12 강

컴퓨터과학 개론

프로그래밍언어(2)

컴퓨터과학과 정광식 교수

학습목차

1 블록과 변수

2 함수의 기본 개념

3 변수의 수명

4 객체 지향 프로그램을 위한 추상 자료형

01

블록과 변수

블록과 변수의 유효 범위

개요

- 변수나 기타 식별자가 프로그램 코드의 유효 범위 존재여부를 결정하는 유효 범위 결정 문제는 프로그램의 실행과 깊은 관련을 가짐

- 변수의 유효범위 문제는

  변수에 대한 기억 장소의 할당 및 유지에 대한 문제임

블록과 변수의 유효 범위

고급언어들

- 대부분 여러 개의 명령문이 모여서 하나의 명령문을 만드는 복합문 및 여러 개의 명령문이 모여 있는 블록을 프로그래밍 언어 내에서 구현함

- 블록을 기초로 변수의 유효 범위를 결정함

```
for ( … )
{                          [블록 1]
    int gx = 10;

  if ( … )
  {                        [블록 2]
      int  y = 2;
      print y;
      print gx;
      int  gx = 20;
      print gx;
  }
  print gx;
}
```

블록과 변수의 유효 범위

개요

- 블록들은 중첩되는 구조도 가질 수 있어서
  블록 안에 다른 블록이 들어가 있을 수 있음

```
for ( … )
{                          [블록 1]
    int gx = 10;

  if ( … )
  {                        [블록 2]
      int  y = 2;
      print y;
      print gx;
      int  gx = 20;
      print gx;
  }
  print gx;
}
```

- 전역 변수(global variable)는 프로그램 코드의 모든 영역에서
  기억 장소의 할당이 유효하며
  지역 변수(local variable)는 그 변수가 정의된 블록 안에서만
  기억 장소의 할당이 유효함

블록과 변수의 유효 범위

개요

- 특정 블록에서 변수가 선언되면, 블록에서 사용될 지역 변수에 대한 기억 장소의 할당이 이루어지고 종료되면 해당 지역 변수는 기억장소에서 삭제됨

블록과 변수의 유효 범위

예)

- 수행 결과 출력 :

  [블록 2; 변수 y]의 값 : 2

  [블록 1; 변수 gx]의 값 : 10

  [블록 2; 변수로 재정의된 gx]의 값 : 20

  [블록 1; 변수 gx]의 원래 값 :10

```
for ( … )
{                          [블록 1]
    int gx = 10;

  if ( … )
  {                        [블록 2]
      int   y = 2;
      print y;
      print gx;
      int   gx = 20;
      print gx;
  }
  print gx;
}
```

블록과 변수의 유효 범위

변수의 유효 범위 문제

- 여러 단계로 중첩된 블록들 사이에서 특정 블록에서 정의되지 않은 변수의 접근 유효성의 결정 문제

- 블록 사이에서의 변수의 유효 범위를 결정하는 기준이 필요함

블록과 변수의 유효 범위

정적 유효 범위 규칙

- 변수의 유효 범위 결정은 컴파일이 이루어지는 시기에 코드에서 가장 가까이 정의된 것으로 유효 범위가 결정됨

동적 유효 범위 규칙

- 코드의 실제 실행 환경에 따라 변수의 유효 범위가 결정됨

블록과 변수의 유효 범위

유효 범위

- 같은 이름의 변수가 프로그램의 여러 곳에서 정의되어 사용될 때, 어디서 정의된 어떤 변수의 값을 참조하고 접근할 것인가는 유효 범위 규칙에 따라 결정됨

```
for ( … )
{                          [블록 1]
    int gx = 10;

  if ( … )
  {                        [블록 2]
      int  y = 2;
      print y;
      print gx;
      int  gx = 20;
      print gx;
  }
  print gx;
}
```

정리 문제

- C 언어에서 int x=10+“hello”; 의 명령이 주어졌을 때 어떤 오류나 경고가 뜨는가 ?

- 문법 오류(syntax error)
- 실행 오류(runtime error)
- 논리 오류(logic error)
- 형(type) 경고

02

함수의 기본 개념

부프로그램: 함수와 프로시저

개요

- 반복 사용되는 코드 부분을 하나의 단위로 묶어서,
  이에 대해 고유의 이름을 정의하고 그 이름을 일반 명령어처럼 사용할 수 있도록 만든 것을 부프로그램이라 함

- 부프로그램은 함수와 프로시저로 구분됨

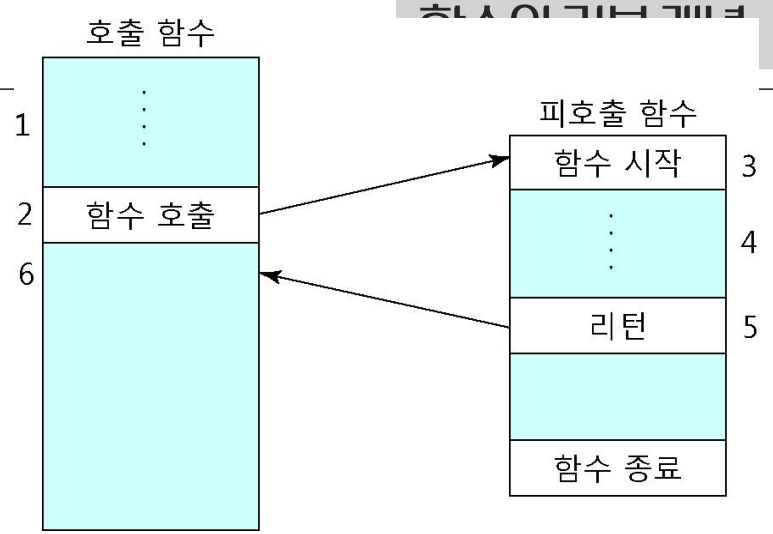
부프로그램: 함수와 프로시저

개요

- 함수와 프로시저는 기능적으로 유사함
  - ✓ 함수 : 함수의 코드 부분의 실행 결과값을 돌려줌(return)
  - ✓ 프로시저 : 실행 결과값을 돌려주지 않음

- C언어나 C++언어 같은 프로그래밍 언어에서는 함수와 프로시저의 구분없이 모두 함수로 취급됨

부프로그램: 함수와 프로시저



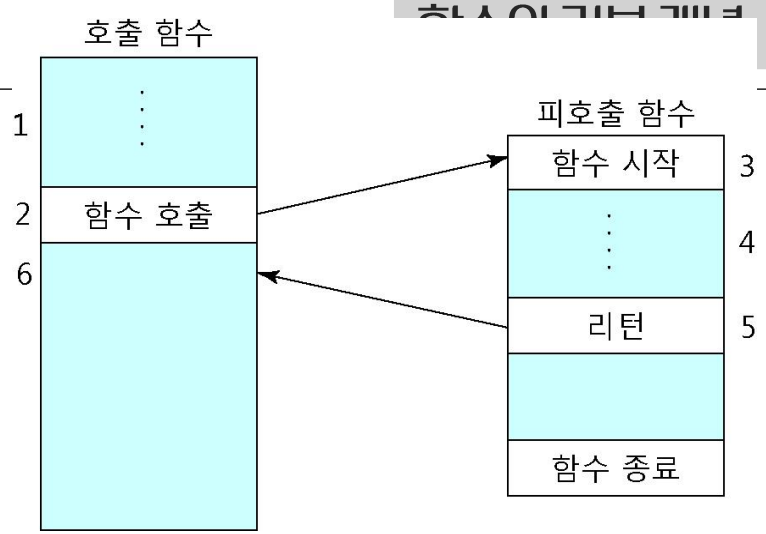
함수는 기본적으로 다음과 같은 요구조건을 가짐

- 제어의 시작이 되는 제어 진입점이 한 곳으로 한정됨
- 함수의 호출이 발생되면, 함수를 호출한 프로그램의 수행이 일시 중단되고 호출된 함수로 실행 제어가 이전됨
- 호출된 함수(피호출 함수)의 실행 중에 함수 종료 조건이 만족되면 실행 제어가 호출한 함수나 호출한 프로그램으로 돌아감

함수 호출과 제어의 이동

함수의 호출

- 호출 함수는 '1번 구역'의 명령어들을 수행하다가 '2번 구역'에서 피호출 함수를 호출함

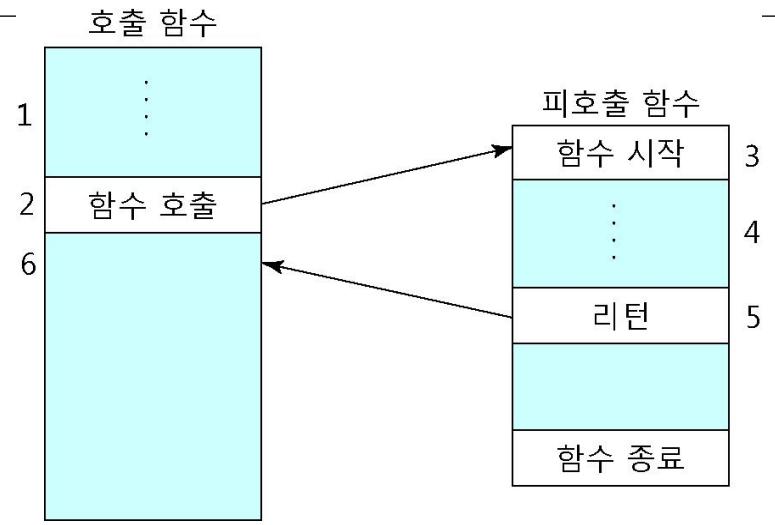


- 피호출 함수의 '함수 호출'이 실행되는 순간, 호출 함수는 실행을 멈추고, 실행 제어는 피호출 함수의 첫 줄인 '함수 시작(3번 구역)'으로 이동되어 '4번 구역'의 명령어들이 수행됨

함수 호출과 제어의 이동

함수의 호출

- 피호출 함수의 실행 중에 '5번 구역'의 반환(return) 조건이 만족되면 피호출 함수의 실행이 멈춰지고 호출 함수로 실행 제어가 되돌아감

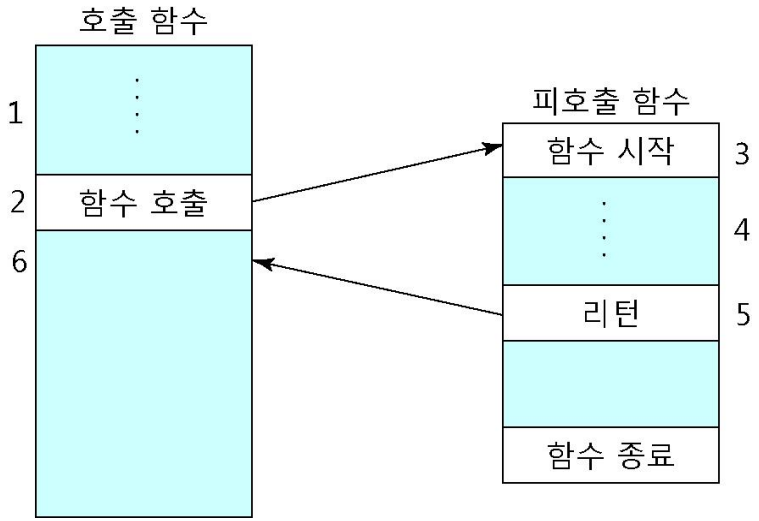


- 실행 제어가 호출 함수에게 돌아오고 나면, 호출 함수의 '함수 호출' 명령어의 다음 명령어 ('6번 구역')부터 수행됨

함수 호출과 제어의 이동

함수의 호출

- 함수의 제어 진입점은 항상 첫 줄이지만
  제어 종료(반환(return))는 여러 조건에 따라
  여러 곳에서 이루어질 수 있음



함수의 매개변수

매개변수

- 호출하는 프로그램(호출 함수)과 호출되는 함수 (피호출 함수)는 서로 주고 받을 정보가 필요함

- 호출하는 프로그램은 피호출 함수의 실행 대상이 되는 데이터를 알려줄 수 있는 매개변수가 있어야 함

- 매개변수(parameter) : 호출하는 프로그램과 호출되는 함수 사이에서 주고받는 데이터임

함수의 매개변수

매개변수

- 함수를 호출하는 프로그램은 호출될 함수 이름만을 지정하기도 하지만, 일반적인 경우에는 피호출 함수에 여러 가지 다른 조건이나 데이터를 전달하고, 이에 따라 피호출 함수가 다양한 기능을 수행할 수 있도록 함
- 피호출 함수에서 처리될 값을 전달하는 매개체 역할을 하고, 매개변수는 특정 데이터형을 가짐

함수의 매개변수

반환값

- 피호출 함수의 실행 결과를 돌려주는 반환값도 특정 데이터 형을 가짐

- C 언어와 유사한 의사코드로 함수를 표현하면 다음과 같은 기본 형태를 가질 수 있음

```
int f (int x) {
        return x+2;
    }
```

함수의 매개변수

매개변수

```
int f (int x) {
        return x+2;
    }
```

```
int val;
int func_val;
val = 10;
func_val = f (val);
```

- 피호출 함수의 정의에 사용된 매개변수(x)를 형식매개변수(formal parameter)라고 함

- 호출 프로그램에서 피호출 함수를 호출하기 위해 사용된 매개변수(val)를 실매개변수(actual parameter)라고 함

함수의 매개변수

매개변수

```
int val;                  변수 val 선언
int func_val;             변수 func_val 선언
val = 10;                 변수 val 초기화
func_val = f (val);       함수 f(x) 호출
```

```
int f (int x) {
        return x+2;
    }
```

- val이 실매개변수이고

  그에 상응하는 형식매개변수가 int f (int x)에서의 x임

- 결과값은 120이고, 그 값이 대입문을 통해 func_val 변수에 저장됨

함수의 매개변수 전달 방식

호출 방식

- 호출하는 프로그램과 호출되는 함수 사이의 매개변수를 전달하는 방식에 따라 값 호출 방식과 참조 호출 방식으로 나뉨

```
void swap (int x, int y) {   x와 y의 값을 교환하는 함수
  int tmp;                    swap 함수내에서만 접근 가능한 지역변수 tmp 선언
  tmp = x;                    매개변수 x값을 지역변수 tmp에 저장
  x = y;                     매개변수 x에 매개변수 y의 값을 대입
  y = tmp;                    매개변수 y에 지역변수 tmp의 값을 대입
   }
```

값 호출 방식

개요

- swap(x, y)를 정의하고, 프로그램에서 swap 함수를 호출함

```
int a = 2;
int b = 3;
swap(a,b);
```

```
void swap (int x, int y) {
  int tmp;
  tmp = x;
  x = y;
  y = tmp;
   }
```

값 호출 방식

개요

- 프로그램에서는 a와 b 변수가 각각 2와 3으로 저장되었고,
  swap(x, y)함수를 swap(2, 3)으로 호출함

- C 언어로 컴파일해서 수행하면 함수 호출 후,
  프로그램에서 a와 b를 출력하면 swap 함수의 실행에도 불구하고
  변수 a와 변수 b 값의 교환이 이루어지지 않음

값 호출 방식

개요

- 교환이 이루어지지 않은 이유는 C 언어에서 함수 호출의 실매개변수를 전달할 때 매개변수 주소를 전달하는 것이 아니고 실매개변수의 값만을 형식매개변수에 복사하는 방식을 취하기 때문임

  ⇒ 값만 보내주는 것과 같음

- swap(2, 3)을 보내게 되는 것과 같다고 할 수 있음
- swap함수의 x, y 매개변수는 a와 b는 전혀 신경을 쓰지 않고 받은 값 2와 3만을 가지고 실행을 하게 됨

값 호출 방식

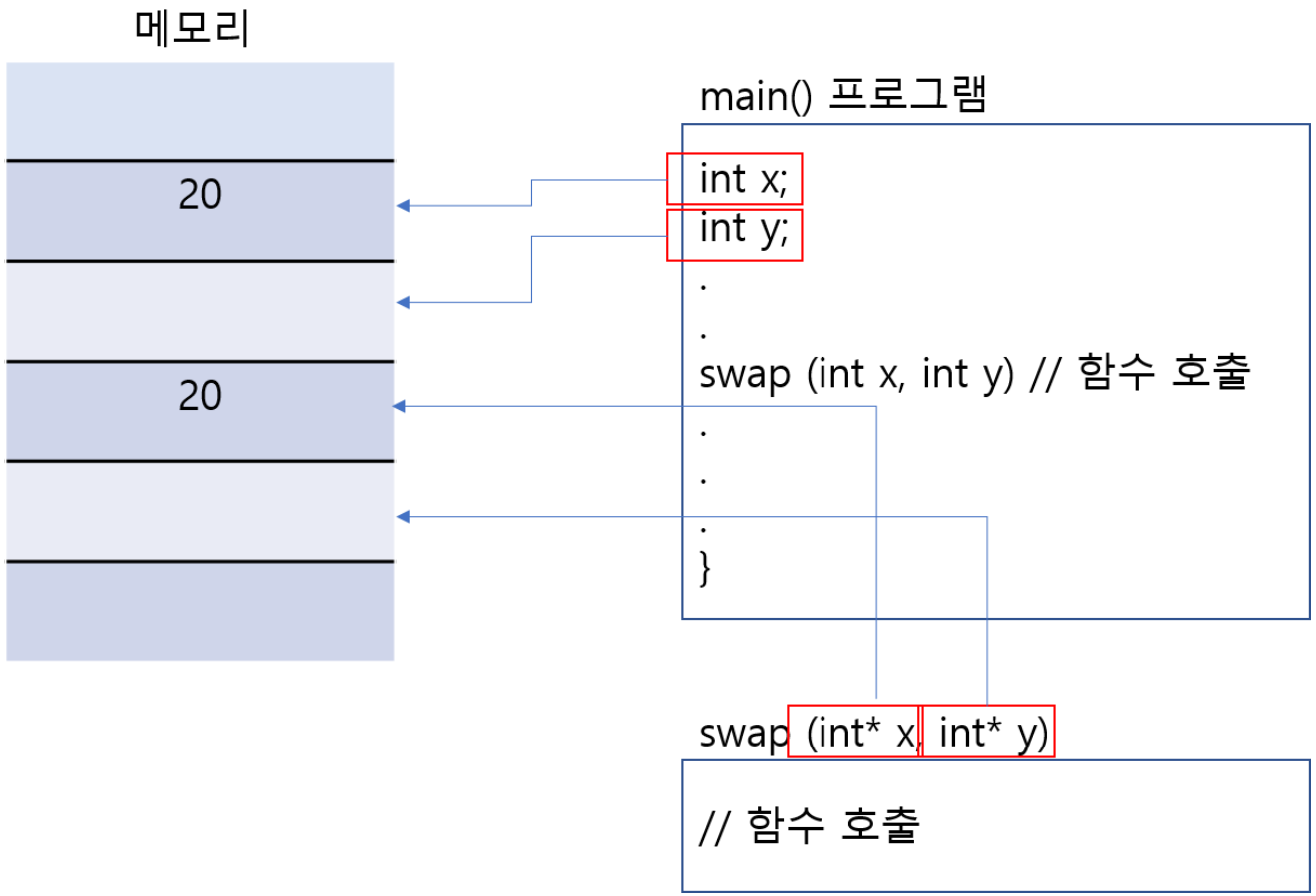
개요

- swap(x,y) 함수 안에서 변수 a와 변수 b의 교환은 복사된 형식매개변수의 값들 간에 교환이 이루어졌기 때문에,
  프로그램 안에서의 실매개변수의 값에는 아무런 영향을 주지 않음

- 이런 방식을 값 호출(call-by-value) 방식이라고 함

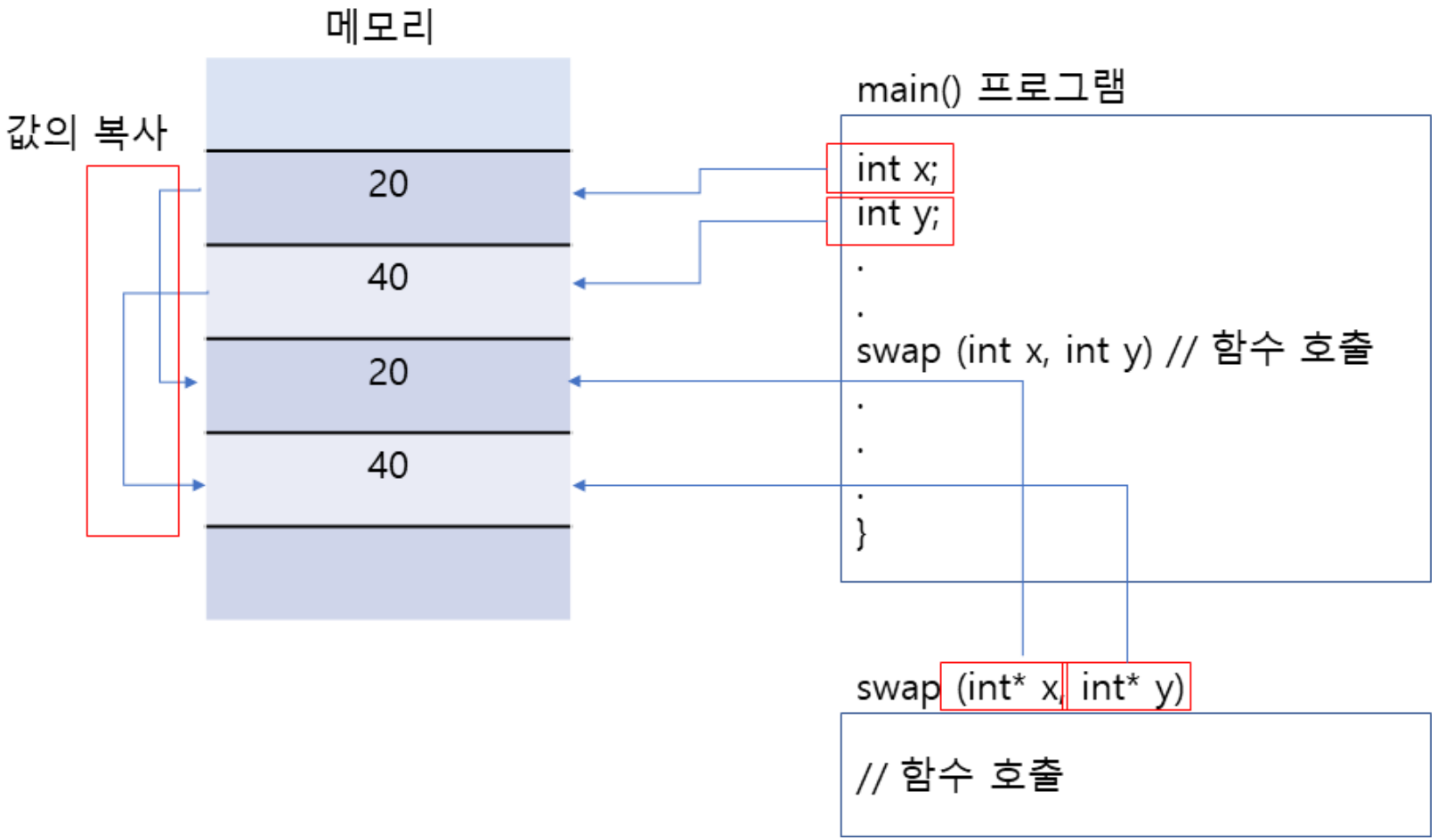
값 호출 방식

개요


메모리
20
20
main() 프로그램
int x;
int y;
.
.
swap (int x, int y) // 함수 호출
.
.
.
}
swap (int* x, int* y)
// 함수 호출

값 호출 방식

개요


메모리
값의 복사
20
40
20
40
main() 프로그램
int x;
int y;
.
.
swap (int x, int y) // 함수 호출
.
.
.
}
swap (int* x, int* y)
// 함수 호출

참조 호출 방식

개요

- 실매개변수가 형식매개변수 자리를 취해서 함수 안에서 형식매개변수에 행해진 모든 조작이 그대로 실매개변수에 반영되는 방식을 참조 호출(call-by-reference) 방식이라고 함

- C 언어에서 함수의 실행 결과를 실매개변수에 반영하기 위해서는 실매개변수의 주소를 호출 함수의 매개변수로 전달함

참조 호출 방식

개요

```
void swap (int* x, int* y) { 매개변수의 포인터값(주소)를 매개변수로 전달받음
    int* tmp;                  함수내에서만 접근 가능한 지역 포인터변수 tmp 선언
   *tmp = *x;                  매개변수 x값을 지역변수 tmp에 저장
   *x = *y;                   y값을 x에 대입
   *y = *tmp;                  tmp값을 y에 대입
  }
```

참조 호출 방식

개요

- 앞에서 '* 연산자'가 두 가지 다른 용도로 사용됨
- 매개변수와 tmp 정의 시에는 정수 포인터 타입(int*)임을 명시함
- 마지막 두 줄에서는 변수 x의 주소가 가리키는 위치에 담긴 내용을 지칭하는 용도로 쓰임

참조 호출 방식

개요

- 함수를 호출할 때는

```
int a = 2;
int b = 3;
swap(&a, &b);
```

- '& 연산자'를 사용해서 a와 b의 주소를 계산한 후 그 주소값을 함수 호출에 사용함

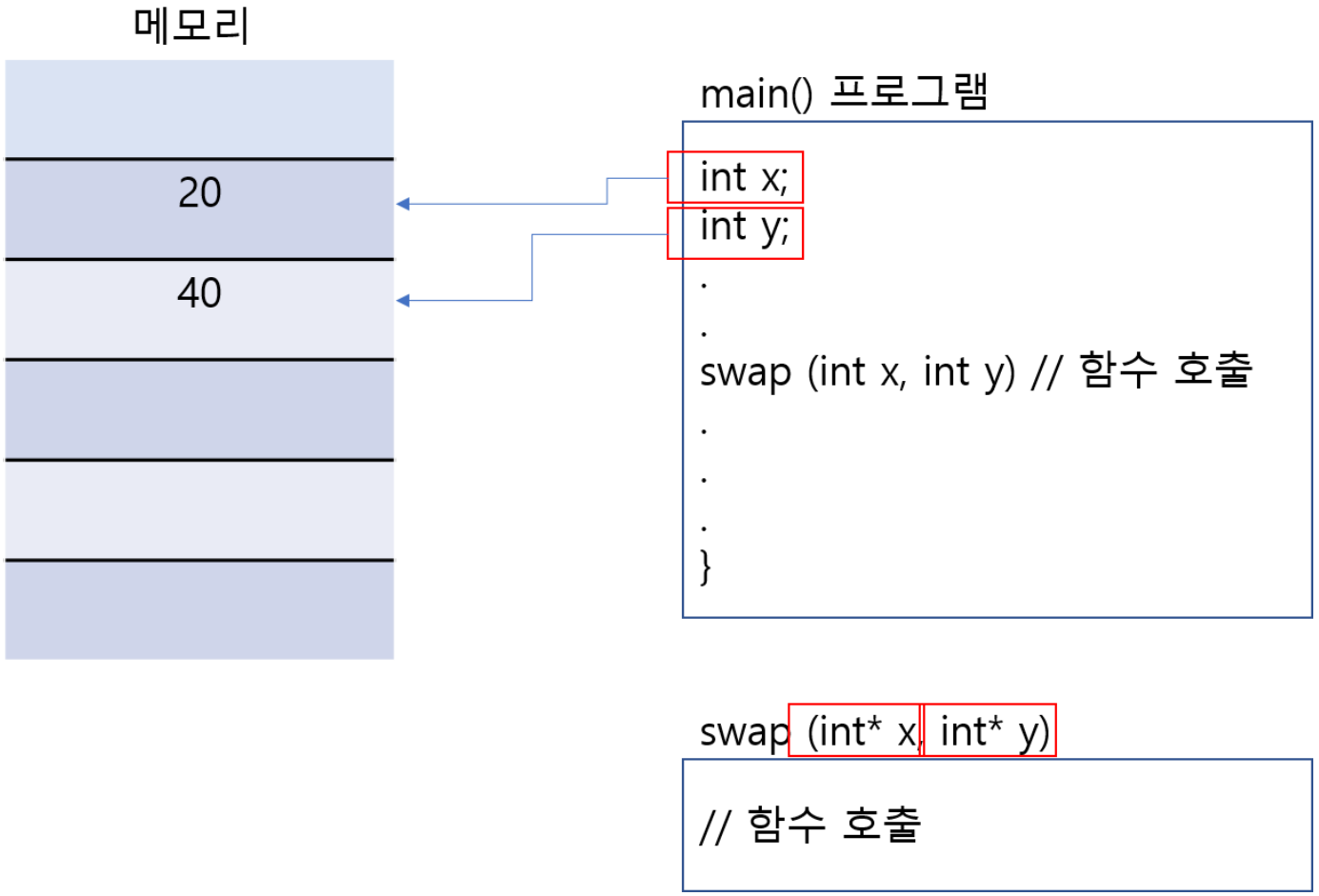
참조 호출 방식

개요

- swap 함수의 결과가 실매개변수에 반영되어,
  a = 3, b = 2로 원하는 결과를 얻을 수 있음

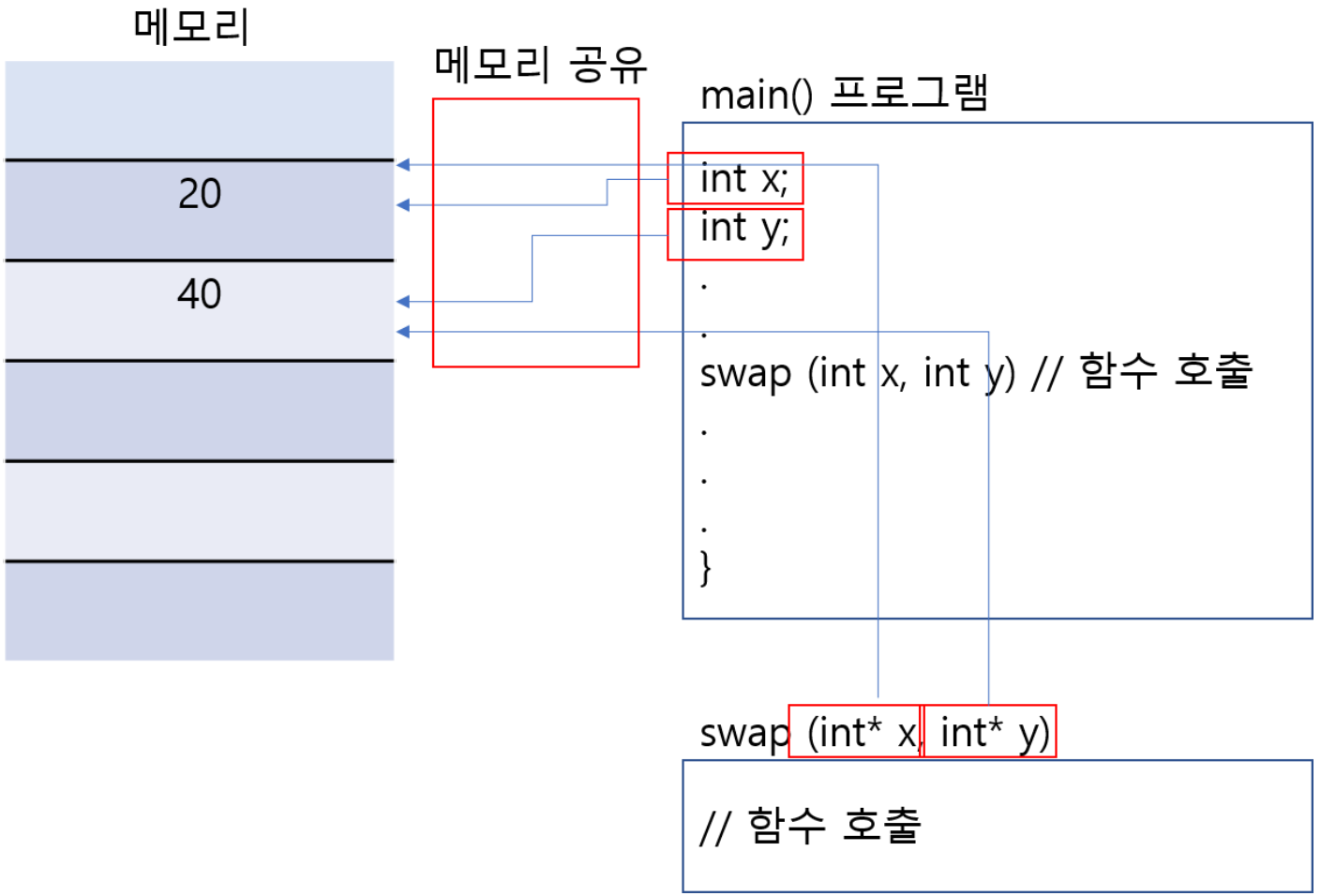
참조 호출 방식

개요


메모리
20
40
main() 프로그램
int x;
int y;
.
.
swap (int x, int y) // 함수 호출
.
.
.
}
swap (int* x, int* y)
// 함수 호출

참조 호출 방식

개요


메모리
메모리 공유
main() 프로그램
20
40
int x;
int y;
.
.
swap (int x, int y) // 함수 호출
.
.
.
}
swap (int* x, int* y)
// 함수 호출

03

변수의 수명

변수의 수명

개요

- 변수의 수명이란

  변수가 값을 저장하기 위해 기억 장소를 할당 받고 있는 시간을 의미함

- 변수의 수명은 그 변수 이름으로 기억 장소의 할당되면서부터

  할당된 기억 장소 해제될 때까지의 시간임

변수의 수명

개요

- 변수의 속성으로 자동 할당, 정적 할당, 프로그래머 지정 할당 등을 이용하여 기억 장소가 할당될 수 있음

자동 할당 방식

개념

- C 언어에서 주로 사용되는 변수를 선언하는 방법임
- 자동 할당 방식에서 변수의 수명은 그 변수가 포함된 블록의 범위와 같음
- 한 변수가 선언된 블록이 시작할 때, 변수는 기억 장소를 할당받고 블록이 끝나면 변수의 기억 장소는 자동적으로 회수됨

정적 할당 방식

개념

- 프로그램이 시작될 때, 기억 장소가 할당되며 블록이 끝나더라도 기억 장소는 그대로 유지되고 프로그램 종료 시 회수됨

프로그래머 지정 할당 방식

개념

- 프로그램의 실행 도중에 프로그래머가 기억 장소를 요청하여 할당받고, 프로그래머가 직접 할당받은 기억 장소를 해제하여 운영체제에게 기억 장소를 회수시킬 때까지 기억 장소가 유지됨

04

객체 지향 프로그램을 위한 추상 자료형

추상화

객체 지향 프로그램을 위한 추상 자료형

개요

- 프로그래밍 언어에서 추상화라는 개념은 필수적인 속성만을 가지고 주어진 것을 묘사함으로써 나머지 부수적이거나 불필요한 속성들은 숨겨지거나 삭제됨

- 공통의 유사성을 표현하고 차이점을 삭제함으로써 동일한 부류의 객체들을 하나로 묶어서 표현하는 방법임

프로시저의 추상화

객체 지향 프로그램을 위한 추상 자료형

개요

- 프로시저의 추상화는 수행 방법을 기술하지 않고 무엇이 수행되는가를 묘사함으로써 추상화시켜 주는 실행 과정의 추상화 기법임

- 프로시저 sort_int(list, list_len)을 이용하여 정수 배열의 정렬 작업을 수행했다면, 호출문은 정렬 작업에 대한 구체적 알고리즘의 명세없이 정렬 작업을 추상화하여 수행한 것임

프로시저의 추상화

객체 지향 프로그램을 위한 추상 자료형

개요

- 프로시저 sort_int의 필수적인 속성은

  정렬할 배열 이름, 배열 원소의 자료형, 배열의 크기이며,

  필수적인 부분만을 프로그래머에게 제공하며 프로그래머는

  실제로 부프로그램 sort_int의 내부 구현 방법이나 알고리즘에

  대해서는 몰라도 됨

- 부차적인 속성(구현 소스 코드, 알고리즘)은 구현된 정렬

  알고리즘으로서 프로그래머의 입장에서 중요하지 않음

자료의 캡슐화

객체 지향 프로그램을 위한 추상 자료형

개요

- 자료 추상화 또는 자료 캡슐화 개념은 프로그램의 재사용을 위해 다양한 이름으로 여러 프로그래밍 언어에 구현되어 있음

자료의 추상화

객체 지향 프로그램을 위한 추상 자료형

개요

- 캡슐화는 프로그래머에게 추상 자료형의 정의된 이름을 통해 객체를 호출하여 사용하도록 하는 윈도우(window)를 제공함

자료의 추상화

객체 지향 프로그램을 위한 추상 자료형

개요

- 윈도우를 통해서 객체의 호출을 외부에 알려주는 부분을 공용부(public part) 또는 가시부(visible part)라 부르고, 캡슐화를 통해 보호되는 구현 부분을 전용부(private part)라 부름

정리 문제

- **프로그래머에게 추상 자료형의 정의된 이름을 통해 객체를 호출하여 사용하도록 하는 윈도우(window)를 제공하는 것은 ?**

- 클래스
- 캡슐화
- 메시지
- 메소드

정리하기

1. 변수의 유효범위

- 변수나 기타 식별자가 코드의 어떤 범위에서 유효한가 하는 유효 범위 결정 문제

2. 함수의 매개변수

- 매개변수 : 호출하는 프로그램과 호출되는 함수 사이에서 주고받는 데이터
- 형식 매개변수 : 호출되는 함수의 정의에 사용된 매개변수
- 실매개변수 : 호출하는 프로그램에서 함수를 호출하기 위해 사용된 매개변수

3. 변수의 수명

- 변수가 값을 저장하기 위해 기억장소를 할당받고 있는 시간

4. 객체 지향 프로그램을 위한 추상 자료형

- 자료와 그 자료를 처리할 연산을 함께 선언할 수 있어야 하며, 선언은 구현에 의존적이어서는 안 되며, 연산의 선언에는 의미에 대한 명세가 포함되어야 함

13 강

다음시간 안내

데이터베이스(1)