C Fundamentals

adopted from KNK C Programming : A Modern Approach

Program: Printing a Pun

- 파일 이름은 자유롭게 짓되 확장자는 항상 .c 로 지정할 것을 권고
  - for example: pun.c

```
#include <stdio.h> // directive 지시자

int main(void)      // function 함수의 시작
{                   /* 문장 시작 */
  printf("To C, or not to C: that is the question.\n");
  return 0;
}                   /* 문장 끝, 함수 끝 */
```

- 컴파일하기

```
% cc -o pun pun.c
```

or

```
% gcc -o pun pun.c
```

- 실행하기 (linux 라는 조건 하에)

```
% ./pun
```

간단한 프로그램의 기본 골격

C 언어의 세 가지 핵심 요소

1 *Directives 지시자* Example: `#include <stdio.h>`

2
```
int main(void)
{
```

시작

3
```
   문장들
}
```

끝

The General Form: Directives 지시자

c 언어의 세 가지 핵심 요소

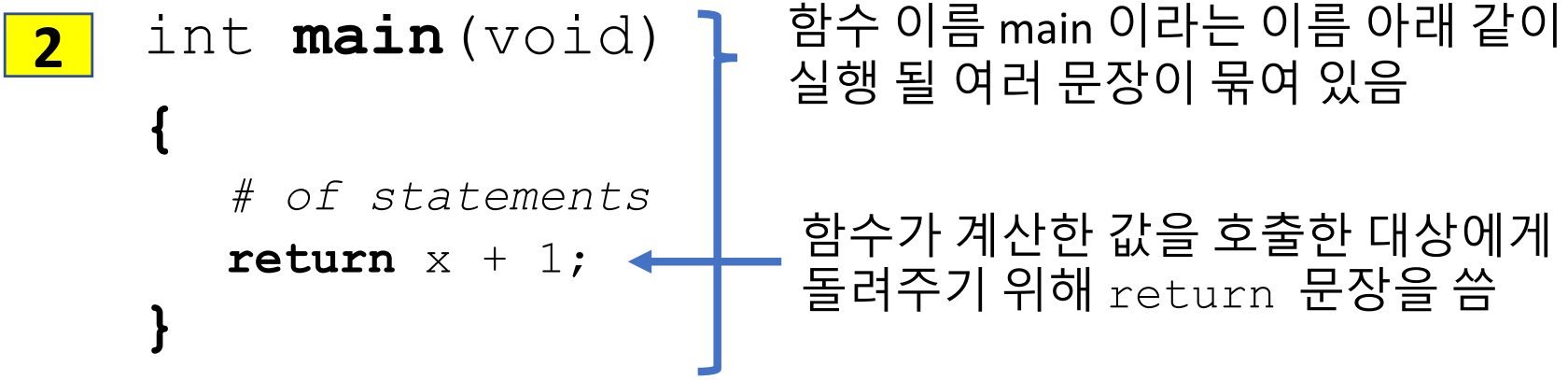
1 *directives* Example: `#include <stdio.h>`

- **c 프로그램이 컴파일되기 전에 전처리기를 거침**
- **지시자에 쓰인 명령들이 전처리기에서 활용됨**

- `<stdio.h>` 는 헤더이고 표준 입출력 라이브러리 함수들의 정의를 포함
- Directives 는 언제나 #으로 시작
- 기본적으로 directives는 **한 줄씩 표현**;
  - 줄의 끝을 알리는 세미콜론이나 특별한 마커가 없음

The General Form: 함수

c 언어의 세 가지 핵심 요소



- ***Library functions*** 들은 c 언어에서 기본으로 제공함
- `main` 함수는 필수적임
  - `main` 특별함: 프로그램이 실행되면 제일 처음 호출되어 실행됨
  - `main` 의 실행 결과로 상태 코드를 리턴함; 0이면 정상 종료를 뜻함
  - 어떤 컴파일러에서는 `return` 문장이 없으면 경고 메시지를 출력함

The General Form: Statements 문장

C 언어의 세 가지 핵심 요소

3 *statements* 프로그램이 동작하면 실행해야 될 명령들로서
모든 문장은 세미콜론으로 마침표를 찍어야 함

- `pun.c` 은 두 종류의 문장을 썼음uses only two kinds of statements.
  - 하나는 `return` 문장; 다른 하나는 ***function call (함수 호출).***

- 함수에 명시된 작업을 처리하라고 시키는 것을 함수를 호출한다라고 함 (***calling*** the function).
- `pun.c` 은 `printf` 를 호출하여 문자를 출력함:

```
printf("To C, or not to C: that is the question.\n");
```

문자열 출력: printf

- printf 함수가 문자열을 출력하면 쌍따옴표로 묶인 모든 글자는 출력하되 쌍따옴표는 출력하지 않음
- printf 는 문장을 출력 후 줄 바꿈을 하지 않음
- printf 에서 문장을 바꾸기 위해서는 \n (the ***new-line character***) 를 쌍따옴표 안에 넣어야 함

같은 효과

```
printf("To C, or not to C: that is the question.\n");
```

```
printf("To C, or not to C: ");
printf("that is the question.\n");
```

```
printf("Brevity is the soul of wit.\n  --Shakespeare\n");
```

more on printf on following section

주석 comment

- ***comment*** 는 /*으로 시작하고 */으로 끝남.

```
/* This is a comment */
```

- 주석은 프로그램 내에 어디든 나타날 수 있음 여러 줄에 걸쳐서 또는 한 줄에도 나올 수 있음
- 한 줄 이상 걸치는 주석

```
/* Name: pun.c
   Purpose: Prints a bad pun.
   Author: K. N. King */
```

- C99 표준에서는 한 줄짜리 주석을 위해 // 을 사용함

```
// A comment, which ends automatically at the end of a line
```

- // 의 장점:
  - 안전함: 한줄만 주석되기 때문에 실수를 줄여줌
  - 여러 줄을 // 로 주석 처리할 때 가독성이 좋음

Variables and Assignment 변수와 할당

- 프로그램이 데이터를 임시로 저장할 수 있어야 함
- 임시로 저장할 수 있는 공간이 변수 ***variables.***

- 변수와 할당을 사용하려면 다음을 알아야 함
  1. Type 형/타입
  2. Declaration 선언
  3. Initialization 초기화

1 2 3

```
int height = 183;
```

Variables and Assignment: 형/타입

- 모든 변수는 ***type***이 있어야 함
  - c 에는 여러 형이 있음 그 중 몇 가지는 `int` 와 `float`.

- `int` (*integer 의/ 약어*) 형은 정수를 저장함 예: 0, 1, 392, or –2553.

- `float` (short for *floating-point 의/ 약어*) 형은 `int` 형 보다 더 큰 수를 표현
  - `float` 형 변수는 소수점도 저장할 수 있음 예: 379.125.

- `float` 변수의 단점:
  - 연산이 늦음
  - `float` 정밀하지 않음

Variables and Assignment: Declarations 선언

- 변수는 먼저 ***declared*** 선언된 후 사용

- 다음과 같이 사용하는 것이 가능하다

```
int height;                int height, length, width, volume;
float profit;              float profit, loss;
```

- `main` 에 선언문들이 포함되어 있다면 이를 활용하는 문장들보다 앞에 있어야 함

```
int main(void)
{
   declarations
   statements
}
```

Variables and Assignment: 할당 (1/2)

- 변수는 할당(***assignment***)을 통해 값을 갖게 됨*:*

```
height = 8;    // 숫자 8은 상수
```

- 변수에 값을 할당하거나 사용하기 전에는 먼저 선언되어야 함

- `float` 형 변수는 소수점 단위까지 저장함

```
profit = 2150.48;
```

- 소수점 단위의 상수 뒤에 `f` 를 붙여서 `float` 변수에 저장하는 것이 좋음:

```
profit = 2150.48f;
```

어떤 컴파일러는 `f` 가 없으면 경고 메시지를 출력함

Variables and Assignment: 할당 (2/2)

- `int` 형 변수는 정수를 저장하고 `float` 형 변수는 실수를 저장함
  - 형을 섞어서 쓰는 것은 가능하지만, 안전하지 않음 (예, `int` 값을 `float` 변수에 할당하기)

- 변수에 값이 할당되면 연산식에 사용될 수 있음

```
height = 8;
length = 12;
width = 10;
volume = height * length * width;  // volume is now 960
```

- 할당자 우변은 식이 올 수 있음. 표현식에는 상수, 변수, 연산자들로 구성됨

Variables and Assignment: Initialization 초기화

- 어떤 변수들은 자동으로 0으로 초기화 되지만, 대부분 그렇지 않다
  - 기본 값이 없는 변수인 경우 그리고 할당되지 않은 변수를 ***uninitialized.*** 초기화 안됨이라고 표현함

- 초기화 안된 변수를 쓰면 예상치 못하는 결과를 얻게 됨
  - 어떤 컴파일러들에서는 프로그램이 멈추는 경우도 생김

- 초기화과정을 선언에 포함시킬 수 있음

```
int height = 8;      // The value 8 is said to be an initializer.
int height = 8, length = 12, width = 10;
int height, length, width = 10; // initializes only width
```

변수의 값 출력하기

- `printf` 를 사용하여 변수의 현재 값을 출력할 수 있음
- 아래와 같이 출력하기 위해 `printf`를 호출하려면:

  `Height: `*`h`*

```
printf("Height: %d\n", height);
```

- `%d` 는 `height` 의 값이 표현될 위치를 표시하는 형식지정자

- 하나의 `printf` 문장에 표현될 수 있는 변수의 개수에 제한 없음:

```
printf("Height: %d  Length: %d\n", height, length);
```

Details of `printf` on following section

입력값 읽기

- scanf 는 printf 의 형제와 같은 입출력 라이브러리 함수이다.
- scanf 는 어떤 형의 값을 저장할 수 있는지 나타내는 형식지정자가 필요
- scanf 가 int 형 값을 읽는 예:

```
scanf("%d", &i);        /* reads an integer; stores into i */
```

- & 심볼은 scanf를 쓰는데 쓰임.

Details of scanf on following section

상수에 이름 부여하기

- 매크로 정의 기능을 사용하여 상수에 이름을 부여 가능:

```
#define INCHES_PER_POUND 166
```

매크로 이름은 대문자로 쓰는 것이 관용적

- 프로그램이 컴파일 되면 전처리기가 매크로가 사용된 곳을 숫자로 치환함
- 전치리 과정에서 다음과 같은 문장은

```
weight = (volume + INCHES_PER_POUND - 1) / INCHES_PER_POUND;
```

다음과 같이 바뀜

```
weight = (volume + 166 - 1) / 166;
```

- 매크로의 값이 수식일 수도 있음:

```
#define RECIPROCAL_OF_PI (1.0f / 3.14159f)
```

- 단, 연산자가 있는 경우 괄호로 묶어야 함

Identifiers 식별자

- ***Identifiers 식별자:*** 변수, 함수, 매크로와 다른 여러 것들의 이름
  - 문자, 숫자, 밑줄이 허용되지만, 숫자 또는 밑줄만 첫글자로 쓸 수 있음:

```
times10   _done          symbol_table   current_page
                         symbolTable    currentPage
```

- c 식별자의 글자수는 제한 없음

- 쓰면 안되는 식별자의 예:

```
10times   get-next-char
```

- c 대소문자 구분 `job  joB  jOb  jOB  Job  JoB  JOb  JOB`

  all are different

Keywords

- 아래의 ***keywords*** 는 식별자로 쓸 수 없음:

auto	do	goto	return	typedef	inline*
break	double	if	short	union	restrict*
case	else	int	signed	unsigned	_Bool*
char	enum	long	sizeof	void	_Complex*
const	extern	register	static	volatile	_Imaginary*
continue	float		struct	while	*C99 only
default	for		switch		

- 키워드는 (예외 `_Bool`, `_Complex`, `_Imaginary`) 모두 소문자로만 쓸 수 있음.
- 라이브러리 함수의 이름 (e.g., `printf`) 역시 소문자임

Layout of a C Program (1/2)

- C 프로그램은 ***tokens***으로 이루어짐.
- 토큰은:
  - Identifiers 식별자
  - Keywords 키워드
  - Operators 연산자
  - Punctuation 구두점
  - Constants 상수
  - String literals 문자

- 다음 문장은

```
printf("Height: %d\n", height);
```

7 개의 토큰으로 이루어짐:

`printf`	Identifier
`(`	Punctuation
`"Height: %d\n"`	String literal
`,`	Punctuation
`height`	Identifier
`)`	Punctuation
`;`	Punctuation

Layout of a C Program (2/2)

- *하나의 문장은* 여러 줄로 나뉠 수 있음
- *토큰 사이의 공백*(연산자 앞 뒤의 공백, 쉼표 뒤의 공백 등) 은 사람이 읽기 편하라고 존재.
- *Indentation* 들여쓰기는 중첩문장들을 읽기 쉽게 해줌
- *Blank lines* 빈 줄은 프로그램을 논리적으로 구성하는데 용이함