Chapter

1

강좌를 시작하면서

너무 깊이, 너무 많이 알려고 하지 마세요

프로그래밍 입문서의 대부분은 컴퓨터의 역사로부터 시작됩니다. 사실 최초의 컴퓨터의 이름이 무엇이고 그 특징이 무엇인지가 여러분들이 프로그래밍 공부하는데 손톱만큼이라도 도움이 되는 지, 저는 상당히 의심스럽습니다.

이러한 내용들은 "불필요하다"라기 보다 입문자에게는 짐만 되는 지식들이라고 생각합니다.

자동차 운전을 위해서 엔진을 만드는 법을 알 필요는 없습니다. 저는 이 강좌를 통해서 최대한 자질구레한 설명을 피하고, 입문자에게 적합한 진행을 하도록 노력하겠습니다.

이해가 안 되는 것은 포기하시고, 쉽게 이해할 수 있는 것부터 시작하세요

너무 깊이, 너무 많이 알려 하지 않고, 적당한 수준을 유지하는 방법은 무엇이 있을까요? 저는 입문자분들에게 항상 충고합니다. "이해 안 되는 것은 포기하세요!"라고 말입니다.

프로그래밍 강의를 수없이 진행해본 필자가 보아온 대부분의 경우에는 입문자 본인이 이해하고 있는 것은 쉽게 무시하고, 이해하지 못하는 곳에 너무 많은 시간과 열정을 쏟아 붇는 것을 보았습니다. 이러한 경우에는 쉽게 지치고 흥미를 잃게 되기 십상입니다.

우선 이해를 할 수 있는 것부터 시작하시다 보면, 그것이 밑거름이 되어, 지금 당장은 이해할 수 없었던 것들도 쉽게 이해할 수 있는 시기가 올 것입니다. "천리 길도 한 걸음부터!!"

어느 정도 현재의 단계가 익숙해지면, 그때 다음 단계를 찾아 심도 깊은 공부를 하시기 바랍니다.

예제 프로그램을 통한 연습만큼 중요한 소스 읽기 연습

프로그래밍이란 여러 가지 기술을 익히고 수많은 지식을 외우는 것으로 익숙해질 수가 없습니다. 많은 시행착오를 거치면서 예제 프로그램을 따라 하고 스스로 새로운

문제를 접하고 풀어나가는 동안 익숙해지는 것입니다. 마치 수학에서 정석을 외웠다고 해서 바로 수학실력이 나아지지 않는 것과 마찬가지입니다.

하지만, 그러한 노력만큼 중요한 것이 바로 다른 사람의 소스를 읽고 이해하는 능력입니다. 프로그래밍도 하나의 언어체계이기 때문에, 마치 영어공부를 할 때처럼 쓰기보다 읽기가 우선되어야 효율적인 프로그래밍 공부가 될 수 있습니다. 읽기는 개념적인 이해를 돕는데 상당히 효과적입니다.

문제는 자신에게 너무 버거운 소스는 피해야 하는데, 입문자에게 딱 어울리는 소스들의 모음은 흔하지는 않습니다. 웹 상의 블로그나 카페 또는 각종 커뮤니티에 올라온 소스들과 책에 설명된 쉬운 소스들을 시작으로 그 소스가 뜻하는 바가 무엇인지 알려고 노력하는 동안 한층 발전된 자신의 모습을 발견하실 수 있을 것입니다.

프로그램이란?

프로그래밍에 대해서는 여러 가지 정의를 내릴 수 있겠지만, 필자의 경우에는 **"프로그렘이란 컴퓨터에게 하달하는 작업 스케줄"**입니다. 즉, 프로그래밍이란 컴퓨터에게 일을 시키기 위해서 컴퓨터가 알아듣는 언어로 작업 스케줄을 작성하는 것입니다.

만약 여러분들이 자동으로 세차하는 프로그램을 만든다고 가정하겠습니다. 이때 작업 스케줄을 우리가 사용하는 언어로 표현하여 아래와 같이 작성해보겠습니다

```
1 : 자동차에 물을 뿌려라.
2 : 자동차에 비누칠을 해라.
3 : 자동차에 솔질을 해라.
4 : 자동차에 물을 뿌려라.
5 : 자동차를 건조기로 말려라.
```

만약 여러분들이 게임을 만들려고 한다고 가정하여 다른 작업 스케줄을 작성하겠습니다.

```
1 : 왼쪽 방향키를 누르면 → 캐릭터를 왼쪽으로 옮겨라.
2 : 오른쪽 방향키를 누르면 → 캐릭터를 오른쪽으로 옮겨라.
3 : 적과 부딪치면 → 캐릭터의 생명포인트를 삭감해라.
```

위에서 예를 든 두 가지 작업 스케줄 중 첫 번째 형식은 절차형 방식이며, 본 강좌에서 사용할 방식의 프로그래밍 기법입니다. 두 번째 형식은 이벤트 처리 방식입니다. 주로 윈도우 프로그래밍을 할 때 사용하는 방식입니다. 절차형 프로그래밍 방식이 순서대로 실행되는 반면, 이벤트 처리 방식은 "어떤 일이 발생하면 무엇을 한다"와 같은 형식으로 프로그램이 작동됩니다.

아직까지는 절차형 방식이 무엇인지 또는 이벤트 처리 방식이 무엇인지 알아두실 필요는 없습니다.

정리하면, **프로그래밍이란 여러분들이 원하는 결과물을 얻기 위해 컴퓨터에게 작업을 지시하는 활동입니다**. 이제부터 필자는 어떻게 작업 스케줄을 작성하면 컴퓨터가

알아듣고 작업을 시작하는 가에 대해서 설명할 것입니다.

컴퓨터에게 작업을 지시하기 위해서는 컴퓨터가 알아들을 수 있는 언어로 이야기하여야 합니다. 컴퓨터 프로그래밍 언어에는 여러 가지 종류가 있으며, 본 강좌에서는 C 언어를 통해서 컴퓨터와 대화를 시도하도록 하겠습니다.

프로그램의 구성 요소

어떤 컴퓨터 프로그래밍 언어를 사용해도, 작성된 모든 프로그램을 크게 두 가지로 나눌 수 있습니다. 또한 이것을 각각 다시 세부 분류로 나눌 수 있으며, 대부분의 컴퓨터 언어는 대동소이한 구조를 가지고 있습니다.

우선 컴퓨터 프로그래밍의 구성 요소는 아래와 같습니다.

프로그램 = 코드(명령어) + 데이터(변수)

예를 들어 컴퓨터 모니터에 글자를 표시하고 싶다면, "화면에 글자 표시"라는 명령어와 "원하는 글자"에 해당하는 데이터를 컴퓨터에게 알려주면 되는 것입니다. (여기서 명령어란 컴퓨터가 알아들을 수 있는 지시어들의 집합을 말하며, 변수라는 것은 데이터를 저장할 수 있는 컴퓨터 프로그램 내부의 공간입니다.)

좀더 컴퓨터 프로그래밍에 가까운 형식으로 변환해서 설명하겠습니다. "화면에 글자 표시"라는 명령어는 C 언어에서는 "printf()"라는 명령어가 있습니다. 정확하게는 함수라고 하지만, 개념을 이해하는데 별로 도움이 되지 않기 때문에 우선 이것에 대한 설명은 넘어가도록 하겠습니다. "화면에 표시할 글자"는 큰 따옴표에 넣어서 표시합니다. 아직 여러분들이 문법을 배우지 않은 관계로 이것을 그냥 아래와 같이 나열하겠습니다.

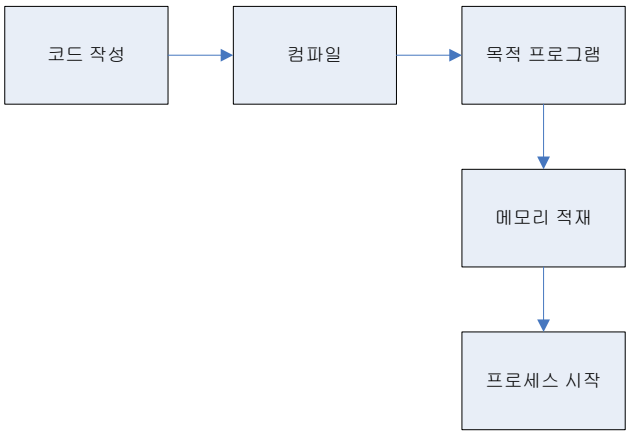
```
printf(), "즐거운 프로그래밍의 세계로 나가자!"
```

사실 위에서 작성한 프로그램은 작동하지 않습니다. 왜냐하면 문법을 어겼기 때문입니다. 사람들끼리의 대화라면 상대가 문법을 다소 잘못 사용했다고 해도 어림짐작으로 알아들을 수 있는 경우가 많지만, 컴퓨터는 문법이 틀리면 전혀 알아듣지 못합니다. (성격이 좀 까다로워요 -.-)

가끔은 데이터가 필요 없는 명령어도 사용됩니다. 하지만, 명령어 없이 데이터만 사용되는 경우는 없습니다.

현실 상황에서 우리가 원하는 결과물은 상당히 복잡한 것들이 많아서 위에서처럼 간단한 문법체계만으로는 원하는 내용을 쉽게 표현할 수는 없습니다. 그래서, 컴퓨터 언어의 문법에도 다양한 구성요소가 있습니다. 그것은 이후에 다시 설명 드리도록 하겠습니다.

프로그램 동작의 원리



[그림 1] 프로그램 동작의 원리

[그림 1]은 프로그램을 작성하고 그것을 실행하는 과정을 간단하게 도식화한 것입니다. 우선 여러분들은 "코드작성" 단계에서 여러분이 원하는 작업 스케줄을 작성합니다. 이 과정을 "코딩" 또는 "코드작성"이라고 합니다.

이렇게 작성된 코드는 컴퓨터가 바로 알아듣지 못합니다. 실제로 컴퓨터가 알아들을 수 있는 언어의 구조는 인간이 이해하기 어려운 형태로 구성되어 있습니다. 따라서, 우리가 컴퓨터 언어구조를 익히는 것은 상당히 어렵기 때문에 중간형태인 프로그래밍 언어로 표현하고 이것을 "컴파일"이라는 과정을 통해서 컴퓨터가 알아들을 수 있는 "목적 프로그램"을 만들게 됩니다.

이제 이 프로그램을 컴퓨터에게 실행하라고 명령합니다. 근래에는 MS사의 윈도우를 사용하는 사람들이 많기 때문에 모니터에 나타나는 아이콘을 더블 클릭하여 실행하시면 됩니다.

하지만, 본 강좌에서는 프로그램을 작성하기 쉬운 도스용(콘솔) 프로그램을 작성할 것입니다. 더블클릭도 가능하지만 프롬프트 상에서 실행하는 것이 일반적입니다. 이 부분은 추후 강좌가 진행되면서 실습을 통해서 자세히 설명하도록 하겠습니다.

이제, 여러분들이 프로그램 실행을 컴퓨터에게 명령하면, 컴퓨터는 여러분들의 프로그램을 메모리에 읽어둡니다. 그리고, "프로세스 시작" 단계에서 컴퓨터는 프로그램의 첫 번째 작업부터 차례로 실행하게 됩니다.

조건의 중요성

로봇에게 매일 아침 세차를 하도록 프로그램을 작성하였다고 가정하겠습니다. 만약 비가 온다면, 세차하는 사람은 거의 없을 것입니다. 하지만, 컴퓨터는 언제나 여러분들이 작성한 프로그램을 차례대로 하나씩 실행하기 때문에, 로봇은 어떠한 경우에도 매일 아침 세차를 할 것입니다.

이와 같은 상황에서는 특정 조건을 설정하여 그 조건이 만족되었을 때만 프로그램을 시작한다던가, 또는, 특정한 조건에서 다른 내용의 작업을 실행해야 할 필요가 있습니다.

조건은 단순한 기계덩어리인 컴퓨터가 마치 인간처럼 사고하는 것처럼 행동할 수 있는 능력을 부여합니다.

Hello, World!

이제 화면에 "Hello, World!"라는 문자열을 표시하는 프로그램을 작성하는 것으로 이번 Chapter를 마치려고 합니다. 아직은 C 언어의 문법체계를 설명 드리지 못하였기 때문에 프로그램을 작성하고 실행하는 전반적인 순서를 익히는 시간이라고 생각하여 주시기 바랍니다.

작업하는 순서는 아래와 같으며, 동영상 설명을 참고하시기 바랍니다.

1. 코드작성
2. 컴파일 (목적프로그램 생성
3. 프로그램 실행

Chapter

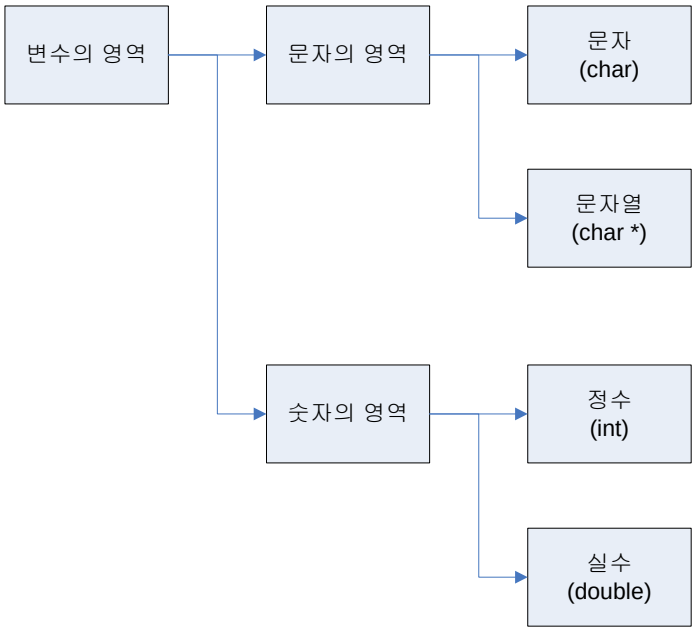
2

기본문법

변수란 무엇인가?

"변수란 데이터를 저장하는 공간"을 뜻 합니다. 특정 데이터를 저장하거나 다른 데이터로 변경이 가능합니다. 또한, 우리가 박스에 물건을 집어넣고, 나중에 찾기 쉽도록 라벨을 붙이는 것처럼, 모든 변수는 변수명이라는 라벨을 갖게 됩니다.

여러분들은 데이터를 저장하거나 읽기 위해서, 데이터를 저장하거나 읽어올 변수의 이름을 알고 있어야 합니다.



[그림 2] 기본적인 변수의 종류

데이터의 경우에는 크게 문자관련 변수와 숫자관련 변수 두 가지로 나뉩니다. 이것들이 쓰임새에 따라 다시 세분화되는 형태입니다.

입문자분들은 익숙해질 때까지는 그냥 변수는 문자형과 숫자형 두 가지로 크게 나뉜다 정도로 알고 계셔도 무방합니다. 다만, 프로그래밍에서 자주 사용하는 변수형태가 다시 문자관련에서 두 가지, 숫자관련에서 두 가지 정도입니다. 따라서, 여러분들께서 초기에 외워두실 것은 아래와 같습니다.

문자관련 변수의 종류

분류	변수형태	크기(바이트)	범위
문자	char	1	0 ~ 255
문자열	char *	미정	

문자는 영문을 기준으로 한 글자에 해당하는 변수타입입니다. 컴퓨터가 서양에서부터 발전한지라, 동양의 문자에 대한 배려가 없었기 때문입니다. (사가지가 없어요, 기냥 ㅡ.ㅡ+)

컴퓨터는 모든 데이터를 숫자로만 표현하기 때문에, 1바이트가 표시할 수 있는 0부터 255번 숫자마다 문자를 하나씩 대입하여 사용합니다.

한글의 경우에는 영문자보다 많은 수의 표시방법이 존재하기 때문에 1바이트 크기의 변수로는 어림도 없습니다. 따라서, 영문자 2개에 해당하는 2바이트 영역을 차지합니다. 하지만, 한글의 경우에는 한 글자에 대한 변수를 지정할 수가 없습니다. char 자체가 1바이트만 차지하기 때문입니다. 따라서, 한글은 무조건 문자열을 사용해서 저장해야 합니다.

문자열의 경우에는 문자의 집합입니다. 즉, char 여러 개가 모인 형태라고 생각하시면 됩니다.

아래는 아스키코드 표라고 불리는 것입니다. 각 문자마다 어떤 값을 가지고 있는지를 알려줍니다.

10진	16진	문자	10진	16진	문자	10진	16진	문자	10진	16진	문자	10진	16진	문자
0	0	Null	47	2F	/	68	44	D	89	59	Y	110	6E	n
7	7	Bell	48	30	0	69	45	E	90	5A	Z	111	6F	o
8	8	BS	49	31	1	70	46	F	91	5B	[	112	70	p
9	9	Tab	50	32	2	71	47	G	92	5C	₩	113	71	q
10	A	LF	51	33	3	72	48	H	93	5D	]	114	72	r
13	D	CR	52	34	4	73	49	I	94	5E	^	115	73	s
32	20	공백	53	35	5	74	4A	J	95	5F	_	116	74	t
33	21	!	54	36	6	75	4B	K	96	60	`	117	75	u
34	22	"	55	37	7	76	4C	L	97	61	a	118	76	v
35	23	#	56	38	8	77	4D	M	98	62	b	119	77	w
36	24	$	57	39	9	78	4E	N	99	63	c	120	78	x
37	25	%	58	3A	:	79	4F	O	100	64	d	121	79	y
38	26	&	59	3B	;	80	50	P	101	65	e	122	7A	z

39	27	'	60	3C	<	81	51	Q	102	66	f	123	7B	{
40	28	(	61	3D	=	82	52	R	103	67	g	124	7C	|
41	29	)	62	3E	>	83	53	S	104	68	h	125	7D	}
42	2A	*	63	3F	?	84	54	T	105	69	i	126	7E	~
43	2B	+	64	40	@	85	55	U	106	6A	j	127	7F	Del
44	2C	,	65	41	A	86	56	V	107	6B	k			
45	2D	-	66	42	B	87	57	W	108	6C	l			
46	2E	.	67	43	C	88	58	X	109	6D	m			

숫자관련 변수

숫자의 경우에는 상당히 다양한 종류의 변수타입이 존재합니다. 종류는 다양하지만 서로 다른 타입(종류)의 숫자관련 변수끼리는 크기의 차이점만 존재한다고 생각하시면 됩니다.

만약 여러분들이 가장 큰 크기를 가지고 있는 변수타입을 사용하신다면, 구태여 다른 타입을 몰라도 됩니다. 하지만, 왜 이렇게 많은 타입이 존재할까요?

대부분의 가정집에 있는 냉장고에는 여러 가지 크기의 반찬통이 있을 겁니다. 왜 같은 크기의 반찬통을 사용하지 않을까요?

작은 반찬 한 가지를 넣기 위해서 냄비만한 반찬통을 사용한다면 공간을 낭비하게 될 것입니다. 이와 같이 프로그래밍에서도 변수가 차지하는 공간의 효율을 위해서 필요한 만큼의 크기의 변수타입을 정할 수 있도록 한 것입니다.

아래는 숫자관련 변수들의 종류입니다. 실제로는 아래의 목록보다 많은 타입의 변수가 존재하지만 우선 입문자에게는 아래의 종류만으로 공부하셔도 무방합니다. 실수형의 경우에는 float보다 double이 더 많이 쓰이는 추세입니다.

분류	변수형태	크기(바이트)	범위
정수	int	4	-2147483648 ~ 2147483647
실수	float	4	2.9*10E-39 ~ 1.7*10E38
	double	8	5.0*10E-324 ~ 1.7*10E308

코드(명령어)의 영역

코드의 구조

[리스트 1] 예제 소스

```
 1 : #include <stdio.h>
 2 :
 3 : #define Max 100
 4 :
 5 : int main(int argc, char* argv[])
 6 : {
 7 :     char Ch;
 8 :     char Ch1, Ch2, Ch3;
 9 :
10 :     char *Str;
11 :     char *Str1, *Str2, *Str3;
12 :
13 :     int iTemp;
14 :     int iTemp1, iTemp2, iTmep3;
15 :
16 :     double bTemp;
17 :     double bTemp1, bTemp2, bTemp3;
18 :
19 :     Ch = 'A';
20 :     Str = "문자열입니다.";
21 :
22 :     iTemp = 1234;
23 :     bTemp = 0.1234;
24 :
25 :     iTemp1 = 1 * Max;
26 :     iTemp1 = 2 * Max;
27 :     iTemp1 = 3 * Max;
28 :
29 :     return 0;
30 : }
```

숫자와 ":"은 각 라인을 구별하기 위해 표시한 것입니다. 실제 프로그램 작성하실 때는 입력하시면 안됩니다.

각 라인마다 ";"이 붙어 다니는 이유는 하나의 문장이 끝났다는 것을 알려주기 위함입니다. C 언어에서는 여러 줄로 나눠 쓰더라도 ";"이 없으면 아직도 한 라인으로 인식하게 됩니다. 마치 우리가 한 문장을 다 쓰면 마침표를 찍는 것과 같습니다.

한 라인에는 한 가지 이상의 명령어 및 선언문이 들어갈 수 없습니다. 또한, 한 라인에는 한 가지 이상의 대입문("=")이 올 수가 없습니다. 대입문과 선언문에 대해서는 예외사항도 있습니다. 아래 8: 라인에서처럼 콤마로 분리하여 여러 선언을 한꺼번에 하는 것이 그 예입니다. 하지만, 그것은 여러 라인을 한꺼번에 묶어 둔 것이라고 생각하는 것이 옳습니다.

프로그래밍을 완료하고 나서 프로그램을 실행하게 되면, main() 함수의 첫 번째 라인에서부터 프로그램이 시작됩니다. 아직 함수가 무엇인지는 배우지 않았기 때문에 일단은 넘어가도록 하겠습니다.

프로그램의 종료는 return을 만나거나 main() 함수의 마지막 라인이 실행되면 이루어집니다.

대입문

대입문이란, 변수에 데이터를 저장할 때 사용하는 문장입니다.

[리스트 1]의 19: 라인에서는 문자 변수에 데이터를 입력하는 방법을 설명하고 있습니다. "=" 표시는 "오른쪽의 데이터를 왼쪽에 있는 변수에 입력하라"라는 뜻입니다. 문자의 표시방법은 작은 따옴표 안에 문자 하나를 입력하시면 됩니다.

[리스트 1]의 20: 라인에서는 문자열 변수에 데이터를 입력하는 방법을 설명하고 있습니다. 문자열의 표시방법은 큰 따옴표 안에 문자열을 입력하시면 됩니다.

[리스트 1]의 22: 라인은 정수를 23: 라인은 실수를 입력하는 방법을 설명하고 있습니다. 실수의 경우에는 소수점을 사용할 수 있다는 것이 정수와 다릅니다.

연산문

연산문은 말 그대로 무엇인가 계산하는 기능을 담당하는 문법입니다. 연산문은 아

래와 같이 분류할 수 있습니다. 이중에서 비트연산은 추후 다시 설명하기로 하겠습니다.

- 산술연산

+	더하기	
-	빼기	
*	곱하기	
/	나누기	3 / 2 → 1 3.0 / 2.0 → 1.5
%	나누기의 몫	정수형태의 숫자에 대한 나누기의 몫 5 % 3 = 2

- 관계연산

두 데이터간의 크기에 관한 정형적인 관계를 나타내주는 연산자를 뜻하며 그 종류는 아래와 같습니다.

==	같다	a == b, a와 b는 같다.
<	작다	a < b, a는 b보다 작다.
>	크다	a > b, a는 b보다 크다.
<=	작거나 같다	a <= b, a는 b보다 작거나 같다
>=	크거나 같다	a >= b, a는 b보다 크거나 같다.
!=	같지 않다	a != b, a와 b는 같지 않다.

- 논리연산

!	아니다	! true → false
&&	그리고	true && true → true true && false → false false && true → false false && false → false
||	혹은	true || true → true true || false → true false || true → true false || false → false

- 비트연산

~	not	반전

&	and	논리곱
|	or	논리합
^	xor	배타적 논리곱
<<	Shift Left	비트 왼쪽 이동
>>	Shift Right	비트 오른쪽 이동

연산문은 주로 대입문하고 자주 사용하게 됩니다. 관계연산과 논리연산의 경우도 대입문을 사용할 수는 있지만, 대부분은 뒤에서 설명할 조건문에서 주로 사용하게 됩니다. 아래 [리스트 2]는 연산문을 통해서 계산된 값을 대입문으로 통해 변수에 저장하는 과정을 예를 들어 보인 것입니다.

[리스트 2] 연산문의 예제 소스

```
 1 : #include <stdio.h>
 2 :
 3 : int main(int argc, char* argv[])
 4 : {
 5 :
 6 :     int iTemp;
 7 :
 8 :     iTemp = 1 + 2 * (5 - 1) / 2;
 9 :
10 :     return 0;
11 : }
```

입출력문

입출력문은 데이터를 특정한 장치에 전달하는 것입니다. 여기서 특정한 장치는 모니터, 프린터, 파일(디스크) 등이 있습니다. 입문자께서 반드시 알아야 할 개념은 아니라고 판단하지만, 내부적으로 모든 장치는 C 언어에서 (대부분의 언어에서) 파일로 취급됩니다.

예를 들어 데이터를 모니터에 전달하면 화면에 데이터(문자나 숫자)가 표시됩니다. 프린터의 경우에는 종이에 문자나 숫자가 출력될 것입니다.

입출력문에는 크게 아래와 같이 분류할 수 있습니다. 이중에서 파일 입출력문은 프린터를 비롯 기타 장비를 다루는데도 사용됩니다. 하지만, 기본문법 과정에서는 콘솔

입출력문만 배워보도록 하겠습니다. 콘솔 입출력문이란 예전 도스용 모니터를 생각하시면 됩니다. 윈도우로 넘어오면서 “명령 프롬프트”라는 이름으로 변경되었습니다. 이것을 콘솔이라고 부릅니다.

- 콘솔 입출력문
- 파일 입출력문

[리스트 3] 콘솔 입출력문의 예제 소스

```
 1 : #include <stdio.h>
 2 :
 3 : int main(int argc, char* argv[])
 4 : {
 5 :
 6 :     int iTemp;
 7 :
 8 :     printf("정수를 입력하세요 : ");
 9 :     scanf("%d", &iTemp);
10 :
11 :     printf("입력된 정수의 값은 : %d", iTemp);
12 :
13 :     return 0;
14 : }
```

[리스트 3]의 9: 라인의 scanf() 함수는 콘솔 입력의 기능을 가진 함수입니다. 이 함수는 아래와 같은 구조로 구성되어있습니다.

scanf(“서식문자열”, &변수1 [,&변수2, …]);

scanf()에서 서식문자열은 오로지 서식만 들어갈 수 있습니다. 서식은 %로 시작되고 영문자 한자리가 따라오게 됩니다. 아래는 자주 사용하는 서식의 종류입니다.

- %d : 정수형 서식
- %s : 문자형 서식
- %d : 실수형 서식

위에서 []는 생략이 가능하다는 뜻입니다. []를 생략했을 경우에는 서식문자열만 출력이 됩니다. 변수는 하나 또는 여러 개가 존재할 수 있습니다. 하나 이상의 변수

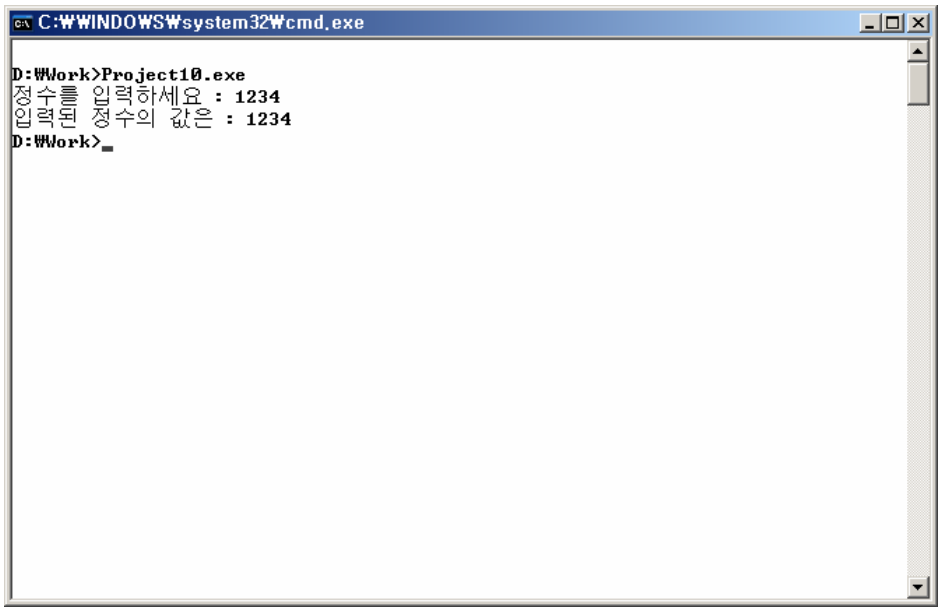
를 전달하고자 할 때에는 콤마로 분리합니다.

scanf() 함수에서는 변수 앞에 & 문자를 붙여서 사용합니다. 이때, &는 연산자의 하나로 변수가 저장되어 있는 메모리 공간의 숫자로 된 주소를 연산합니다. 즉, 무엇인가 콘솔에서 입력 받아와서 변수가 저장된 메모리 공간에 데이터를 입력하겠다는 뜻입니다. 이러한 것을 포인터라고 합니다. 기본문법에서는 다루기 힘든 주제이기 때문에, 이것 역시 나중으로 설명을 미루도록 하겠습니다.

[리스트 3]의 8: 라인의 printf() 함수가 바로 콘솔 출력의 기능을 가진 함수입니다. 이 함수는 아래와 같은 구조로 구성되어있습니다.

printf(“서식문자열”[, 인자1, 인자2, …]);

printf()의 “서식문자열”은 화면에 서식문자열을 출력하되 %로 시작되는 서식이 있으면 해당 서식의 자리에 인자를 대신 출력하게 됩니다.



[그림 3] 리스트3의 실행결과 화면

함수란 프로그램의 구성요소인 명령어와 데이터가 한 개 또는 복수로 구성되어 있는 상태입니다. 특정 목적을 가지고 있는 일련의 명령어와 데이터를 하나로 묶어두고 그것을 함수라는 형식으로 지정하면, 다음부터는 내부에 어떤 명령어와 데이터가 사용되었는지 신경 쓰지 않고, 12: 라인에서처럼 함수 이름만 호출해서 사용할 수 있는 것입니다.

모든 언어에는 수 많은 함수가 미리 제공됩니다. 이것을 표준함수라고 부릅니다. 중요한 표준함수를 외워두시는 것이 좋습니다.

표준함수를 통해서 여러분들은 힘들이지 않고, 미리 제공되는 막강한 기능들을 사용하실 수가 있는 것입니다. 또한, 사용자가 필요에 따라서 추가로 함수를 만들어서 사용할 수도 있습니다.

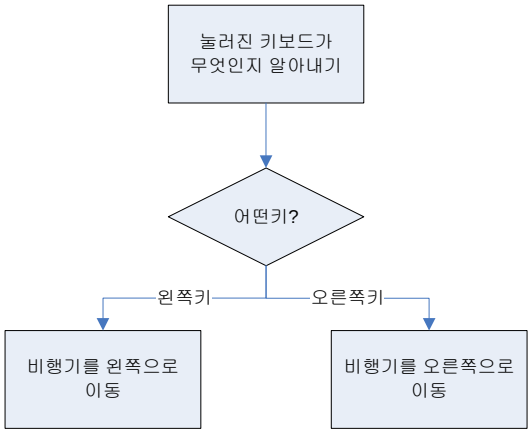
인자란, 함수에 전달하는 데이터(변수)를 뜻합니다. 변수와 인자의 차이점은 변수는 변수만 허용하지만, 인자는 변수와 데이터 모두를 허용합니다.

조건문

조건문이란 특정한 조건이 맞아야만 명령어를 실행하고자 할 때 사용합니다. 조건문에는 아래의 두 가지 종류가 있습니다.

- If
  조건이 맞으면 명령어 실행
- switch
  변수의 값을 조건으로 사용하고, 각각의 변수가 가질 수 있는 값에 대하여 실행문을 배정

조건문은 프로그램이 언제나 똑 같은 동작이 아닌 조건과 상황에 따라 다른 작동을 할 수 있도록 합니다. 비행기 관련된 게임을 만든다고 가정하겠습니다. 여러분들이 키보드를 누르면 비행기의 방향이 바뀌도록 하려 합니다. 왼쪽 방향키를 누르면 왼쪽으로, 오른쪽 방향키를 누르면 오른쪽으로 비행기를 이동하고 싶습니다. 이런 경우 어떤 조건이냐, 즉, 어떤 키가 눌러졌는지 검사하고 그 결과에 따라서 다른 동작을 하고 싶을 때 바로 if문 도는 switch문을 사용하게 됩니다.



[그림 4]

A. if문

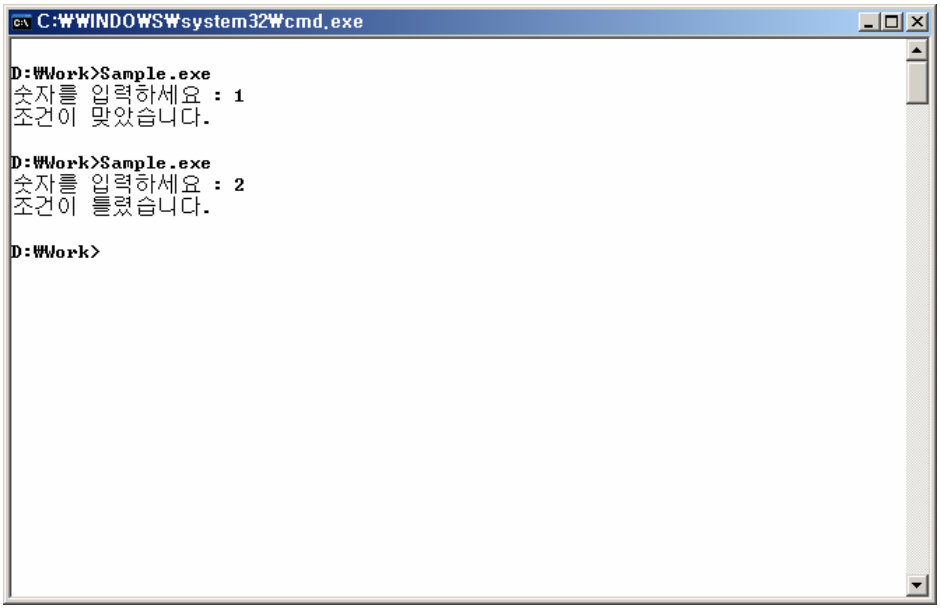
if문의 기본 문법은 아래와 같습니다.

```
If (조건1) 실행문1;

If (조건1) 실행문1;
else 실행문2

if (조건1) 실행문1;
else if (조건2) 실행문2;
else 실행문3;
```

[리스트 4] if문 사용의 예

```
 1 : #include <stdio.h>
 2 :
 3 : int main(int argc, char* argv[])
 4 : {
 5 :     int iTemp;
 6 :
 7 :     printf("숫자를 입력하세요 : ");
 8 :     scanf("%d", &iTemp);
 9 :
10 :     if (iTemp == 1) {
11 :         printf("조건이 맞았습니다.₩n");
12 :     } else {
```

```
13 :        printf("조건이 틀렸습니다.₩n");
14 :    }
15 :
16 :    return 0;
17 : }
```



[그림 5] if문 예제 실행결과

B. switch문

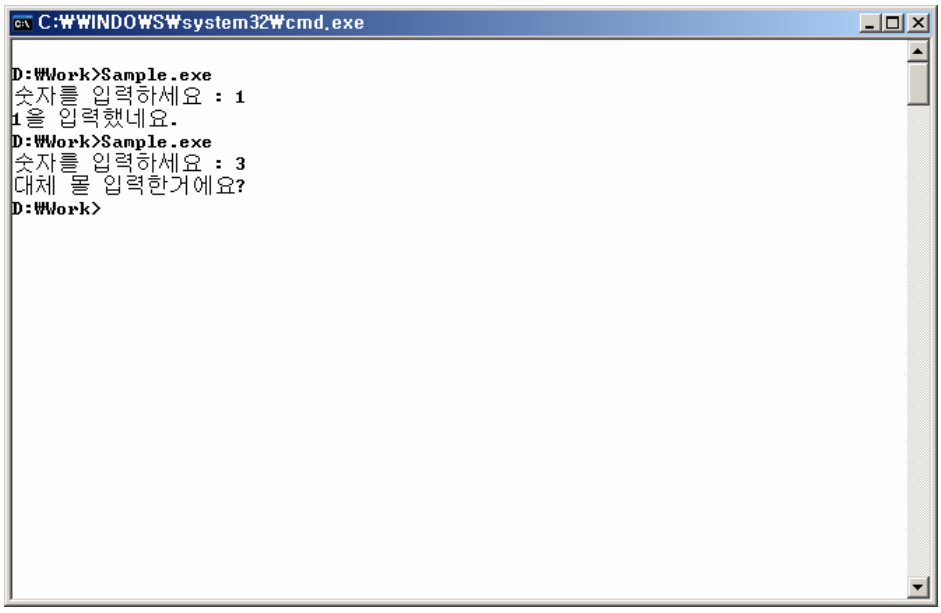
switch문의 기본 문법은 아래와 같습니다.

```
switch (변수) {
    case 값1: 실행문1; break;
    case 값2: 실행문2; break;
    ...
    default: 실행문n; break;
}
```

[리스트 5] switch문 사용의 예

```
1 : #include <stdio.h>
2 :
3 : int main(int argc, char* argv[])
4 : {
```

```
 5 :     int iTemp;
 6 :
 7 :
 8 :     printf("숫자를 입력하세요 : ");
 9 :     scanf("%d", &iTemp);
10 :
11 :     switch (iTemp) {
12 :         case 1: printf("1을 입력했네요."); break;
13 :         case 2: printf("2를 입력했네요."); break;
14 :     default:
15 :         printf("대체 몰 입력한 거에요?"); break;
16 :     }
17 :
18 :     return 0;
19 : }
```



[그림 6] switch문 예제 실행결과

반복문

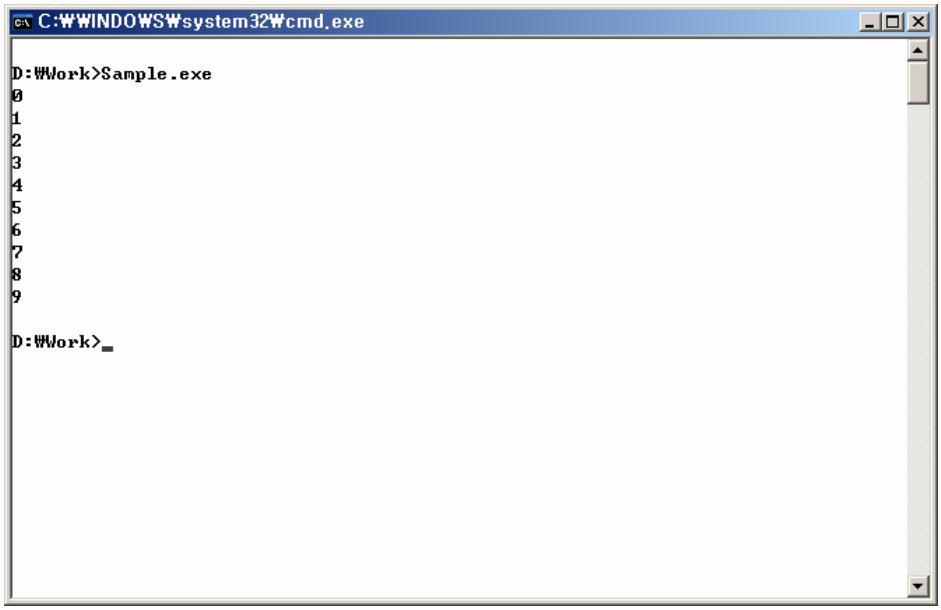
특정 실행문을 여러 번 반복하고 싶을 때 사용하는 것이 반복문입니다. 아래와 같이 세 가지 종류가 있습니다.

- for loop

  주어진 숫자만큼 반복합니다.

[리스트 6] for loop문의 예

```
 1 : #include <stdio.h>
 2 :
 3 : int main(int argc, char* argv[])
 4 : {
 5 :     int iTemp;
 6 :
 7 :     for (iTemp = 0; iTemp < 10; iTemp++) {
 8 :         printf("%d ₩n", iTemp);
 9 :     }
10 :
11 :     return 0;
12 : }
```



[그림 7] for loop문 예제 실행결과

- do while

  조건이 맞을 때까지 반복합니다. 일단 한 번은 실행하고 조건을 검사합니다.

[리스트 7] do while문의 예

```
 1 : #include <stdio.h>
 2 :
 3 : int main(int argc, char* argv[])
 4 : {
 5 :     int iTemp;
 6 :
 7 :     iTemp = 0;
 8 :     do {
 9 :         printf("%d ₩n", iTemp);
10 :         iTemp = iTemp + 1;
11 :     } while (iTemp < 10);
12 :
13 :     return 0;
14 : }
```

- while

  조건이 맞을 때까지 반복합니다. 조건을 먼저 검사하고 실행 여부를 결정합니다.

[리스트 8] while문의 예

```
 1 : #include <stdio.h>
 2 :
 3 : int main(int argc, char* argv[])
 4 : {
 5 :     int iTemp;
 6 :
 7 :     iTemp = 0;
 8 :     while (iTemp < 10) {
 9 :         printf("%d ₩n", iTemp);
10 :         iTemp = iTemp + 1;
11 :     }
12 :
13 :     return 0;
14 : }
```

제어문

제어문은 실행중인 라인의 흐름을 변경하고자 할 때 사용합니다. 아래와 같은 종류들이 있습니다.

- goto

  이동할 곳에 레이블 표시를 해두고, 해당 레이블로 이동하도록 합니다. goto 근래에 들어오면서 사장되어가는 문법입니다. goto문을 사용하면 소스코드의 품질이 떨어지게 됩니다. 최대한 사용하지 않는 것이 좋습니다.

[리스트 9] goto문의 예

```
 1 : #include <stdio.h>
 2 :
 3 : int main(int argc, char* argv[])
 4 : {
 5 :
 6 :     goto GotoHere;
 7 :
 8 :     printf("이것이 화면에 출력될까요?");
 9 :
10 :     GotoHere:
11 :
12 :     return 0;
13 : }
```

- break

  반복문을 빠져나가고 싶을 때 사용합니다.

[리스트 10] break문의 예

```
 1 : #include <stdio.h>
 2 :
 3 : int main(int argc, char* argv[])
 4 : {
 5 :
 6 :     int iTemp;
 7 :
```

```
 8 :     for (iTemp = 0; iTemp < 10; iTemp++) {
 9 :      break;
10 :      printf("이것이 화면에 출력될까요?");
11 :    }
12 :
13 :     return 0;
14 : }
```

- continue

  반복문의 반복을 한 번 지나치고 싶을 때 사용합니다.

[리스트 11] continue문의 예

```
 1 : #include <stdio.h>
 2 :
 3 : int main(int argc, char* argv[])
 4 : {
 5 :
 6 :     int iTemp;
 7 :
 8 :     for (iTemp = 0; iTemp < 10; iTemp++) {
 9 :      if ((iTemp % 3) == 0) continue;
10 :      printf("이것이 화면에 출력될까요? ₩n");
11 :    }
12 :
13 :     return 0;
14 : }
```

- return

  함수의 실행을 마치고 함수를 호출하기 이전 상태로 돌아갑니다.

Chapter

3

프로그래밍 연습 1

프로그래밍의 순서

프로그래밍을 하는 방법론은 상당히 많지만, 교재의 특성에 맞게 아래와 같은 순서로 프로그래밍을 설명 드리도록 하겠습니다.

- 기능분석
- 구현방법 설계
- 프로그램 작성
- 테스트 및 수정
- 완료

기능분석이란, 여러분들이 작성해야 할 프로그램이 가져야 할 기능을 나열하는 것입니다. 기능분석은 여러분들이 작성할 프로그램의 규모나 성격을 파악하는데 상당한 도움이 됩니다.

구현방법 설계는, 구체적으로 프로그램을 어떻게 동작하도록 할 것인가에 대한 동작 설계입니다. 다른 용어로는 동적분석이라고도 합니다. 본 강좌에서는 플로우 차트를 사용하겠습니다.

테스트 및 수정에서는 작성된 프로그램을 동작시켜서, 여러분들이 원하는 결과를 얻을 수 있는 지 확인하고 문제점을 수정하는 과정입니다.

기능분석

1부터 100까지 더하기 프로그램이 가져야 할 기능은 다음과 같습니다.

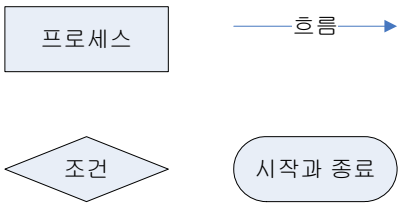
- 1부 100까지 합을 연산한다.
- 결과값을 화면에 출력한다.

기능분석은 프로그램을 작성하기 전에 필요한 기능들을 나열함으로써, 작성하고자 하는 프로그램의 이해를 높이는 과정입니다. 본 예제는 간단한 편이기에 그 필요성이 느껴지지 않을 수 있지만, 실무에서는 종종 처음에 놓쳤던 요구사항을 나중에 알게 되어 개발이 힘들어지는 경우가 많습니다.

만약 이번 예제에서도 기능분석을 하지 않았다면, 1부터 100까지 더하기만 하고 결과값을 출력하는 것을 빼먹는 경우도 가정할 수 있습니다. 컴퓨터 내부에서는 계산이 되었지만 이것을 알 방법이 없다면, 프로그램은 별로 쓸모 없는 존재가 될 것입니다.

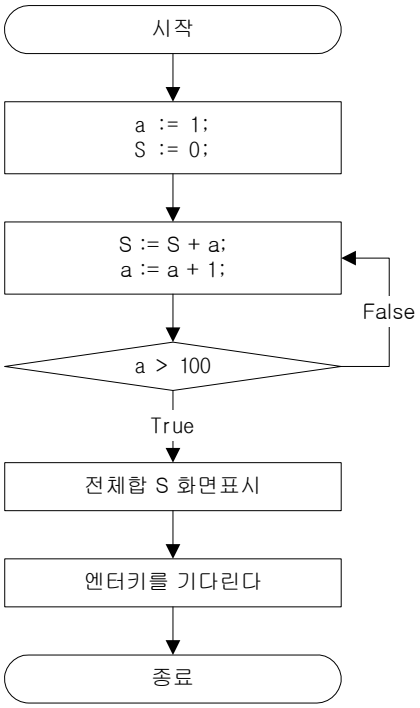
구현방법 설계

구현방법을 설계하는 가장 기본적이고 간단한 방법은 플로우 차트를 그리는 방법입니다.



[그림 8] 플로우 차트의 기본 구성 요소

[그림 8]은 플로우 차트의 기본 구성요소입니다. 출력과 같은 요소 등 다른 종류의 요소들도 존재하지만, 기본적인 것만을 표시했습니다. 그리고, 화면 출력과 같은 요소의 경우는 프로세스의 종류라고 생각할 수 있기 때문에 생략하였습니다.



[그림 9] 1부터 100까지 더하기의 플로우 차트

1부터 100까지 더하는 것에 대한 결과값은 순차수열에 관한 공식으로 쉽게 구할 수 있으나, [그림 9]에서는 실습의 목적상, 프로그램을 통하여 그것을 일일이 더하는 방법을 설명하고 있습니다.

우선 우리가 구하려는 합을 S라고 하면,

$a_n = n$

$S_n = 1 + 2 + 3 \cdots + n$

$S_{100} = 1 + 2 + 3 \cdots + 100$

즉, a_n 항의 n을 1부터 100까지 순차적으로 더해나가면 됩니다.

플로우 차트에서 보면 S_{100}을 S라는 변수로, a_n을 a라는 변수로 지정하여 표현했습니다. 또한 변수 a는 1씩 증가하다가 100이 넘으면 반복되는 루프를 벗어나고 결과값을 화면에 표시한 다음 엔터키를 기다렸다가 프로그램이 종료되도록 되어 있습니다.

반복되는 동안에는

S := S + a;

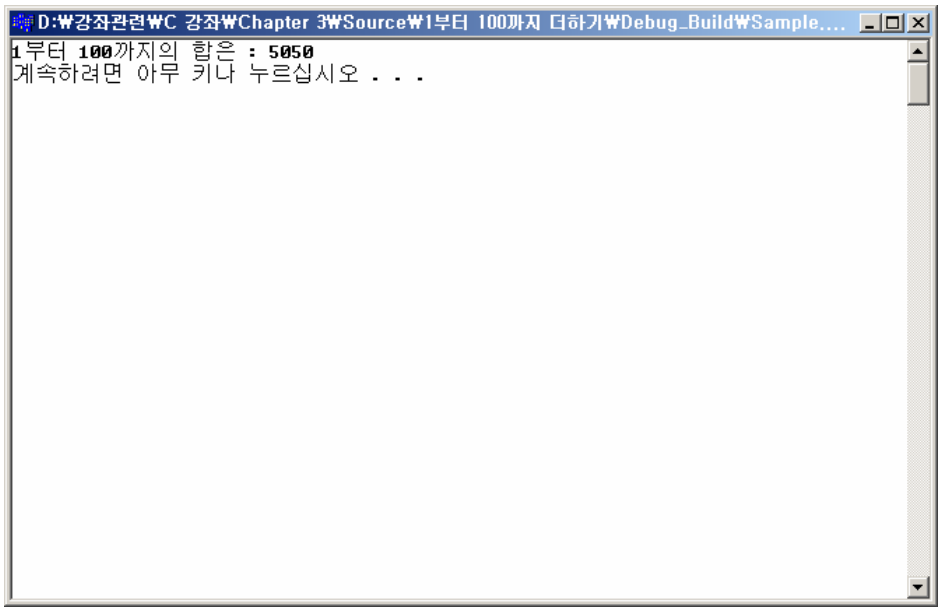
와 같이 각 항이 계속 S라는 변수의 현재 값에 더해지게 되어 결과적으로는

$S_{100} = 1 + 2 + 3 \cdots + 100$

과 같은 결과값이 S라는 변수에 저장될 것입니다.

소스작성 및 실행

소스의 설명은 동영상을 통해서 하도록 하겠습니다.



[그림 10] 결과화면

[리스트 12] 1부터 100까지 더하기의 소스

```
 1 : #include <stdio.h>
 2 :
 3 : int main(int argc, char* argv[])
 4 : {
 5 :     int Loop;
 6 :     int Sum;
 7 :
 8 :     Sum = 0;
 9 :
10 :     for (Loop = 1; Loop <= 100; Loop++) {
11 :         Sum = Sum + Loop;
12 :     }
13 :
14 :     printf("1부터 100까지의 합은 : %d \n", Sum);
15 :
16 :     system("pause");
```

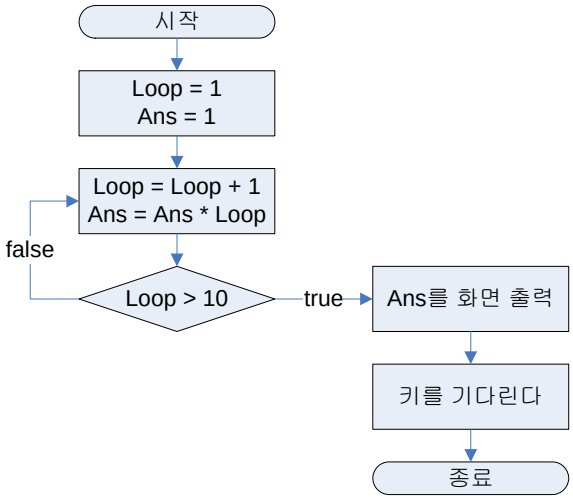
```
17 :
18 :     return 0;
19 : }
```

1부터 10까지 곱하기

기능분석

- 1부터 10까지 곱한다.
- 결과값을 화면에 출력한다.

구현방법 설계



[그림 11] 1부터 10까지 곱하기 플로우 차트

소스작성 및 실행

소스의 설명은 동영상을 통해서 하도록 하겠습니다.

[리스트 13] 1부터 10까지 곱하기 소스

```
1 : #include <stdio.h>
2 :
3 : int main(int argc, char* argv[])
4 : {
5 :     int Loop;
6 :     int Ans;
7 :
```

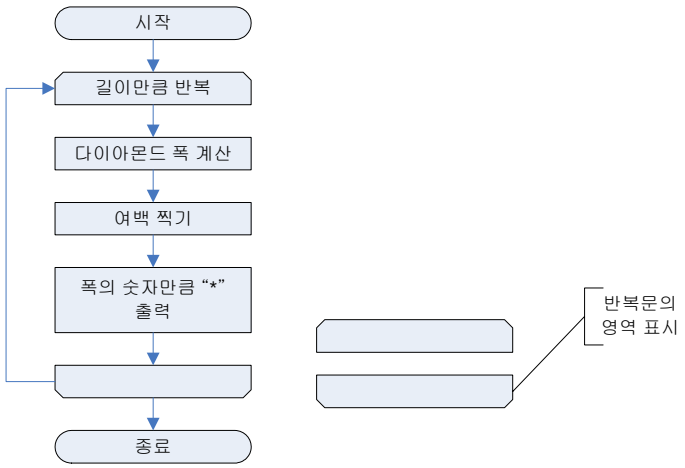
```
 8 :    Ans = 1;
 9 :    for (Loop = 1; Loop <= 10; Loop++) {
10 :        Ans = Ans * Loop;
11 :    }
12 :
13 :    printf("1부터 10가지 곱하기 (10!) : %d \n", Ans);
14 :
15 :    system("pause");
16 :
17 :    return 0;
18 : }
```

다이아몬드 만들기

기능분석

- 화면에 다이아몬드 표시 출력한다.

구현방법 설계



[그림 12] 다이아몬드 만들기 플로우 차트

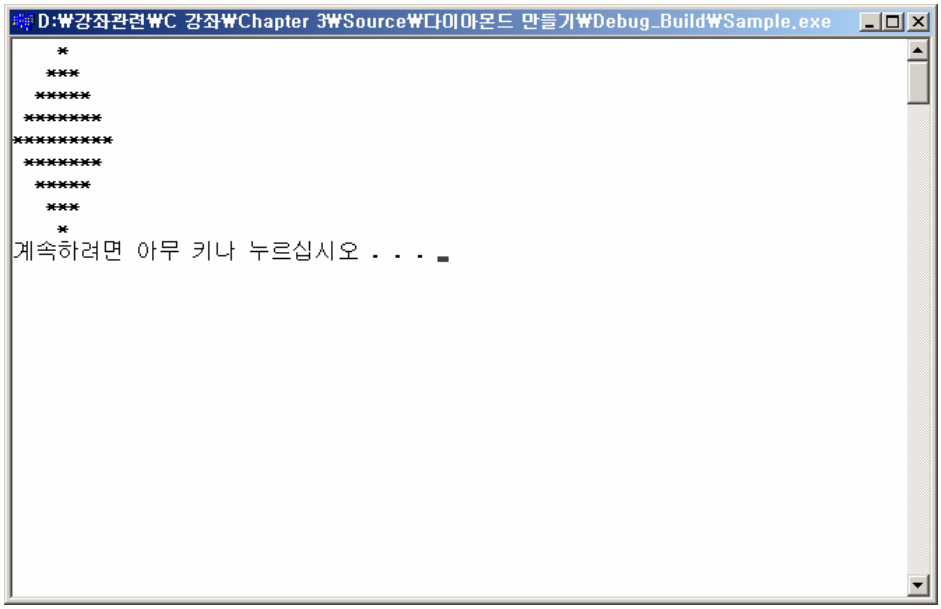
소스작성 및 실행

소스의 설명은 동영상을 통해서 하도록 하겠습니다.

[리스트 14] 다이아몬드 만들기 소스

```
1 : #include <stdio.h>
2 : #include <stdlib.h>
3 :
4 : int main(int argc, char* argv[])
5 : {
6 :     int Loop1, Loop2, Middle, Index, Width;
7 :
8 :     Middle = (Length / 2) + 1;
9 :
```

```
10 :    for (Loop1 = 1; Loop1 <= Length; Loop1++) {
11 :        Index = Middle - abs(Middle-Loop1);
12 :        Width = (Index-1)*2 + 1;
13 :
14 :        for (Loop2 = 1; Loop2 <= Middle-Index; Loop2++) {
15 :            printf(" ");
16 :        }
17 :
18 :        for (Loop2 = 1; Loop2 <= Width; Loop2++) {
19 :            printf("*");
20 :        }
21 :
22 :        printf("\n");
23 :    }
24 :
25 :    system("pause");
26 :
27 :    return 0;
28 : }
```



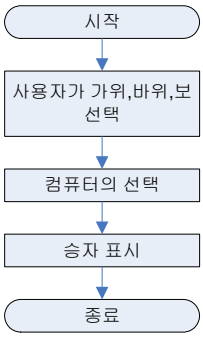
[그림 13] 다이아몬드 만들기 프로그램 결과 화면

가위, 바위, 보 게임

기능분석

- 사용자는 가위, 바위, 보 중 하나를 선택한다.
- 컴퓨터는 무작위로 가위, 바위, 보 중 하나를 선택한다.
- 승리자를 화면에 표시한다.

구현방법 설계



[그림 14] 가위, 바위, 보 게임 플로우 차트

가위, 바위, 보 게임의 승패를 구하는 부분은 입문자에게 어려울 듯 하여 간단한 설명을 하겠습니다.

아래의 [리스트 15]와 달리 다소 단순한 방식도 있긴 합니다만, 조금만 머리를 써서 코드를 작성하도록 하겠습니다. 일단 아래와 같이 가위, 바위, 보를 일렬로 반복해서 늘여 놓도록 하겠습니다.

가위, 바위, 보, 가위, 바위, 보, …

이때 여러분들이 가위, 바위, 보 중 하나를 선택했을 때, 컴퓨터가 두 칸 뒤에 것을 선택하면 컴퓨터가 진 경우가 됩니다. 둘이 같은 것을 내면 비긴 것이며, 위의 두 가지 경우가 아니면 컴퓨터가 이긴 경우입니다.

이처럼 가위, 바위, 보가 순환하는 성질을 이용하면 쉽게 승패를 찾을 수가 있습니다. 자세한 것은 소스를 통해서 설명하도록 하겠습니다.

소스작성 및 실행

소스의 설명은 동영상을 통해서 하도록 하겠습니다.

[리스트 15] 가위, 바위, 보 게임 소스

```
 1 : #include <stdio.h>
 2 : #include <windows.h>
 3 :
 4 : int main(int argc, char* argv[])
 5 : {
 6 :     int PC, Player;
 7 :     char *stPC, *stPlayer;
 8 :
 9 : //    srand((unsigned)time(NULL));
10 :     srand((unsigned)GetTickCount());
11 :
12 :     PC = rand() % 3;
13 :
14 :     printf("가위(1), 바위(2), 보(3)를 선택하세요 : ");
15 :     scanf("%d", &Player);
16 :     Player = Player - 1;
17 :
18 :     switch (PC) {
19 :         case 0 : stPC = "가위"; break;
20 :         case 1 : stPC = "바위"; break;
21 :         case 2 : stPC = "보"; break;
22 :     }
23 :
24 :     switch (Player) {
25 :         case 0 : stPlayer = "가위"; break;
26 :         case 1 : stPlayer = "바위"; break;
27 :         case 2 : stPlayer = "보; break;
28 :     }
29 :
30 :     printf("컴퓨터 : %s, 사용자 : %s \n", stPC, stPlayer);
31 :
32 :     if (PC == Player) {
```

```
33 :        printf("비겼습니다. \n");
34 :    } else if (PC == ((Player+2)%3)) {
35 :        printf("이겼습니다. \n");
36 :    } else {
37 :        printf("졌습니다. \n");
38 :    }
39 :
40 :    system("pause");
41 :
42 :    return 0;
43 : }
```

Chapter

4

고급문법

배열

데이터는 비슷한 유형끼리 뭉치는 경향이 있습니다. 예를 들면 주소록을 관리하는 수첩과 같은 경우입니다. 주소록 비슷한 데이터는 이렇게 같은 곳에서 관리하는 것이 편리할 때가 많습니다. 이것을 프로그래밍에서 표현할 때 우리는 배열을 사용하게 됩니다. 즉, 배열이란 비슷한 복수의 데이터를 한 곳에 저장할 수 있도록 해주는 문법입니다.

아래는 배열의 기본적인 문법입니다.

변수타입 배열명[크기1][크기2]…

배열의 경우에는 비슷한 복수의 데이터를 몇 개를 저장할 것인지, 그 숫자를 정의하도록 되어 있습니다. 또한, 이 크기는 하나가 아니라 여러 개를 지정할 수가 있는데, 이때 크기의 개수만큼을 우리는 차수라고 부릅니다. 크기가 하나만 지정된 상태를 1차원 배열이라고 부릅니다. 차수가 두 개일 때는 마치 엑셀에서 데이터를 저장하는 하나의 칸을 열과 행으로 구분 짓는 것과 마찬가지입니다.

아래의 소스는 1차원 배열과 일반 변수 선언의 차이점을 설명하기 위한 소스입니다.

[리스트 12] 배열의 예제 소스

```
 1 : #include <stdio.h>
 2 :
 3 : int main(int argc, char* argv[])
 4 : {
 5 :     char *Name0, *Name1, *Name2, *Name3, *Name4;
 6 :     char *Names[5];
 7 :
 8 :     Name0 = "Lee";
 9 :     Name1 = "Ryu";
10 :     Name2 = "Do";
11 :     Name3 = "Su";
12 :     Name4 = "Young";
```

```
13 :
14 :     Names[0] = "Lee";
15 :     Names[1] = "Ryu";
16 :     Names[2] = "Do";
17 :     Names[3] = "Su";
18 :     Names[4] = "Young";
19 :
20 :     return 0;
21 : }
```

[리스트 12]에서 보듯이 다섯 개의 이름을 저장하기 위해서 일일이 변수 Name0에서 Name4를 선언하고 입력하는 것보다, Names[] 배열을 선언하고 사용하는 것이 더욱 편리하다는 것을 알 수 있습니다. 또한, 반복문을 사용하게 되면 배열의 경우 [] 안에 첨자(위치를 알리는 숫자)를 이용해서 쉽게 처리할 수 있습니다.

6: 라인에서 5개의 항목을 갖는 배열을 선언하였고, C의 경우 배열의 첨자는 0에서부터 시작되기 때문에 0에서부터 4번이라는 첨자를 사용하여 예제를 작성하였습니다.

여기서는 "배열이란 변수 여러 개를 한꺼번에 선언하고 사용할 수 있는 문법"으로만 기억하시고, 추후 예제를 통해서 다시 설명하도록 하겠습니다.

함수

함수란 프로그램의 미니어처라고 할 수 있습니다. 프로그램이란 "입력데이터→처리→출력"과 같은 일련의 프로세스를 정의한 일정표라고 할 수 있습니다. 만약 여러분들이 복잡한 프로그램을 작성하다 보면, 모든 프로세스의 요소들이 많아지고 프로그램은 점점 더 복잡해지게 됩니다. 이때, 프로그램의 부분 부분들을 작은 조각으로 나누어서 작성하게 되면 좀더 쉽고 효율적인 프로그래밍을 할 수가 있습니다.

함수란 이러한 경우에 프로그램을 작은 조각으로 나눌 수 있는 방법 중에 하나인 것입니다. 티브이나 라디오 등의 전자제품을 열어놓으면 수많은 부품들이 들어있는 것을 보실 수가 있습니다. 대부분 이러한 부품들은 자신이 각자 맡은 임무가 있습니다. 이렇듯 프로그램을 작은 조각으로 나누면서 각자에게 임무를 나누어주면서 분업화하고자 할 때 사용하는 것이 함수라고 생각하시면 되겠습니다.

함수의 문법은 아래와 같습니다.

```
리턴타입 함수명([인자타입1 인자1, 인자타입2 인자2…])
{
    소스 구현부분
    return 리턴 값;
}
```

위의 문법의 형식에서 [] 안에 있는 것은 생략이 가능하다는 뜻입니다. 리턴타입의 경우에는 void를 포함한 모든 변수타입이 가능합니다. void란 아무런 리턴 값도 존재하지 않는다는 뜻입니다.

함수에는 크게 리턴 값이 있는 함수와 없는 함수가 존재합니다. 리턴타입이 void로 선언된 함수의 경우가 리턴 값이 없는 함수이며, 다른 변수타입으로 정의된 함수는 리턴 값이 존재하는 함수입니다.

리턴 값이 존재한다는 것은 함수가 실행된 다음 결과값을 가지게 된다는 뜻이며, 이 모든 것은 [리스트 13]에서 설명하도록 하겠습니다.

[리스트 13] 함수에 대한 예제 소스

```
 1 : #include <stdio.h>
 2 :
 3 : void Print_Header()
 4 : {
 5 :     printf("함수 사용 예제를 시작합니다. \n");
 6 : }
 7 :
 8 : int Sum(int a, int b)
 9 : {
10 :     int c;
11 :
12 :     c = a + b;
13 :
14 :     return c;
15 : //    return a + b;
16 : }
17 :
18 : int main(int argc, char* argv[])
```

```
19 : {
20 :     Print_Header();
21 :
22 :     printf("1 + 2 = %d \n", Sum(1, 2));
23 :
24 :     system("pause");
25 :
26 :     return 0;
27 : }
```

3: - 6: 라인에 선언된 함수는 리턴 값이 없는 void 타입의 함수입니다. 이 함수를 호출하게 되면 5: 라인에 있는 printf()가 실행되면서 화면에 문자열이 출력될 것입니다.

8: - 16: 라인에서 선언된 함수는 리턴 값이 정수형태인 함수입니다. 이 함수는 실행 후에 정수 값의 데이터를 도해낼 것입니다.

20: 라인은 함수를 실행하는 한 방법을 표시한 것입니다. 우리가 printf() 함수를 사용했듯이 함수이름과 괄호표시를 이어서 사용하면 되는 것입니다. printf() 또한 미리 정의된 함수일 뿐입니다.

22: 라인에서는 인자가 필요한 함수이면서 리턴 값이 존재하기 때문에 함수 호출 후 정수 값을 그대로 사용하여 화면에 출력하는 예제입니다. 1과 2의 합인 3을 출력하게 될 것입니다.

15: 라인에서는 주석처리 문자인 "//"을 사용했습니다. 해당 문자 뒤에 있는 모든 소스는 무시됩니다. 예제를 위해서 소스를 약간 풀어 썼을 뿐, 주석 처리된 15: 라인처럼 짧게 코딩 하셔도 됩니다.

구조체

여행을 갈 때 자주 쓰는 물건 중 하나가 치약,비누 여행용 세트가 아닐까 합니다. 물론 따로따로 준비할 수도 있겠지만, 부피도 그렇고 특히 이동성이 문제가 됩니다. 그래서 조그마한 가방 속에 치약과 칫솔 그리고 비누 등 여러 가지가 한꺼번에 들어있는 가방 하나를 달랑 준비하면 세면도구는 신경 쓰지 않아도 되는 것이지요.

비슷한 경우가 프로그래밍의 세계에도 존재합니다. 그것이 바로 서로 관련이 있는

데이터들을 한 덩어리로 묶어두고 싶을 경우입니다.

예를 들면 사원정보라는 데이터는 “사원의 이름, 사원의 생일, 사원의 경력…” 등 갖가지 정보들이 필요하게 됩니다. 이것을 모두 변수로 따로 선언하면 “사원의 숫자 X 정보의 숫자” 만큼의 변수가 필요하게 됩니다. 이것은 소스를 이해하기 힘들 뿐 아니라, 데이터를 처리하는데 상당한 방해가 될 수 있습니다. 사원정보의 경우라면 구조체와 배열을 혼합해서 사용하면 보다 효과적인 표현이 가능합니다.

이때 사용하는 것이 바로 구조체입니다. 구조체의 문법은 아래와 같습니다.

```
struct 구조체타입명 {
  변수선언1;
  변수선언2;
  …
}

변수선언; → 변수타입 변수; or 변수타입 변수1, 변수2…;
```

[리스트 14] 구조체에 대한 예제 소스

```
 1 : #include <stdio.h>
 2 :
 3 : struct TEmployee {
 4 :    char* Name;
 5 :    char* HandPhone;
 6 :    int Age;
 7 : };
 8 :
 9 :
10 : int main(int argc, char* argv[])
11 : {
12 :     int Loop;
13 :     struct TEmployee Employees[10];
14 :
15 :     for (Loop = 0; Loop <= 10; Loop++) {
16 :         Employees[Loop].Name = "Name";
17 :         Employees[Loop].HandPhone = "HandPhone";
```

```
18 :        Employees[Loop].Age = 0;
19 :    }
20 :
21 :    return 0;
22 : }
```

3: 라인에서 선언한 구조체타입명 TEmployee는 13: 라인에서 마치 새로운 변수타입처럼 사용되고 있습니다. 사실 구조체라는 것은 사용자가 새로운 변수타입을 선언하는 것이라고 볼 수 있습니다. 변수타입명 앞에 "T"를 붙인 것은 필자의 개인적인 습관이니 신경 쓰실 필요는 없습니다.

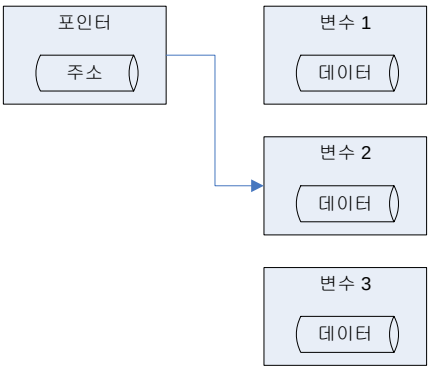
16: 라인에서 보시면, 각각의 구조체에 접근하기 위하여 "."을 사용하는 것을 보실 수가 있습니다. 구조체의 경우 여러 가지 변수를 묶어서 사용하고, 다시 묶여진 변수를 각각 사용할 때는 "구조체변수명.각각의변수명"과 같은 형식을 사용하게 됩니다.

포인터

포인터의 경우에는, 개인적으로, 입문자가 굳이 알 필요가 있을 까 하는 문법입니다. 포인터는 무척 중요합니다. 하지만, 입문자가 자유롭게 다루기에는 너무나 까다로운 문법입니다.

변수를 도서관의 책이라고 생각하겠습니다. 책에는 정보(데이터)가 기록되어 있습니다. 하지만, 도서관에서 책을 찾을 때 그 넓은 공간을 발 품 팔아가면서 일일이 찾아 다니는 것은 너무나 힘든 일입니다. 이때를 위해서 도서관에는 책을 찾을 수 있는 일람카드가 준비되어 있습니다. 책의 제목과 출판사 그리고 저자 등의 정보를 가진 조그마한 카드를 즐비하게 늘어놓고, 열람자는 카드를 찾아내면 카드에 적혀진 책의 위치로 가서 책을 찾으면 됩니다.

이때, 열람카드가 바로 포인터인 것입니다. 데이터가 저장된 곳의 위치를 알려주는 역할을 하고 있습니다. 주로 변수의 메모리 공간에서의 위치정보를 가지고 있는 일종의 변수입니다.



[그림 8] 포인터와 변수의 관계

[그림 8]에서 보시는 것처럼, 포인터는 변수가 있는 주소를 저장하는 변수입니다. 포인터는 실제 데이터를 간직하지는 않습니다. 데이터를 가지고 있는 것은 변수이고, 이 변수 또는 데이터의 위치를 가리키는 것이 포인터입니다. 포인터는 변수를 통하지 않고 데이터의 위치만을 가리키는 경우도 많습니다.

포인터에는 포인터 연산이라는 것이 존재합니다. "&"는 변수의 주소를 읽는 연산자입니다. 창고정리를 예로 들면 입고처리와 비슷합니다. 데이터가 메모리(창고)에 적재되었을 때, 그 위치를 읽어서 포인터 변수에 저장해두고 싶을 경우 사용합니다.

"*"는 출고처리와 비슷합니다. 주소를 가지고 실제 데이터를 꺼내고 싶을 때 사용합니다.

파일처리

파일처리의 경우에는 본 강좌에서 설명하지 않으려고 합니다. 우선 이 강좌는 최소의 지식으로 입문자가 쉽게 프로그래밍을 접하고 여러 가지 응용 법을 배우게 하는데 목적이 있습니다.

더구나, C언어의 기본적인 파일처리 방식은 근래 실무에서 사용되는 방법과 다소 거리가 멀다고 할 수가 있습니다.

따라서, 파일처리 등과 같은 문제들은 코드웨이(http://www.codeway.co.kr/)에서 델파이, C++ 또는 C# 강좌를 참고하시기 바랍니다.