Review: Functions

adopted from KNK C Programming : A Modern Approach

countdown.c

```
/* Prints a countdown */

#include <stdio.h>

void print_count(int n)
{
  printf("count %d\n", n);
}

int main(void)
{
  int i;

  for (i = 10; i > 0; --i)
    print_count(i);

  return 0;
}
```

질문:

1. 프로그램이 하는 일은?
2. 함수 print_count가 main 함수 위에 선언된 이유는?
3. 함수 print_count가 main 함수 아래 선언된다면?
4. print_count 앞에 void의 의미는?
5. main앞에 int의 의미는?
6. main뒤에 void의 의미는?
7. 실행 흐름 순서는?

Program: Computing Averages

- A function named `average` that computes the average of two `double` values: average라는 함수는 두 double 타입의 값의 평균을 구함

```
double average(double a, double b)
{
  return (a + b) / 2;
}
```

- the ***return type*** of `average`: `double` at the beginning
  average함수의 리턴 타입(형)은 함수 이름 앞에 표시된 double 이다.
- the ***parameters*** of `average`: `a` and `b`
  a와 b는 average함수가 활용하는 매개 변수이다
  - These represent the numbers that will be given when `average` is called. a와 b는 average가 호출될 때 전달 된 숫자를 나타낸다.
- 질문
  1. 해당 함수는 위의 코드로 실행되는가?
  2. 해당 함수의 실행 결과는 무엇인가?

Program: Computing Averages

- The `average.c` program reads three numbers and uses the `average` function to compute their averages, one pair at a time:
  세 숫자를 입력받아 두 수 씩 평균을 구함

```
Enter three numbers: 3.5 9.6 10.2
3.5, 9.6: 6.55
9.6, 10.2: 9.9
3.5, 10.2: 6.85
```

average.c

```
/* Computes pairwise averages of three numbers */

#include <stdio.h>

double average(double a, double b)
{
  return (a + b) / 2;
}

int main(void)
{
  double x, y, z;

  printf("Enter three numbers: ");
  scanf("%lf%lf%lf", &x, &y, &z);
  printf("%g, %g: %g\n", x, y, average(x, y));
  printf("%g, %g: %g\n", y, z, average(y, z));
  printf("%g, %g: %g\n", x, z, average(x, z));

  return 0;
}
```

질문:

1. 프로그램이 하는 일은?
2. x, y, z 값으로 1, 3, 5를 입력했다고 하자.
3. 첫번째 average 함수 호출 실행 및 결과는?
4. 두번째 average 함수 호출 실행 및 결과는?
5. 세번째 average 함수 호출 실행 및 결과는?
6. average(5.1, 8.9) 혹은 average(x/2, y/3) 로 호출하는 것이 가능?

Array Arguments

- A function can change the elements of an array parameter, and the change is reflected in the corresponding argument.
  함수는 배열의 요소를 변경할 수 있으며, 이 변경은 호출한 곳에서의 배열에 반영된다.
- A function that modifies an array by storing zero into each of its elements: 0으로 초기화하는 함수

```
void store_zeros(int a[], int n)
{
  int i;

  for (i = 0; i < n; i++)
    a[i] = 0;
}
```

Review: Pointers

adopted from KNK C Programming : A Modern Approach

Pointers as Arguments

- 실수에 있어 정수 부분과 소수 부분을 분리하는 함수를 생각해 보자.
  - 그 함수 이름은 `decompose` function 이다.
  - 질문
    - 어떻게 작성할 수 있는가?
    - 작성할 때 마주치는 문제점은?
      - 함수의 입력은?
      - 함수의 출력은?
      - 함수 작성 시 제약 사항은?

- By passing a *pointer* to a variable instead of the *value* of the variable, `decompose` can be fixed.
  변수의 값 대신 변수에 대한 포인터를 전달하는 것으로 문제를 해결할 수 있음

Pointers as Arguments

- New definition of `decompose`: 새로운 decompose 함수의 정의

```
void decompose(double x, long *int_part,
               double *frac_part)
{
  *int_part = (long) x;
  *frac_part = x - *int_part;
}
```

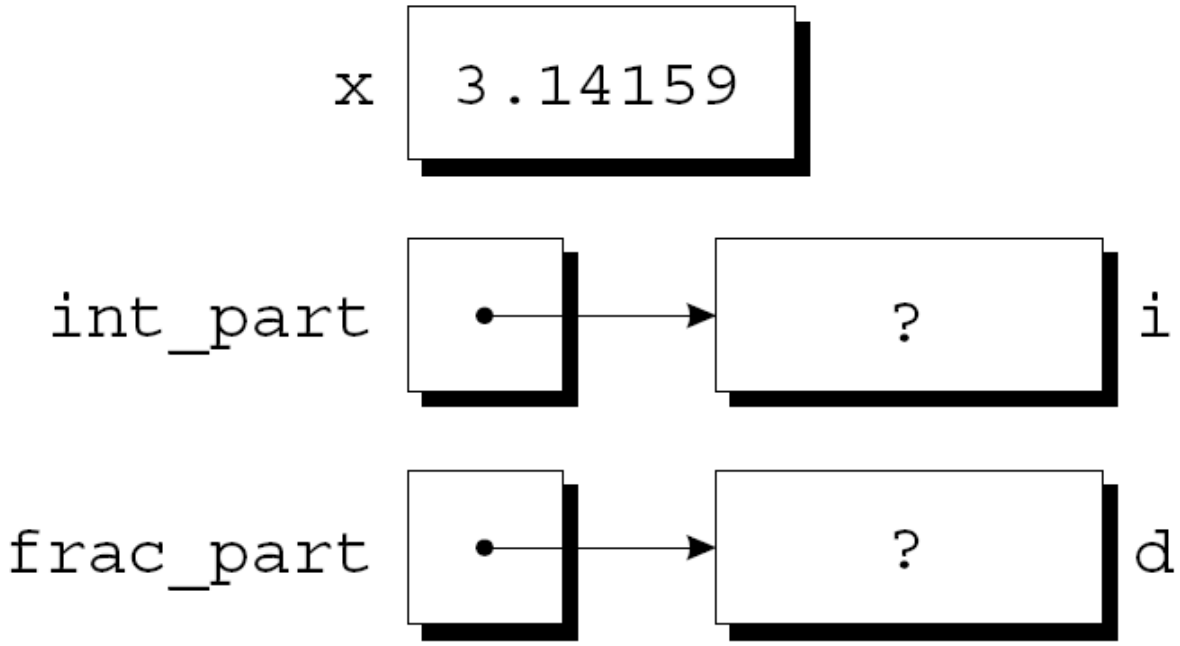
- 질문
  - 이 함수에서 입력 값이 되는 부분은 어디인가?
  - 이 함수에서 돌려주는 값을 나타내는 부분은 무엇인가?
  - 이 함수의 계산 2줄은 무슨 의미인가?
  - 함수 선언은 어떻게 할 수 있는가?

Pointers as Arguments

- A call of `decompose`: 함수 호출 방법

```
decompose(3.14159, &i, &d);
```

- As a result of the call, `int_part` points to `i` and `frac_part` points to `d`:
  호출의 결과로 int_part는 i를 가리키고 frac_part는 d를 가리킴



질문:

1. I에 저장되는 값은?

```
*int_part = (long) x;
```

2. D에 저장되는 값은?

```
*frac_part = x -
*int_part;
```

Review: Pointers and Arrays

adopted from KNK C Programming : A Modern Approach

reverse3.c

```
/* Reverses a series of numbers (pointer version) */

#include <stdio.h>

#define N 10

int main(void)
{
  int a[N], *p;

  printf("Enter %d numbers: ", N);
  for (p = a; p < a + N; p++)
    scanf("%d", p);

  printf("In reverse order:");
  for (p = a + N - 1; p >= a; p--)
    printf(" %d", *p);
  printf("\n");

  return 0;
}
```

질문:

1. 첫 반복문 p=a의 의미는?
2. 첫 반복문 p++의 의미는?
3. 두번째 반복문 p=a+N-1의 의미는?
4. p--의 의미는?
5. 프로그램이 하는 일은?

Review: Structures

adopted from KNK C Programming : A Modern Approach

book.c

```
 struct book {
     char Title[80];
     int Page;
     int Year;
 };


int main() {
     struct book b1, *p;
     scanf("%s", b1.Title);
     scanf("%d", &b1.Page);
     scanf("%d", &b1.Year);
     printf("%s\t%d\t%d\n", b1.Title, b1.Page, b1.Year);
     p = &b1;
     printf("%s\t%d\t%d\n", p->Title, p->Page, p->Year);
}
```

질문:

1. 구조체를 활용한 책 b1 변수 선언은?
2. b1 변수의 각 멤버의 사용자 입력값을 받아들이는 방법은?
3. b1 변수의 각 멤버의 값을 출력하는 방법은?
4. 구조체를 가리키는 포인터 p의 선언과 할당 방법은?
5. P가 구조체의 각 멤버를 가리키는 방법은?

point.c

```
#include <stdio.h>

struct point {
    int x;
    int y;
};


int main() {
    struct point p1 = {1, 3};
    struct point p2 = {5, 7};
    p1 = p2;
    printf("p1=(%d, %d)\n", p1.x, p1.y);
}
```

질문:

1. 프로그램이 하는 일은?
2. p1 = p2가 가능한가?
3. p1 == p2는 가능할까?
4. P1 != p2는 가능할까?

Operations on Structures

- The = operator can be used only with structures of ***compatible*** types. =연산자는 호환가능한 구조체에서만 활용 가능
- In particular, the == and != operators can't be used with structures. 예를 들어 ==나 != 연산자는 쓸 수 없음
- Arrays can't be copied using the = operator.
  배열은 = 연산자 사용 못함
- Some programmers exploit this property by creating "dummy" structures to enclose arrays that will be copied later:
  어떤 프로그래머는 구조체의 기능을 이용해 배열의 값 복사에 활용하기도 함

```
struct { int a[10]; } a1, a2;
a1 = a2;
  /* legal, since a1 and a2 are structures */
```

Review: Structures + Pointers

adopted from KNK C Programming : A Modern Approach

structpoint.c

Structure pointer member can also be accessed using -> operator.
구조체 포인터의 멤버는 ->연산자로 접근 가능

(*personPtr).age is same as personPtr->age (둘은 같은 표현)

(*personPtr).weight is same as personPtr->weight (둘은 같은 표현)

```
#include <stdio.h>
typedef struct person{
   int age;
   float weight;
};
int main(){
  struct person *personPtr, person1;
  personPtr = &person1;

  printf("Enter integer: ");
  scanf("%d",&(*personPtr).age);
  printf("Enter number: ");
  scanf("%f",&(*personPtr).weight);
  printf("Displaying: ");
  printf("%d%f",(*personPtr).age,(*personPtr).weight);
  return 0;
}
```

질문:

1. 프로그램이 하는 일은?
2. 다음의 출력문을 -> 연산으로 표현하면?