",*x,*y);
}
int main(){
    int a=3, b=4;
    printf("a=%d, b=%d\n",a,b);
    swap(&a,&b);
    printf("a=%d, b=%d\n",a,b);
}
```

```
a=3, b=4
x=3, y=4
x=4, y=3
a=4, b=3
```

- Call-by-value에서 포인터 변수와 주소 연산자를 이용하여 call-by-reference 구현

Call-by-reference

Call-by-reference(참조전달)

- 호출 시 argument의 접근 경로(예: reference, 참조, address, 주소)를 대응하는 parameter에 전달(따라서 argument는 할당된 저장소가 있는 변수여야 함)
- 부프로그램 실행 중 parameter 접근 시 대응하는 argument에 접근하게 됨
- Argument 값의 양에 무관하게 빠르지만, call-by-value에 비해 parameter 값 접근 소요 시간 지연 및 alias(이명)로 인한 collision 발생 위험
- C++에서는 call-by-value와 call-by-reference가 사용됨. Parameter 변수에 & 사용 시 참조전달 적용
- 예) Fortran, C++

call-by-reference C++ 문법

```
! gfortran pgm5.f -ffree-form -o pgm5.exe
program pgm5
   Integer x, y
   x=3
   y=4
   print *, x, y
   call swap(x,y)
   print *, x, y
end

subroutine swap(x, y)
   Integer x, y, temp
   temp=x
   x=y
   y=temp
end
```

3 4
4 3

```
// g++ pgm7.cpp -o pgm7.exe
#include <stdio.h>
void swap(int &x, int &y){
   printf("x=%d, y=%d\n",x,y);
   int temp=x;
   x=y;
   y=temp;
   printf("x=%d, y=%d\n",x,y);
}
int main(){
   int a=3, b=4;
   printf("a=%d, b=%d\n",a,b);
   swap(a,b);
   printf("a=%d, b=%d\n",a,b);
   // swap(5,6); // 오류
}
```

a=3, b=4
x=3, y=4
x=4, y=3
a=4, b=3

```
// g++ pgm8.cpp -o pgm8.exe
#include <stdio.h>
void ft(int &x, int &y){
   x=2;
   y=3;
}
int main(){
   int a=1;
   ft(a,a);
   printf("%d\n",a);
}
```

3

- 함수 ft()에서 x와 y는 alias(이명) → x와 y는 이름은 다르지만 둘 다 동일한 a를 나타냄
- Alias는 프로그램 의미 파악을 어렵게 함

시험출제

References

- 박두순. (2016). (내공 있는 프로그래머로 길러주는)컴파일러의 이해. 한빛아카데미.
- 창병모. (2021). 프로그래밍 언어론 : 원리와 실제. 인피니티북스.
- Aho, A., Lam, M., Sethi, R., Ullman, J. (2006). Compilers: Principles, Techniques, and Tools (2nd Ed.). Addison Wesley.
- Nystrom, R. Crafting interpreter. https://craftinginterpreters.com/
- Sebesta, R. (2012). Concepts of Programming Languages (10th. ed.). Pearson.
- Thain, D. (2023). Introduction to Compilers and Language Design.
- Paxson, V. (1995). Flex, version 2.5.
- Donnelly, C., Stallman, R. (2008). Bison.
- LexAndYacc.pdf (epaperpress.com)
- Tom Niemann, LEX & YACC. http://epaperpress.com/lexandyacc