1. 개 요

본 문서는 C++과 C pointer 에 대한 개요를 정리한 것이다.

2. 참고

책명	출판사	저자	비고
Pointer.DOC			

3. 연 혁(Version 관리)

APPENDIX A 를 참조

4. 문제점

인사말

우리가 프로그램을 한다는 것은 보다 더 나은 삶을 위해서다.
C 라는 프로그램을 배우고 C++을 배워나간지도 어언 6 년째다. 내가 인생에 가장 보람을 느꼈던 일들은 친구들에게, 직장동료들에게 인정을 받아가고 누구에게 신뢰를 받았가던 모습들이다.

문서를 정리를 하는 법을 몰라서 노트에 혹은 A4 에 정리하던 학생시절이 떠오른다.
대딩 시절 아는 사람들에게 내가 자주 지껄이는 말이 있다. “우리나라가 이 모양 이꼴이 되고 문화유산이 전승이 안된 것은 기술자를 천시하고 그 기술을 아는 사람들이 그 기술을 책으로 남기지 않아서다”라고 말이다. 그 울분을 토해내던 대학교 앞 곱창집…, 오늘 따라 그 곱창과 소주 그리고 오뎅국물이 그리워진다.

내가 인생동안 할 수 있는 것이 무엇일까?
이젠 학창시절 바라던 축구선수도 야구선수도 그리고 의사도 변호사도 될 수가 없다.
여전히 조선시대의 상인들처럼 프로그래머라면 힘든 직장이고 월급도 쥐꼬리만하고, 누구는 결혼해서 야근을 하고 집에 소홀해지다보니 집안이 콩까루가 됐느니 하는 소리도 들린다.
항상 회사가 힘들어지면 제일 먼저 정리하는게 연구원이라고 하는 소리도 들린다(?).
이렇게 힘든 직장생활을 하고 있지만 항상 나를 추슬려주는게 있다.
인생은 한번 끝나지만 내가 해보고 싶던일을 할 수 있어서 나는 기쁘다.
물론 내가 프로그램의 고수라는 말은 아니다. 단지 할 수 있는 짓 이 이젠 ~~~~~~~~ 삽질밖에 없수니깐
“오~~ 삽질 코리아~~~”
“오~~ 삽질 코리아~~~” “오~~오 코레코레아”
ㅎㅎㅎ
이젠 본론으로 들어가겠다.
하나의 프로그램 코딩을 이해하는데 다소 시간이 걸리지만 문서정리에는 더 많은 시간이 걸린다.
수학능력시험에 유형이 있고 수영하는데 영법이 있다.
삽질(?)에도 방법이 있고 집짓는데도 공법이 있고 글 쓰는데도 그 이치가 있다.
그리고 사람의 머리와 기억력에는 한계가 있다.
프로그램을 하면서 느끼는게 있다. OS(운영체제)는 어찌 그리 우리 인간들의 머리 작동원리와 비슷한지, 놀랍기만 하다. 라이프니찌가 만든 뭐시기라는 기계도 동양의 주역(이진법)을 기반으로 만들었다고 한다.
-물론 서양에도 주역에대한 번역서가 만타고 하는데.
“One Cpu has Only One Job; 한 대가리에 한가지일밖에 못해.”
“Context Switching 무지 어려버; 스타크래프트처럼 멀티 까기 열라 힘들어…”
멀티 까면 손이 불나는게 아니라 머리가 불난다.
“ Volatile Ram;우리 대가리 무지하게 휘발이 잘돼(잘 잊어 묵는다)”

중딩시절 난 내 머리가 좋은 줄 알았다 한번 외우면 한달이상을 갔기때문이다.
대딩을 거치면서 내 머리가 그리 좋지 않다는 것을 느꼈다(술 마니 먹어서 그러나~~). 일주일 내내 외웠던 내용이 3~5 일을 못가는 것이었다. 난 대책을 찾아야 했다. 머리좋은 약을 사먹던지 노트 정리를 하던지….
그러나 볼펜으로 쓴다는 것은 너무 많은 시간이 걸렸다.
최근에 회사에 들어와서야 비로소 문서작성법을 배울 수 있었다. 문서작성법 난 프로그램 알고리즘을 공부한 이후로 가장 나를 환희에 몰아세웠던 기술이고 방법이었다. 내 기억과 지식을 저장할 수 있는 방법.
부디 이글을 보시는 분들은 어느 문서편집기든 상관 없이 여러분의 지식을 그림으로 글로 저장하시기를 바랍니다. 여러분이 작성한 문서가 여러분의 지식을 확고하게 해줄겁니다.

여기까지 읽어주셔서 고맙습니다.(**혹시 이문서가 괜찮다면 혼자만 보지 마시고,이 글을 쓴 저자에 취지에 맞게(당근 기술을 공유하여 Win-Win), 주변분들에게도 이문서를 공유하여 주세요**.)
대충 이렇게 정리 할려고 했는데 안되네요.
문서정리 방법
1. 정의
2. 소스
3. 분석
3.1. 초기화면
3.2. 중간 Trace 와 결과 & Debugging 화면

1. pointer?

⇨ 1 차원 배열을 가리키는 주소값(4bytes : 32 비트 프로그램, 단 16 비트 도스는 포인터가 16 비트)
⇨ 보통 포인터의 크기는 sizeof(int)와 동일한 크기로 설정된다.

1)1 차원 포인터 변수 - 포인터는 주소를 가리키는 것으로서 1 차원적 (정수,char 등등)배열을 가리키고 있다.
2)1 차원배열 -문자열경우는 NULL 문자가 나오면 문자의 끝이된다.

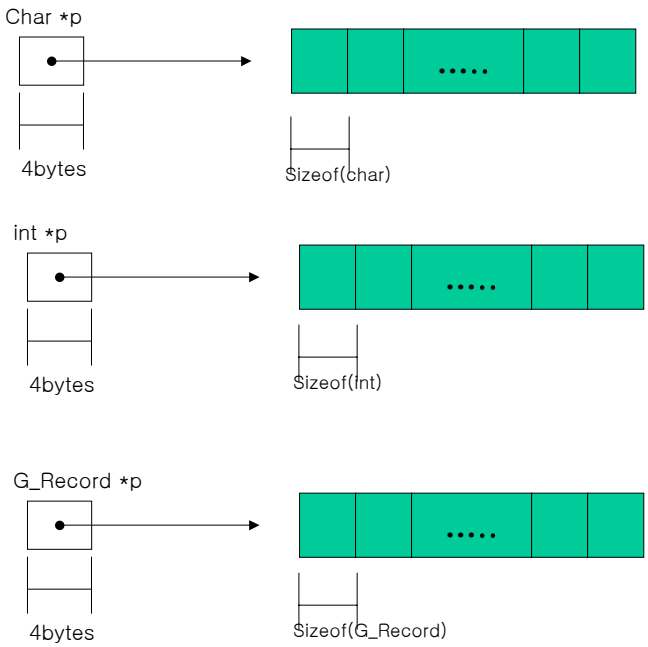
3) 의미
단. Array : n 개 배열. (n: 자연수)
가. **char *p;** **charater type 의 Array** 를 **point(가리키는)**하는 주소값을 담고 있는 변수 **p(4bytes)**

나. **int *p;** **int type 의 Array** 를 **point(가리키는)**하는 주소값을 담고 있는 변수 **p(4bytes)**

```
typedef struct Record{
        int index;
        char number[10];
        char name[30];
        char *next;
}G_Record;    // 48bytes == 0x30bytes
```

다. **G_Record *p;** **G_Reocrd type 의 Array** 를 **point(가리키는)**하는 주소값을 담고 있는 변수 **p(4bytes)**

4)주의점
int *p 혹은 char *p 혹은 G_Record *p 변수가 local Function 내에 local variable 로 선언될 때 Garbage 값이 들어간다.
Global variable 로 선언되어 있다면 0('\0' = NULL)로 자동으로 컴파일러에 의해 초기화된다.
만약 garbage 주소값이 "<메모리영역을 벗어난 곳>을 가리키거나 <시스템영역을 가리킨>상태에서 point 하는 곳의 값들을 고치려고" 하거나 "메모리 영역을 벗어난 곳을 읽으려 할 때" 시스템다운 문제가 발생한다.



1.1.pointer 종류

1.1.1. *와 **

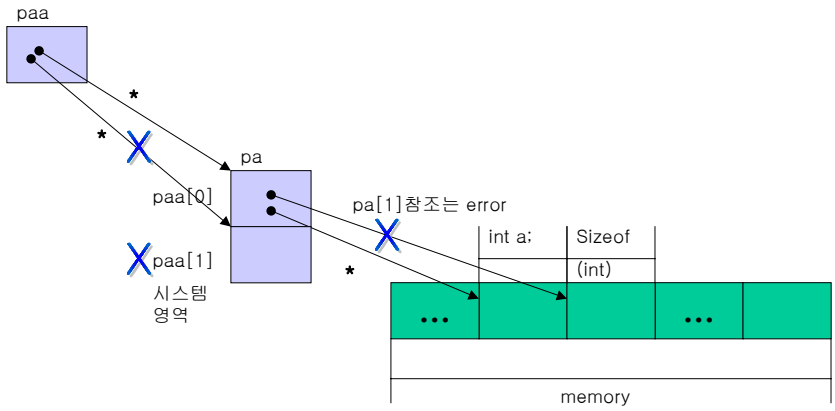
```
int a, *pa **paa;
      pa=&a;
      ppa=&pa;
```

=>위의 예처럼 사용했다면 사용시 주의할 점:
가. 포인터는 1 차원 array 를 가리키는 주소값을 담은 변수다.
따라서 포인터에 넘긴 주소를 가진 변수(int a)가 4 바이트 변수값이므로
사용할 때 paa[0][0]==**paa==pa[0] == a; 만 참조 해야한다.
a 의 주소를 벗어나는 참조를 해서는 안된다.

바른 사용	틀린 사용(문법상 맞지만 pa=&a;로 초기화 했으므로 a 외의 array 를 참조 해서는 안된다.) =>**시스템 영역을 건드리거나 메모리를 벗어난 영역을** 건드리려고 하면 다운의 원인이 된다.

paa[0][0]==**paa==pa[0] == a ; <실제값>
paa[0] == *paa == pa == &a; <주소값>
	paa[x][y]; //<x ,y: [0,n] n:0 이상의 정수이고 x=y!=0>
	paa[x]; // paa[1]……paa[x];
	pa[x]; //pa[1], ,,,,, pa[x];

=> 그림



1.1.2.포인터와 배열

1)변수

```
char *p,i;
p=&i;
```

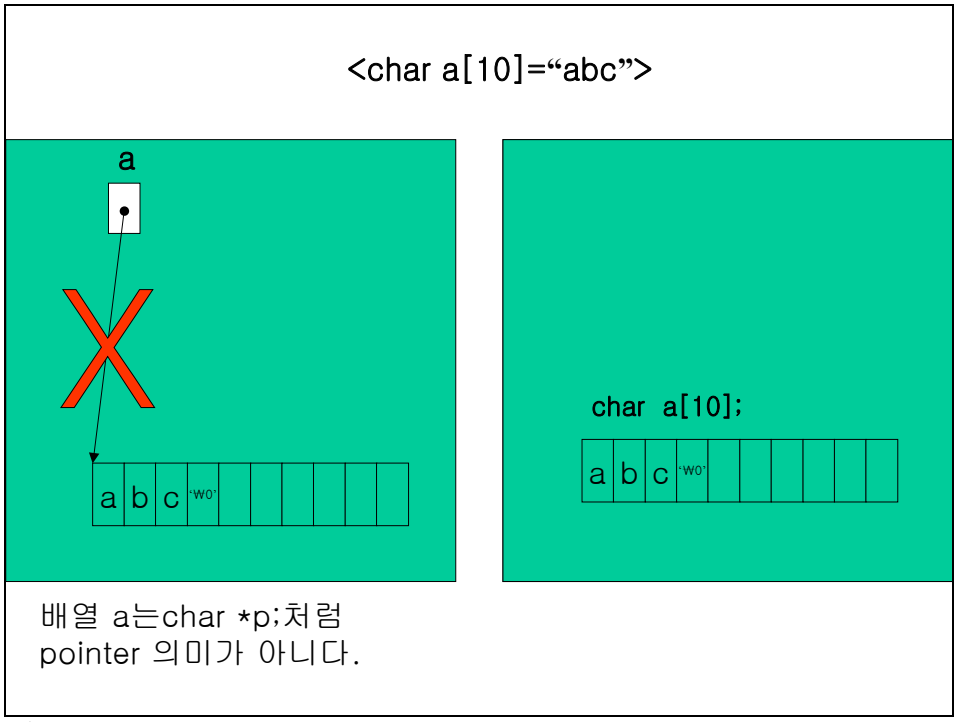
2)포인터와 1 차배열

```
char *p,a[i];
p = &a[0]; // or = a;
```

=>결과

i	<실제값>	<주소값>
0	*p ==p[0] ==a[0];	p==&a[0] ==a;
1	*(p+1)==p[1] ==a[1]	&P[1]== &a[1]
…		
i	*(p+I)==p[i]==a[i]	&p[I]== &a[I];

<참고> char a[10];는 문자열 array 이다. 그러나 a 는 &a[0]을 나타내는 주소값이다.
그러나 **a 가 포인터 변수는** 아니다.



(참고)

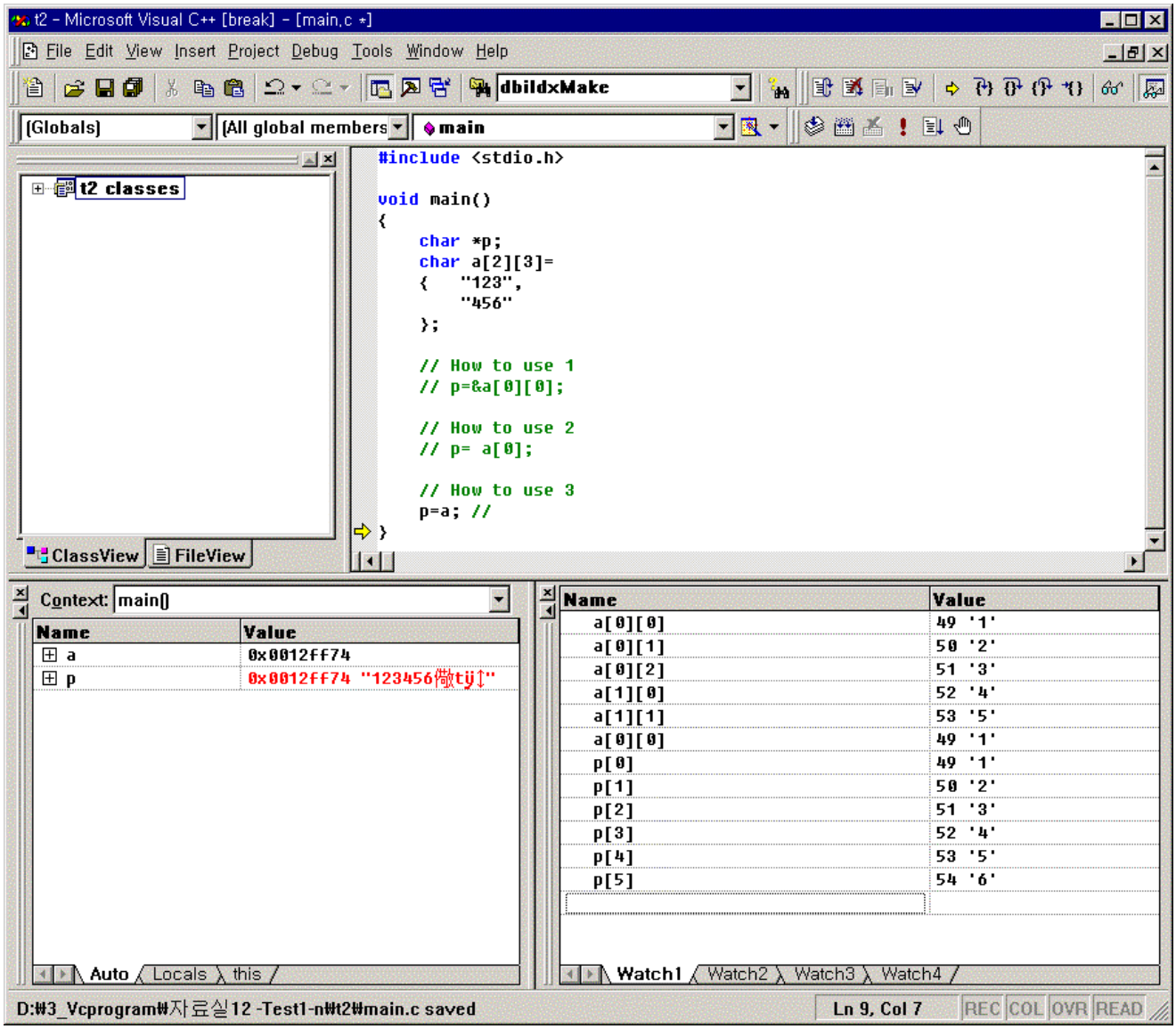
가. (p+I) == p[I]==p 가 가리키고 있는 주소를 sizeof(char) * I 만큼 이동한곳이 가리키는 실제값
=> p 는 char *이므로 이동주소는 sizeof(char) *I 이다.

나. *p+I == (*p)+1==p[0]+1== p[0]의 값에 +1 을 더하는 것

3)포인터와 2 차배열(2 차배열 -> 1 차배열로 표현)

```
char *p,a[i][j];
p=&a[0][0]; // = a[0] or =a ;
```

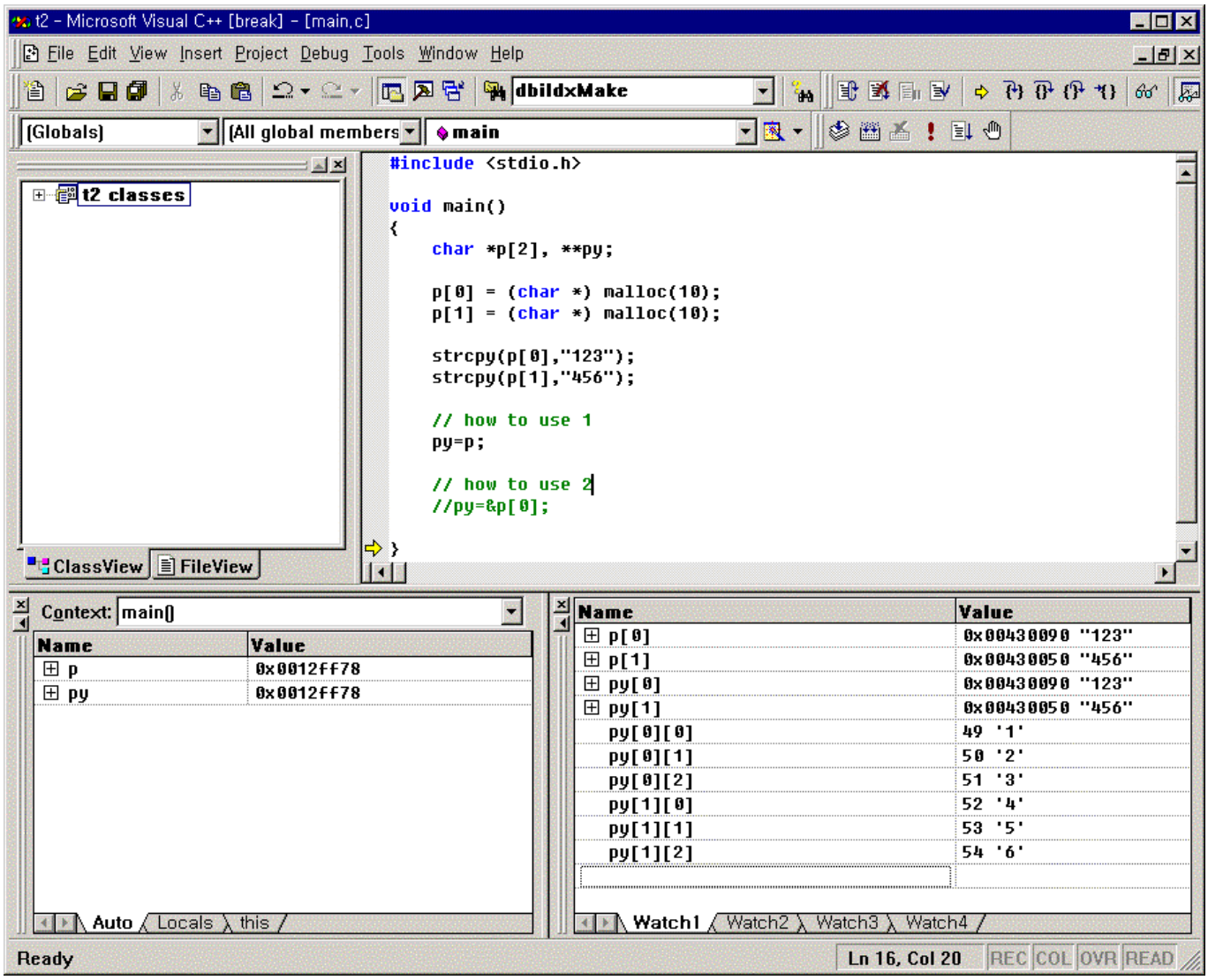
I	<실제값>	<주소값>
0	*p ==p[0]== a[0][0]	
1	*(p+1)== P[1] == a[0][1]	
~		
i	*(p+i) == p[i] ==a[0][i]	

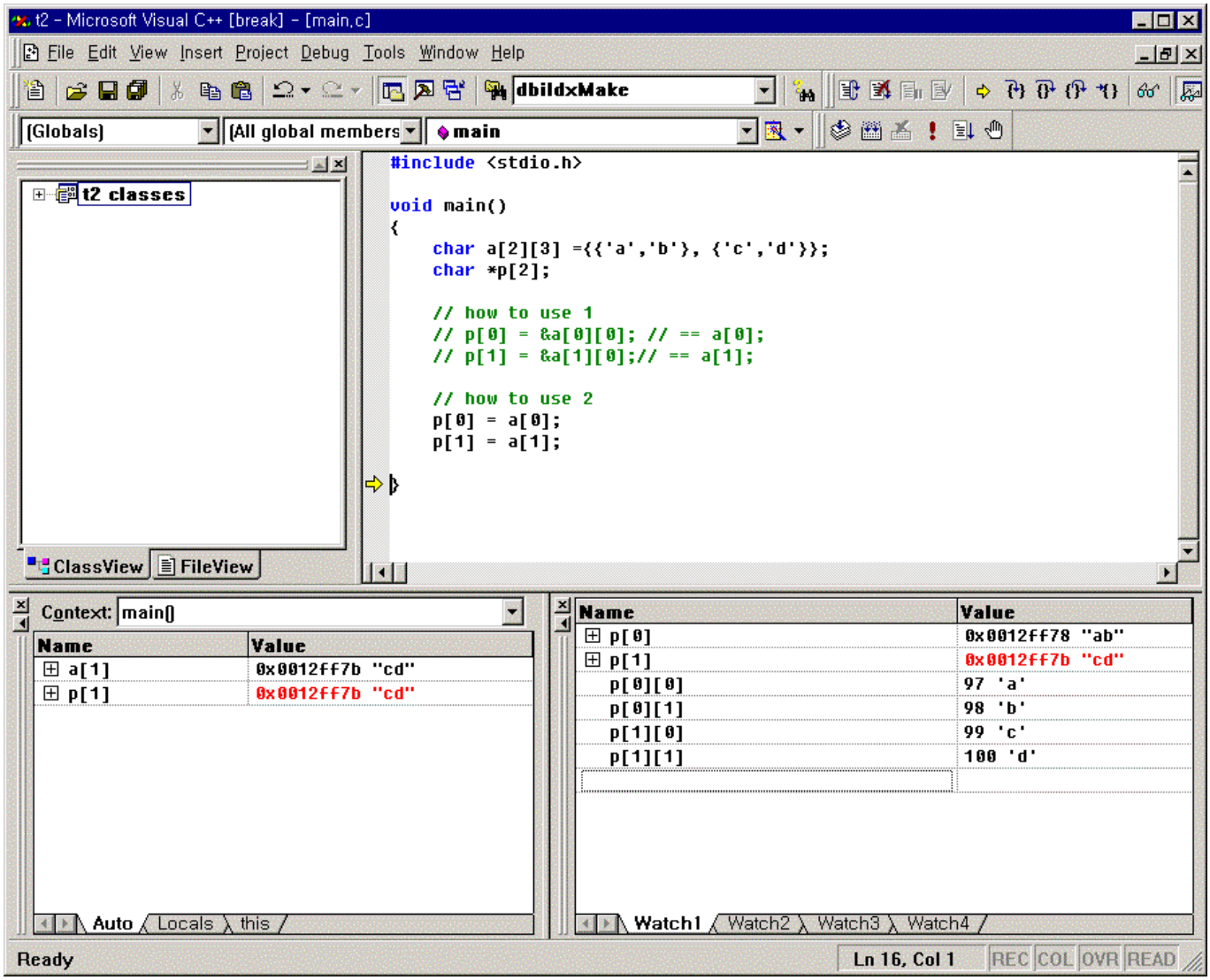


4)배열 포인터와 포인터 배열

4-1) 포인터 배열(*p[i])

char *p[i], **py;
 	
가.<실제값> **py=*p[0];
나.<주소값> *py=p[0];
<주소값> py=&p[0];
=p;
	char a[2][3] =
	{{'a','b'},
	{'c','d'}};
	char *p[2];
	p[0] = &a[0][0]; // == a[0];
	p[1] = &a[1][0];// == a[1];





4-2)배열포인터((*py)[i])

int y[][2]={ {1,2}, {3,4}}; int (*py)[2]; py=y; (why? *py==py[]) =&y[0]; =y[0] = &y[0][0]; =>suspicious pointer conversion	char a[2][3] = {{'a','b'}, {'c','d'}}; char (*p)[3]; p = a; // == &a[0];

1.1.3.배열, 포인터가 초기화(data 주어질 때) 됐을 때

=> 서로 혼용하여 사용

1)p[i][j]=" "; or **py;

주소값		실제값	
배열	포인터	배열	포인터
&p[i][j]	*(p+i)+j	p[i][j]	*(*(p+i)+j)
p[i], &p[i]	*(p+i)		

2)p[i] or *py;

주소값		실제값	
배열	포인터	배열	포인터
&p[i] p=&p[0]	p+I	p[i]	*(p+i)

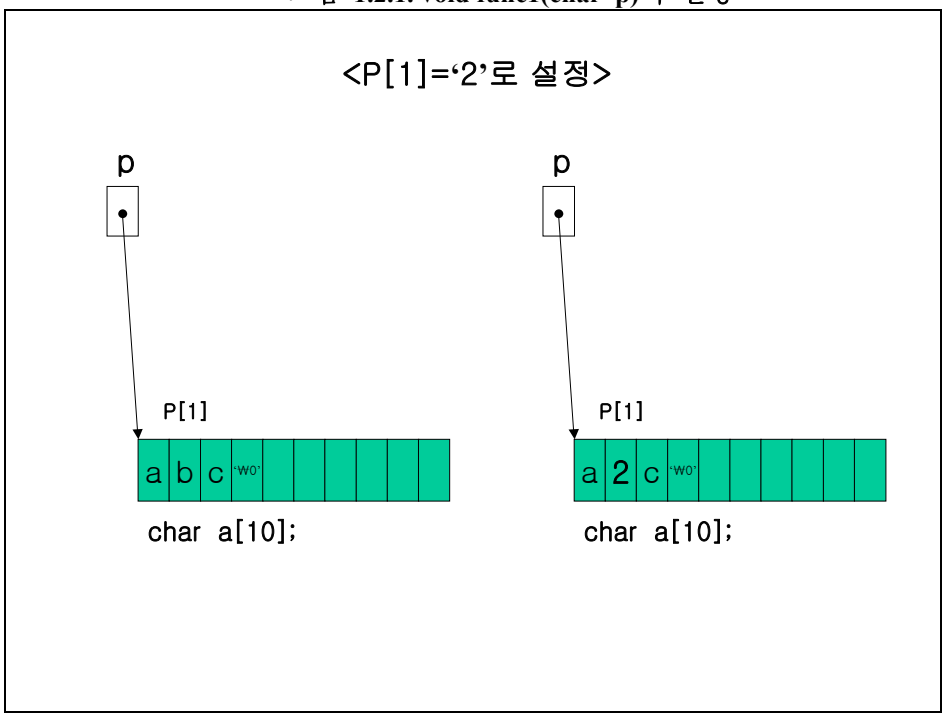
1.2. Call by reference, call by value, call by address

1.2.1. 동일한 의미의 call by reference

- 원래 배열과 포인터는 엄연히 다르지만 **Function 의 parameter 로 사용될때는** 배열이 포인터처럼 적용된다.

char a[10]= " abc";
void func1(char *p)
{
p[1]= '2';
 	
}
=> func1(a); 했을 때 a[1]의 내용 변화
	char a[10]="abc";
	void func2(char p[10])
	{
	p[1] ='1';
	}
	
	=> func2(a); 했을 때 a[1]의 내용 변화

<그림 1.2.1. void func1(char*p)의 설명>

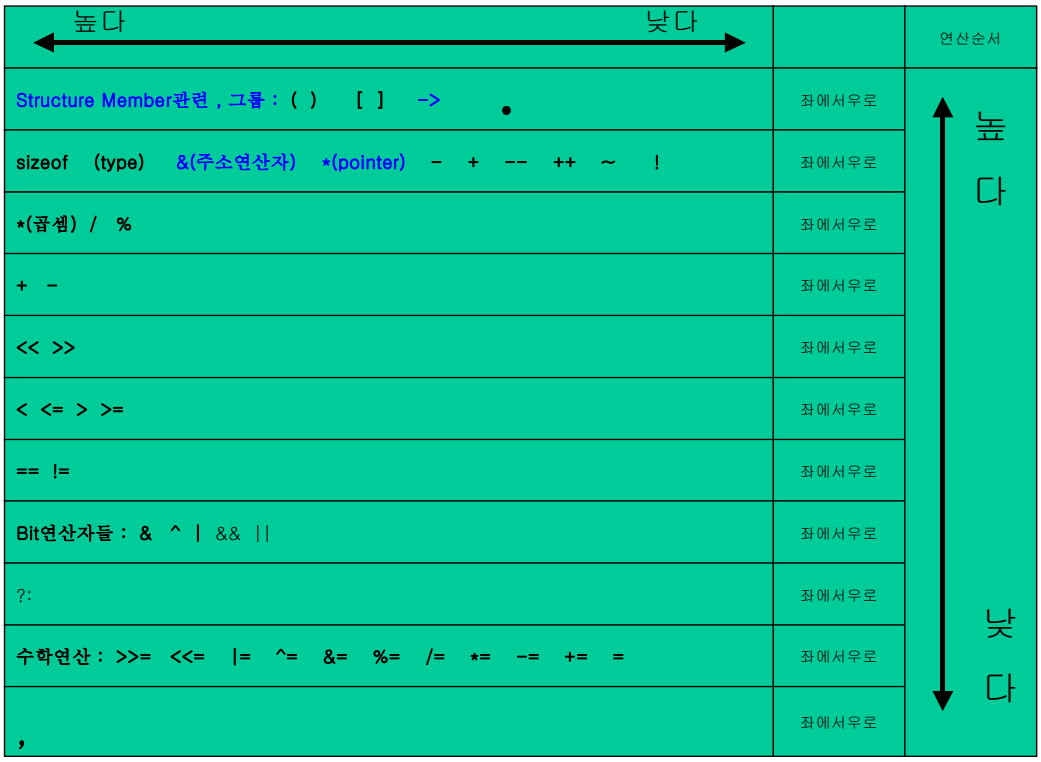


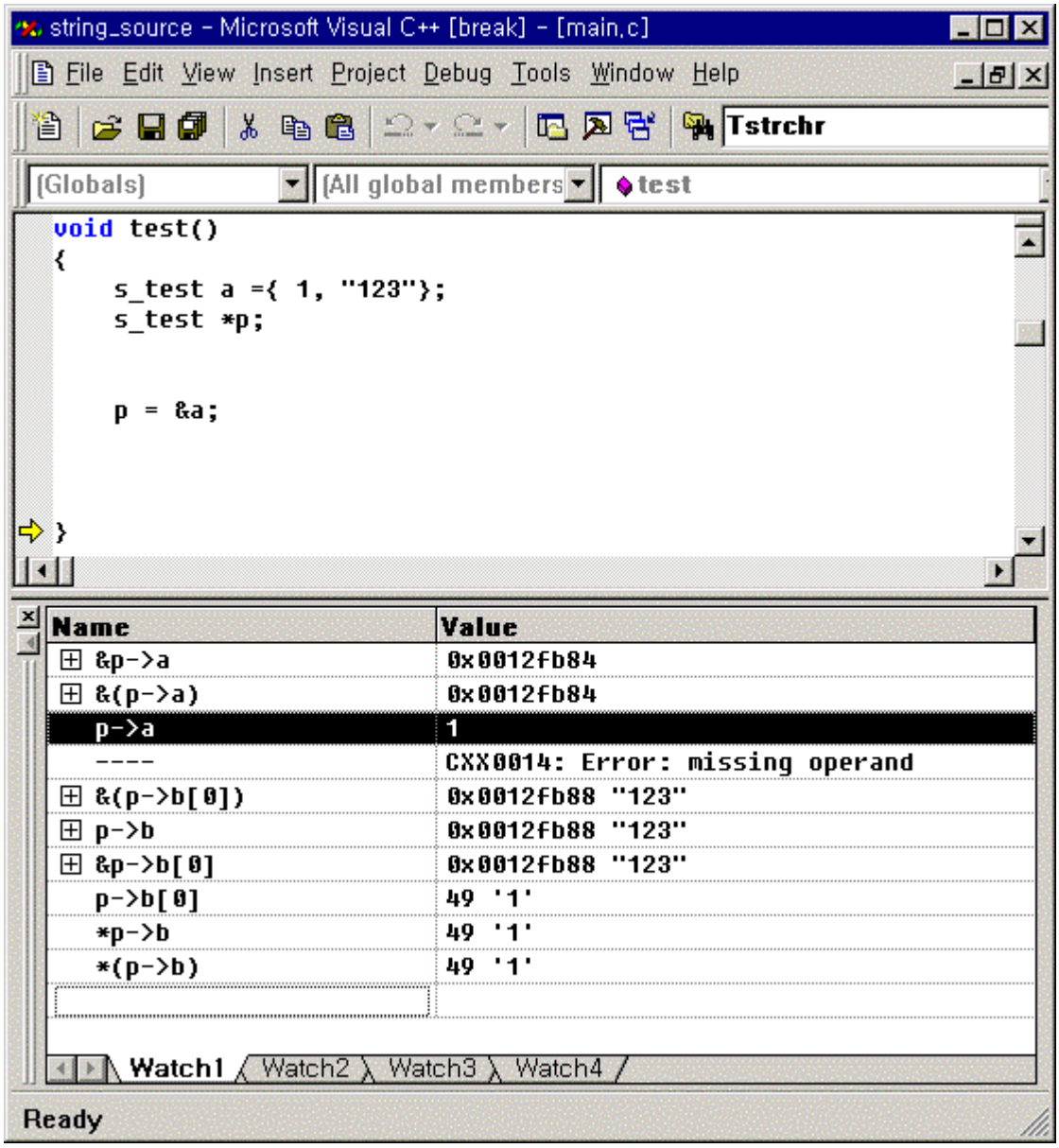
1.2.2. Loacl Variable Pointer Return(X)

=> 함수내의 Local 변수의 pointer 를 return 해서는 안된다.
=> malloc 은 Kernel alloc 이므로 상관이없다.

```
Char * caution_return()
{
    char temp[500] = “1234567890”;

    return temp; // return 하고 나서 temp[500]의 공간은 사라지고 다른 변수에게 할당 될수 있다
}
```

1.3.연산자우선순위(신규추가)




string_source - Microsoft Visual C++ [break] - [main.c]
File Edit View Insert Project Debug Tools Window Help
Tstrchr
(Globals)
(All global members
test
void test()
{
s_test a ={ 1, "123"};
s_test *p;
p = &a;
}
Name
Value
&p->a
0x0012fb84
&(p->a)
0x0012fb84
p->a
1

CXX0014: Error: missing operand
&(p->b[0])
0x0012fb88 "123"
p->b
0x0012fb88 "123"
&p->b[0]
0x0012fb88 "123"
p->b[0]
49 '1'
*p->b
49 '1'
*(p->b)
49 '1'
Watch1
Watch2
Watch3
Watch4
Ready

1.4. ASCII Chart

“” == ‘’=NULL == 0x00

주의 점) “”와 ‘’

“”: 문자열 NULL

‘’:문자의 NULL

“ “ == Space == ‘ ‘ == 0x20

1.4.1.ASCII Character Codes Chart 1

Ctrl	Dec	Hex	Char	Code	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
^@	0	00		NUL	32	20	sp	64	40	@	96	60	`
^A	1	01	☺	SOH	33	21	!	65	41	A	97	61	a
^B	2	02	☻	STX	34	22	"	66	42	B	98	62	b
^C	3	03	♥	ETX	35	23	#	67	43	C	99	63	c
^D	4	04	♦	EOT	36	24	$	68	44	D	100	64	d
^E	5	05	♣	ENQ	37	25	%	69	45	E	101	65	e
^F	6	06	♠	ACK	38	26	&	70	46	F	102	66	f
^G	7	07	•	BEL	39	27	'	71	47	G	103	67	g
^H	8	08	◘	BS	40	28	(	72	48	H	104	68	h
^I	9	09	○	HT	41	29	)	73	49	I	105	69	i
^J	10	0A	◙	LF	42	2A	*	74	4A	J	106	6A	j
^K	11	0B	♂	VT	43	2B	+	75	4B	K	107	6B	k
^L	12	0C	♀	FF	44	2C	,	76	4C	L	108	6C	l
^M	13	0D	♪	CR	45	2D	-	77	4D	M	109	6D	m
^N	14	0E	♫	SO	46	2E	.	78	4E	N	110	6E	n
^O	15	0F	☼	SI	47	2F	/	79	4F	O	111	6F	o
^P	16	10	►	DLE	48	30	0	80	50	P	112	70	p
^Q	17	11	◄	DC1	49	31	1	81	51	Q	113	71	q
^R	18	12	↕	DC2	50	32	2	82	52	R	114	72	r
^S	19	13	‼	DC3	51	33	3	83	53	S	115	73	s
^T	20	14	¶	DC4	52	34	4	84	54	T	116	74	t
^U	21	15	§	NAK	53	35	5	85	55	U	117	75	u
^V	22	16	▬	SYN	54	36	6	86	56	V	118	76	v
^W	23	17	↨	ETB	55	37	7	87	57	W	119	77	w
^X	24	18	↑	CAN	56	38	8	88	58	X	120	78	x
^Y	25	19	↓	EM	57	39	9	89	59	Y	121	79	y
^Z	26	1A	→	SUB	58	3A	:	90	5A	Z	122	7A	z
^[	27	1B	←	ESC	59	3B	;	91	5B	[	123	7B	{
^\	28	1C	∟	FS	60	3C	<	92	5C	\	124	7C	|
^]	29	1D	↔	GS	61	3D	=	93	5D	]	125	7D	}
^^	30	1E	▲	RS	62	3E	>	94	5E	^	126	7E	~
^_	31	1F	▼	US	63	3F	?	95	5F	_	127	7F	⌂†

† ASCII code 127 has the code DEL. Under MS-DOS, this code has the same effect as ASCII 8 (BS). The DEL code can be generated by the CTRL+BKSP key.

1.4.2.ASCII Character Codes Chart 1

Dec	Hex	Char
128	80	Ç
129	81	ü
130	82	é
131	83	â
132	84	ä
133	85	à
134	86	å
135	87	ç
136	88	ê
137	89	ë
138	8A	è
139	8B	ï
140	8C	î
141	8D	ì
142	8E	Ä
143	8F	Å
144	90	É
145	91	æ
146	92	Æ
147	93	ô
148	94	ö
149	95	ò
150	96	û
151	97	ù
152	98	ÿ
153	99	Ö
154	9A	Ü
155	9B	¢
156	9C	£
157	9D	¥
158	9E	₧
159	9F	ƒ

Dec	Hex	Char
160	A0	á
161	A1	í
162	A2	ó
163	A3	ú
164	A4	ñ
165	A5	Ñ
166	A6	ª
167	A7	º
168	A8	¿
169	A9	⌐
170	AA	¬
171	AB	½
172	AC	¼
173	AD	¡
174	AE	«
175	AF	»
176	B0	▒
177	B1	▒
178	B2	▒
179	B3	│
180	B4	┤
181	B5	╡
182	B6	╢
183	B7	╖
184	B8	╕
185	B9	╣
186	BA	║
187	BB	╗
188	BC	╝
189	BD	╜
190	BE	╛
191	BF	┐

Dec	Hex	Char
192	C0	└
193	C1	┴
194	C2	┬
195	C3	├
196	C4	─
197	C5	┼
198	C6	╞
199	C7	╟
200	C8	╚
201	C9	╔
202	CA	╩
203	CB	╦
204	CC	╠
205	CD	═
206	CE	╬
207	CF	╧
208	D0	╨
209	D1	╤
210	D2	╥
211	D3	╙
212	D4	╘
213	D5	╒
214	D6	╓
215	D7	╫
216	D8	╪
217	D9	┘
218	DA	┌
219	DB	▒
220	DC	▒
221	DD	▒
222	DE	▒
223	DF	▒

Dec	Hex	Char
224	E0	α
225	E1	ß
226	E2	Γ
227	E3	π
228	E4	Σ
229	E5	σ
230	E6	µ
231	E7	τ
232	E8	Φ
233	E9	Θ
234	EA	Ω
235	EB	δ
236	EC	∞
237	ED	ø
238	EE	∈
239	EF	∩
240	F0	≡
241	F1	±
242	F2	≥
243	F3	≤
244	F4	⌠
245	F5	⌡
246	F6	÷
247	F7	≈
248	F8	°
249	F9	∙
250	FA	·
251	FB	√
252	FC	n
253	FD	²
254	FE	■
255	FF	

1.5. Window,Linux(or Unix)저장방식차이(big endian, Little endian)

►**Little Endian :** LSB(최하위비트:0 번째 비트)에서 MSB 순서로 저장되어 있다. (하위비트-->상위비트)

►**Big Endian** : MSB(최상위비트)부터 LSB 로 저장되어 있다. (상위비트-->하위비트)

Windows	**Big Endian[MSB ~ LSB]**	**Little Endian [LSB ~ MSB]**
char	**O**	X
Int, long	X	O

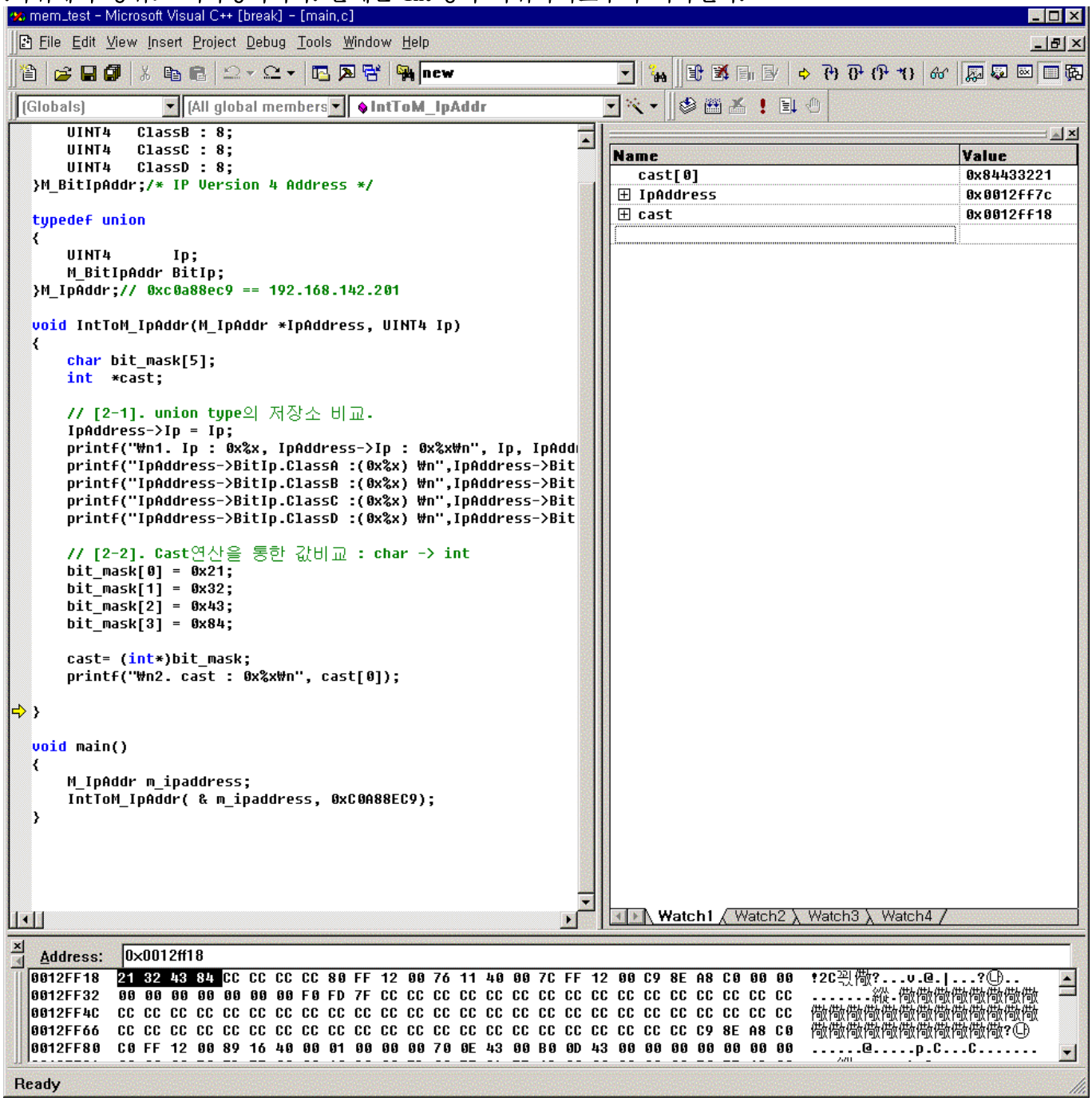
Linux or Unix	**Big Endian[MSB ~ LSB]**	**Little Endian [LSB ~ MSB]**
char	**O**	X
Int, long	**O**	X

```
#include <stdio.h>
#define UINT4 unsigned int
typedef struct
{
	UINT4	ClassA : 8;
	UINT4	ClassB : 8;
	UINT4	ClassC : 8;
	UINT4	ClassD : 8;
}M_BitIpAddr;/* IP Version 4 Address */

typedef union
{
	UINT4		Ip;
	M_BitIpAddr BitIp;
}M_IpAddr;// 0xc0a88ec9 == 192.168.142.201

void IntToM_IpAddr(M_IpAddr *IpAddress, UINT4 Ip)
{
	char bit_mask[5];
	int	*cast;


	// [2-1]. union type 의 저장소 비교.
	IpAddress->Ip = Ip;
	printf("\n1. Ip : 0x%x, IpAddress->Ip : 0x%x\n", Ip, IpAddress->Ip);
	printf("IpAddress->BitIp.ClassA :(0x%x) \n",IpAddress->BitIp.ClassA);
	printf("IpAddress->BitIp.ClassB :(0x%x) \n",IpAddress->BitIp.ClassB);
	printf("IpAddress->BitIp.ClassC :(0x%x) \n",IpAddress->BitIp.ClassC);
	printf("IpAddress->BitIp.ClassD :(0x%x) \n",IpAddress->BitIp.ClassD);

	// [2-2]. Cast 연산을 통한 값비교 : char -> int
	bit_mask[0] = 0x21;
	bit_mask[1] = 0x32;
	bit_mask[2] = 0x43;
	bit_mask[3] = 0x84;

	cast= (int*)bit_mask;
	printf("\n2. cast : 0x%x\n", cast[0]);
}

void main()
```

```
{
    M_IpAddr m_ipaddress;
    IntToM_IpAddr( & m_ipaddress, 0xC0A88EC9);

}
```

1.5.1.윈도우 경우 메모리 상태(char 나 Int 형이 다른 저장방식)

:하위에서 상위로 기록방식이다. 문제는 Int 형이 하위바이트부터 기록된다.



=> 실행결과.

1. Ip : 0xc0a88ec9, IpAddress->Ip : 0xc0a88ec9

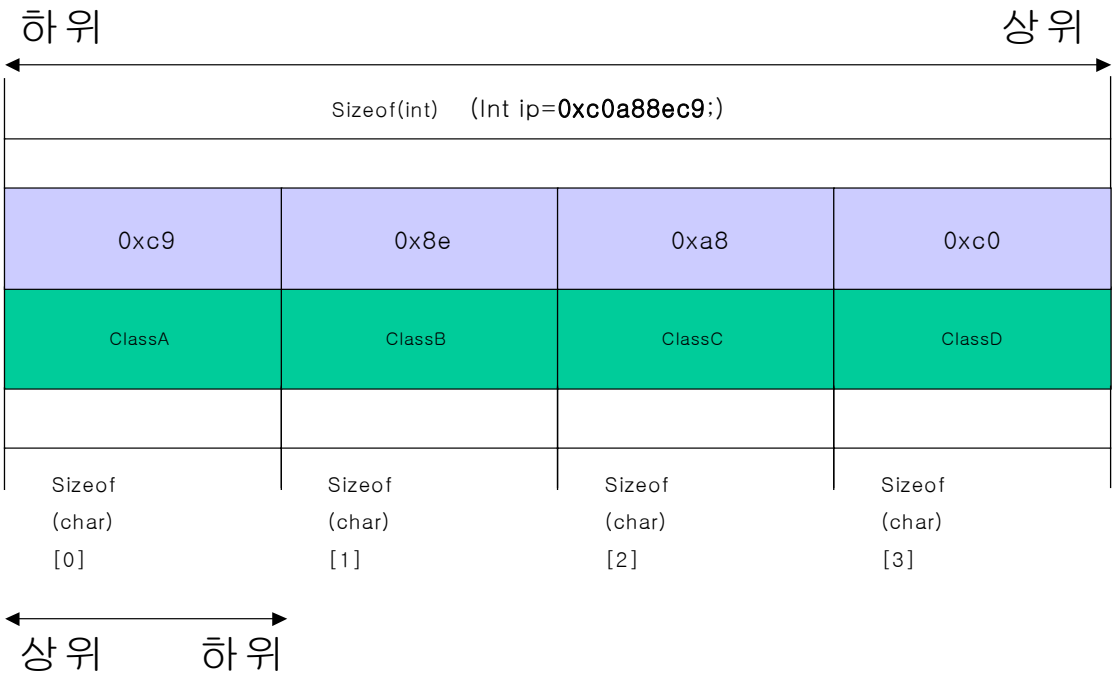
IpAddress->BitIp.ClassA :(0xc9)

IpAddress->BitIp.ClassB :(0x8e)

IpAddress->BitIp.ClassC :(0xa8)

IpAddress->BitIp.ClassD :(0xc0)

2. cast : 0x84433221



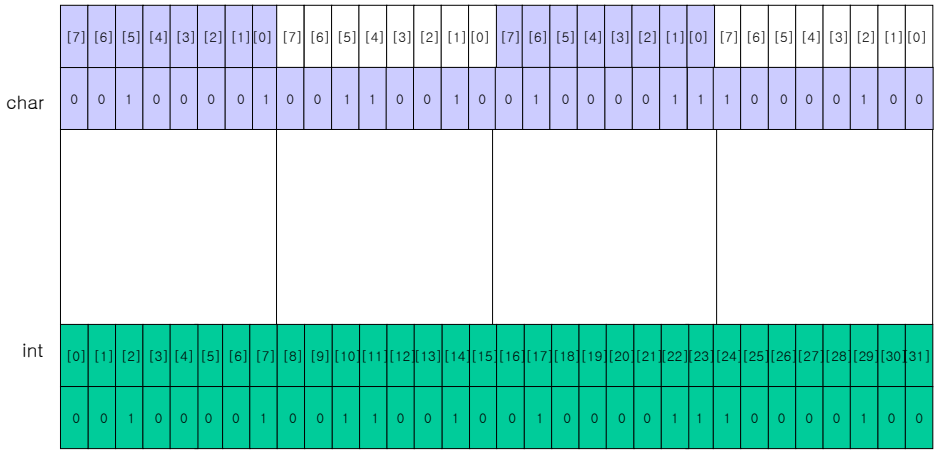
Window의 모든 데이타저장방식

(cast연산이나 Offset방식 read/write 유의)

<그림 1.5.1.가. M_IpAddr test;의 메모리구조>

[윈도우의 int, char형의 저장 방식]

```
Char bit_mask[5];   int *cast;
bit_mask[0] = 0x21;    bit_mask[1] = 0x32;
bit_mask[2] = 0x43;    bit_mask[3] = 0x84;
cast= (int*)bit_mask;
```



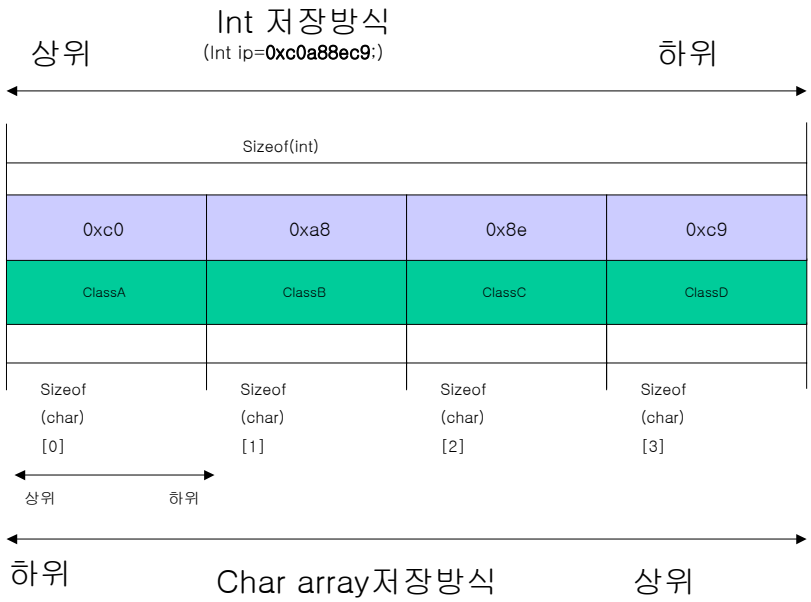
<그림 1.5.1.나. char -> int 형으로 cast 한 결과**>**

1.5.2.Unix(Linux) –동일저장 char 내 bit(상위->하위), int 내 bit(상위->하위)

메모리에 기록될 것을 먼저 기록하는 방식(산수기록방식)

=>결과

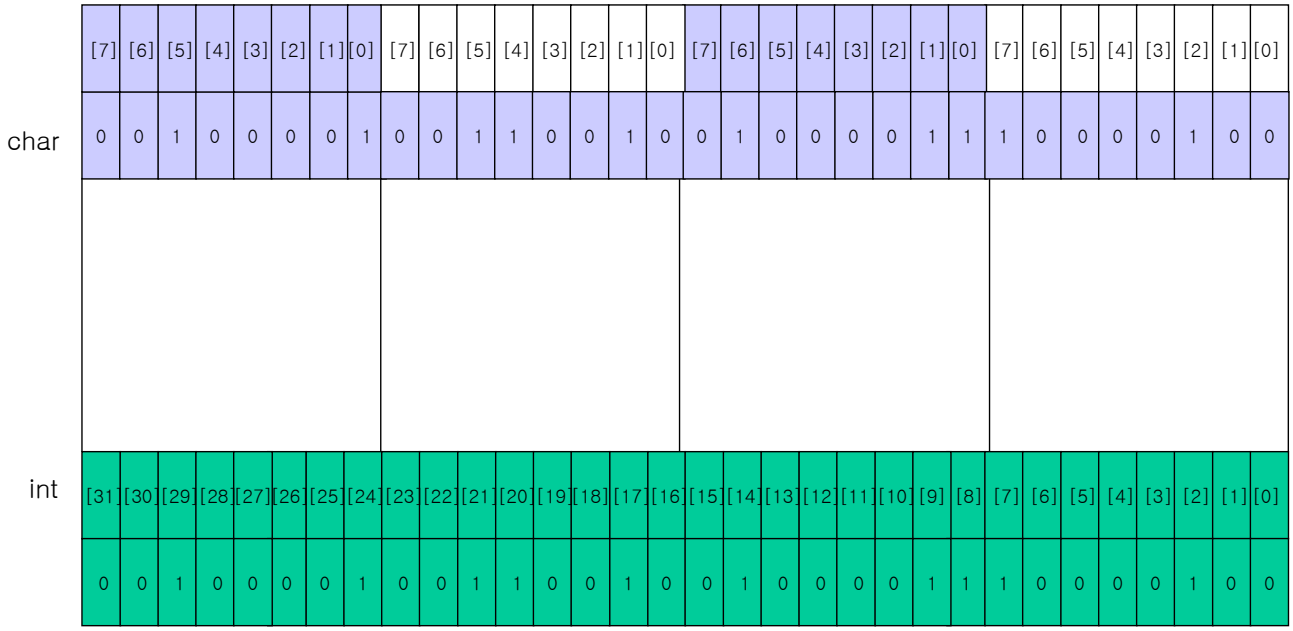
```
fplmts[scwpark: /home/user/scwpark/test/test 123 ] a.out
1. Ip : 0xc0a88ec9, IpAddress->Ip : 0xc0a88ec9
IpAddress->BitIp.ClassA :(0xc0)
IpAddress->BitIp.ClassB :(0xa8)
IpAddress->BitIp.ClassC :(0x8e)
IpAddress->BitIp.ClassD :(0xc9)

2. cast : 0x21324384
fplmts[scwpark: /home/user/scwpark/test/test 124 ]
```



<1.5.2.가. M_IpAddr test;의 메모리구조>

[Linux, Unix의 int, char형의 저장 방식]

```
Char bit_mask[5];   int *cast;
bit_mask[0] = 0x21;    bit_mask[1] = 0x32;
bit_mask[2] = 0x43;    bit_mask[3] = 0x84;
cast= (int*)bit_mask;
```



<1.5.2.나. char -> int 형으로 cast 연산>

1.5.3.C 언어 메모리연산 함수

int 형 데이터에서 c 언어의 Shift 연산의 의미는 Windows 나 Linux(Unix)나 동일한 의미를 나타낸다.
마치 Linux(Unix)메모리모양 방식인 것처럼 Windows Program 이 실행된다.
그러나 windows program 에서 실제 저장되는 방식은 int 타입에서 차이가 난다.

1.5.3.1. << : 곱셈

1.5.3.2. >> : 나눗셈

1.5.3.3. int 내에서 특정 bit 번째 1 이나 0 으로 세팅하는 함수.

1.6. 포인터 수식연산, 포인터간의 수식연산(OFFSET)

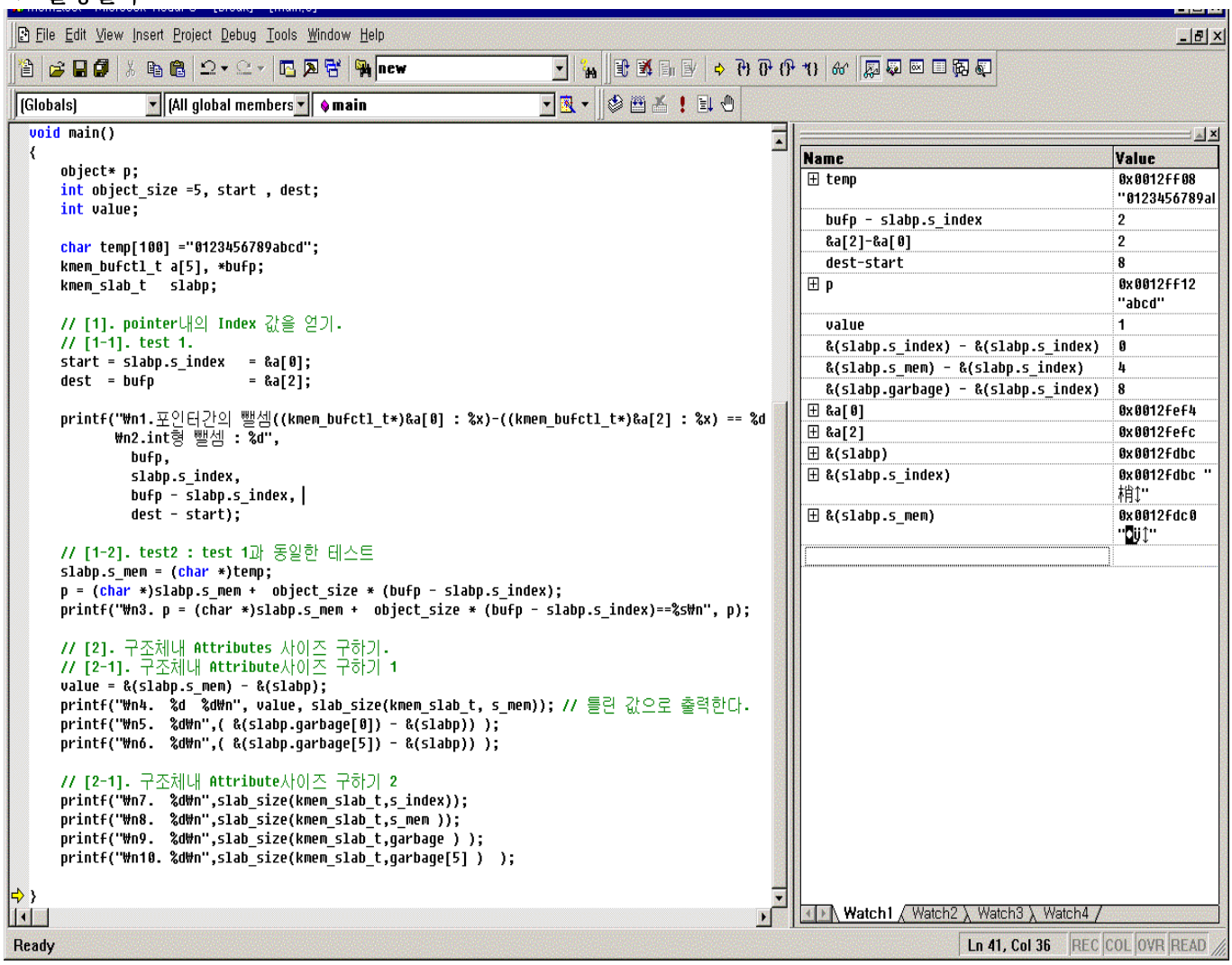
1.6.1. 예제 1. Linux Kernel 2.3.0 분석소스

1.6.1.1.소스

```
typedef struct kmem_slab_s
{
	struct kmem_bufctl_s	*s_index;
	void			*s_mem;	/* addr of first obj in slab */
	char			garbage[300];

} kmem_slab_t;

typedef struct kmem_bufctl_s
{
	union
	{
		kmem_slab_t	*buf_slabp;	/* slab for obj */
		void		*buf_objp;
	} u;
} kmem_bufctl_t;

#define object	char

// 구조체내의 Attribute 의 상대적 크기 구하기.
// 동일한 방식== (int) (& ((type*)0)->member)
#define slab_size(type, member)	(int) (& ((type*)0)->member)

void main()
{
	object* p;
	int object_size =5, start , dest;
	int value;

	char temp[100] ="0123456789abcd";
	kmem_bufctl_t a[5], *bufp;
	kmem_slab_t	slabp;

	// [1]. pointer 내의 Index 값을 얻기.
	// [1-1]. test 1.
	start = slabp.s_index	= &a[0];
	dest	= bufp		= &a[2];

	printf("\n1.포인터간의 뺄셈((kmem_bufctl_t*)&a[0]:%x)-((kmem_bufctl_t*)&a[2] : %x) == %d\n2.int 형 뺄셈 : %d",
			bufp,
			slabp.s_index,
			bufp - slabp.s_index,
			dest - start);

	// [1-2]. test2 : test 1 과 동일한 테스트
	slabp.s_mem = (char *)temp;
	p = (char *)slabp.s_mem +	object_size * (bufp - slabp.s_index);
	printf("\n3. p = (char *)slabp.s_mem +	object_size * (bufp - slabp.s_index)==%s\n", p);

	// [2]. 구조체내 Attributes 사이즈 구하기.
	// [2-1]. 구조체내 Attribute 사이즈 구하기 1
	value = &(slabp.s_mem) - &(slabp);
	printf("\n4.	%d	%d\n", value, slab_size(kmem_slab_t, s_mem)); // 틀린 값으로 출력한다.
```

```
    printf("\n5.   %d\n",( &(slabp.garbage[0]) - &(slabp)) );
    printf("\n6.   %d\n",( &(slabp.garbage[5]) - &(slabp)) );

    // [2-1]. 구조체내 Attribute 사이즈 구하기 2
    printf("\n7.   %d\n",slab_size(kmem_slab_t,s_index));
    printf("\n8.   %d\n",slab_size(kmem_slab_t,s_mem ));
    printf("\n9.   %d\n",slab_size(kmem_slab_t,garbage ) );
    printf("\n10. %d\n",slab_size(kmem_slab_t,garbage[5] )    );

}
```

=> 실행결과



1.포인터간의 뺄셈((kmem_bufctl_t*)&a[0] : 12fefc)-((kmem_bufctl_t*)&a[2] : 12fef4) == 2

2.int 형 뺄셈 : 8

3. p = (char *)slabp.s_mem + object_size * (bufp - slabp.s_index)==abcde

4. 1 4

5. 8

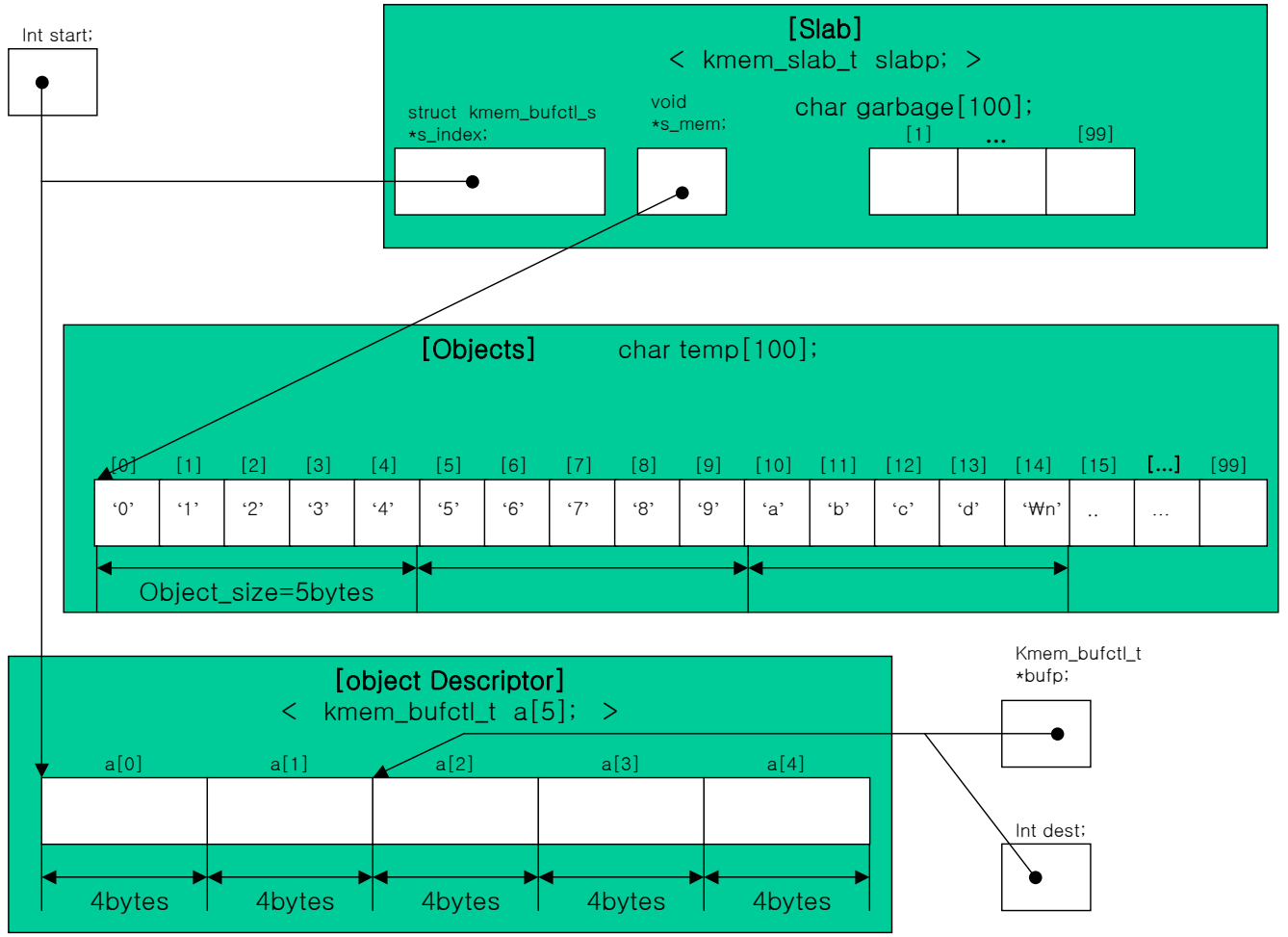
6. 13

7. 0

8. 4

9. 8 10. 13

1.6.1.2.분석



1.6.1.2.1. 동일 구조체 Array 에서 Pointer 의 수식연산 (+/- n)

수식 대상 : kmem_bufctl_t a[5];

Kmem_bufctl_t *od = s_index;	
od +=2; == &a[2] == &s_index[2]	**bufp =bufp - 2**; == &a[0] == s_index;

1.6.1.2.2. 동일 구조체 Array 에서 pointer 들간의 연산

수식 대상 : kmem_bufctl_t a[5];

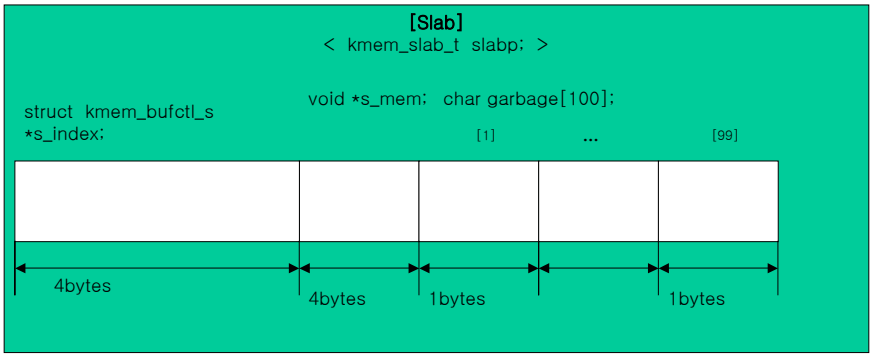
포인터간의 연산 결과	포인터값을 int 형에 담아서 뺄셈한 결과 = 실제 bytes 값
&a[2] - &a[0] == bufp – slabp.s_index == 2 == 실제 크기 8bytes/sizeof(kmem_bufctl_t)	dest – start == 2 * sizeof(kmem_bufctl_t)== 8

1.6.1.2.3.구조체내의 Attribute 들의 상대적 Offset 구하기.

[kmem_slab_t구조체에서 Attribute의 상대적 offset구하기]

//동일한 방식== (int) (& ((type*)0)->member)

#define slab_size(type, member) (& ((type*)0)->member)




```
void main()
{
	…….
	// [2-1]. 구조체내 Attribute 사이즈 구하기 2
	printf("\n7.   %d\n",slab_size(kmem_slab_t,s_index));
	printf("\n8.   %d\n",slab_size(kmem_slab_t,s_mem ));
	printf("\n9.   %d\n",slab_size(kmem_slab_t,garbage ) );
	printf("\n10. %d\n",slab_size(kmem_slab_t,garbage[5] )    );
}
```

=>실행결과

7. 0

8. 4

9. 8

10. 13

1.6.2. 예제 2. 구조체내 attribute 의 offset 구하기

메모리 Alignment 구하기.

1.6.2.1.소스

```
#include <stdio.h>

typedef struct list
{
	int data;
	int sex;
	int face;
} Link;

// the offset of member in structure type
#define list_entry(ptr , type, member )\
( &((type *)0)->member) // the offset of member in structure type

int main()
{
	Link a;

	printf("\n array num of pointer: %x , real offset %x \n",
		&a.sex - &a.data , list_entry(0,Link,sex));
	return 0;
}
```

=> 실행결과

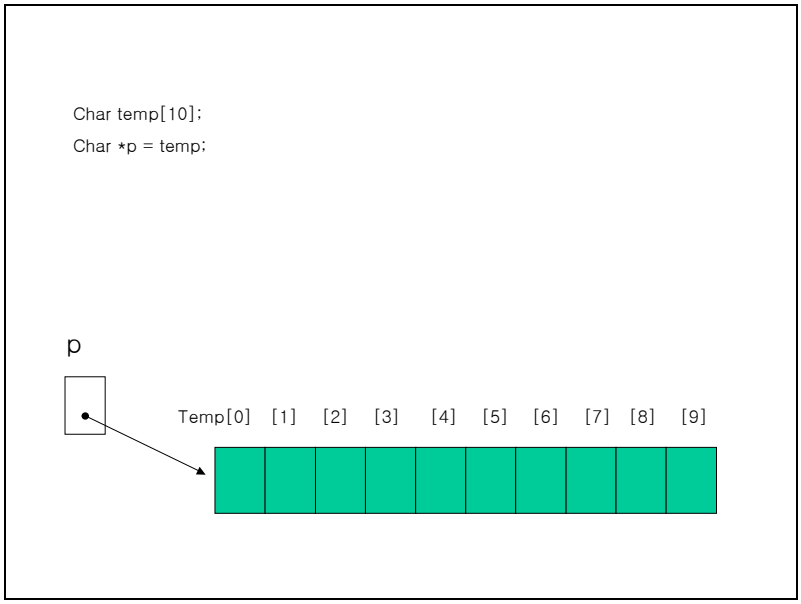
```
array num of pointer: 1 , real offset 4
```

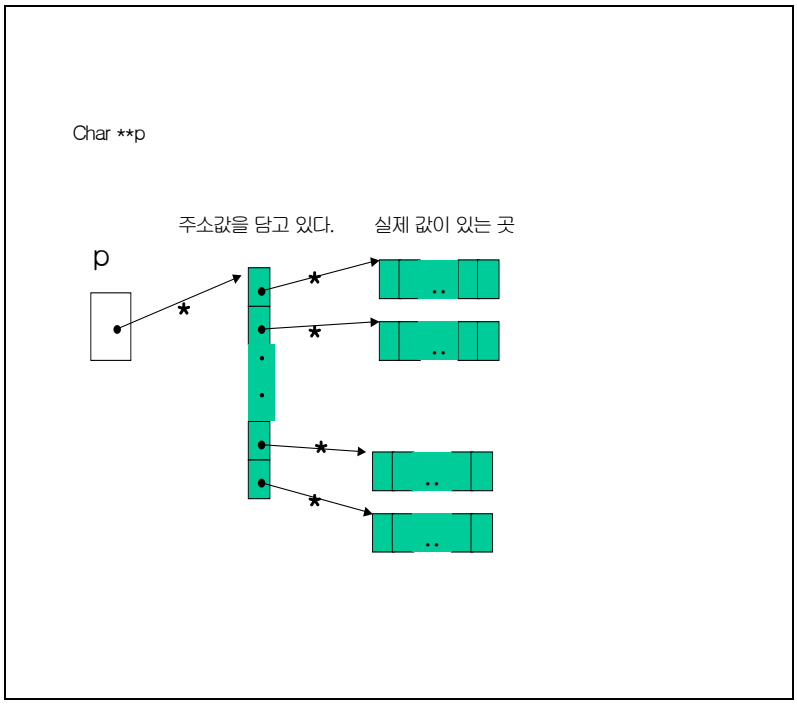
1.6.2.2.분석

2. 예제로 보는 pointer

2.1.pointer 기능

1. 1 차원 array 를 가리킨다.
무조건 1 차원 Array 를 가리킨다.
따라서 포인터에 특정 번지를 넘기면 모두 array 로 인식한다.
결과적으로 예상치 못한 영역도 가리킬 위험성이 있다.
포인터가 가리키는 영역의 범위는 프로그래머가 틀리지 않는 범위내에서 가리키도록 프로그램을 만들어야 한다.





2. pointer 용도 : call by reference 를 응용하여 작업을한다.
Call by reference 는 주소값을 이용하여 실제 데이터들에 접근한다.

1)string 관련 기능
- source data 를 destiny 에 복사 : ex) string copy, memory copy
- string search
- string compare

2) memory swap
3) function pointer
4) memory (graphic, game 에서 sprite 복사등에 응용)
- allocate
- copy
- move

2.2. 소스예제 1 :

2.2.1. class pointer DATA. 구조

Data	기능
public : char a[10];	10 바이트 문자열
public: char **m_p2Array;	m_pArray[5]의 주소를 넘겨받는 이중 pointer
public: char *m_pArray[5];	Array pointer

2.2.2.pointer source

```
#include <string.h> // strcpy
#include <malloc.h> // malloc
#include <iostream.h>

void(*pFunc1)(char *p);

class Pointer{

public :
    // 1. 1 차원 배열
    char a[10];

    // 2. 1 차원 포인터 배열과 2 차원 pointer
    char **m_p2Array;
    char *m_pArray[5];

public:

    //void(*pFunc)(char *p);
    void FuncPoint();
    void PointPoint();
    ~Pointer();
    Pointer(char p[10]);
    void func1(char*p);
    void func2(char p[10]);


};

// [1]. Creator
Pointer::Pointer(char p[10])
{
    // 1.
    StringCpy (a,p);

    // 2. 1 차원 포인터 배열 초기화
    for(int i =0; i<5; i++)
    {
        m_pArray[i]= (char *)malloc(10);
    }

    strcpy(m_pArray[0],"abcde");
    strcpy(m_pArray[1],"fghij");
    strcpy(m_pArray[2],"klmno");
    strcpy(m_pArray[3],"pqrst");
    strcpy(m_pArray[4],"uvwxy");
```

```
}

// [2]. Destructor
Pointer::~Pointer()
{
	for(int i =0; i<5; i++)
	{
		free (m_pArray[i]);

	}

}

// [3]
// [3-1]
void Pointer::func1(char *p)
{
	cout << "before func1 execute.\n" ;
	p[1] = '1';
	cout << "after pointer value : " << a << "\n";

}

void Pointer::func2(char p[10])
{
	cout << "before func2 execute.\n" ;
	p[1] = '2';
	cout << "after pointer value : " << a << "\n";

}

// [3-2].
/********
(1) 기능 : source 를  dest 로 string copy
********/
void StringCpy(char *dest, char *source)
{
	// 방법 1. 포인터를 이용한 방법
	while((*dest = *source) != '\0')
	{
		dest ++;
		source++;
	}

	// 방법 2. 배열을 이용한 방법.
	/**********************************
	int i;

	i =0;
	while((dest[i] = source[i]) != '\0')
	{
		i++;
	}
	**********************************/
}
```

```
/********
1) 기능 : 2 문자를 서로 바꿈.
*********/
void Swapping(char *a, char *b)
{
	char temp ;
	temp	= *b;
	*b	= *a;
	*a	= temp;
}

// [3-3]. allocate memory
typedef struct Record{
	int index;
	char number[10];
	char name[30];
	char *next;
}G_Record;	// 48bytes == 0x30bytes

G_Record *MemoryAlloc1(void)
{
	G_Record * source ;
	source = (G_Record *)malloc(sizeof(G_Record));
	return source;
}


G_Record *MemoryAlloc2(G_Record **source)
{
	*source = (G_Record *)malloc(sizeof(G_Record));
	return *source;
}


// [4].
void Pointer::PointPoint()
{
	// 2 가지 방법
	// m_p2Array = m_pArray;
	m_p2Array = &m_pArray[0];

	for(int i =0; i< 5; i++)
		cout << m_p2Array[i] << "\n";

}

// [5-1]. 클래스에서 함수 포인터 사용은 안됨.
void Pointer::FuncPoint()
{
	// [1]. class member 간 함수 포인터 안됨
	// pFunc = func1;

	// [2].class member 를 c 함수 포인터로 못넘김.
	// pFunc1 = func1;
}

// [5-2]. C 상태에서 함수 포인터 사용 가능
```

```
int (*m_pFunc)(char *p);

int test(char *p)
{

	cout << "function pointer test :" << p <<"\n";

	return 0;

}
#define macro_func test


void main()
{
	G_Record **m_pphead = NULL;
	G_Record *head = NULL;
	char *temp;
	char name[30];
	int index =0;

	Pointer *Test = new Pointer("Test");

	// [1]. pointer 의 2 가지 기능 1) 주소값만 넘겨주기. 2) 공간만큼 할당하여 복사.
		// [1-1].call by refrence
		Test->func2(Test->a);
		Test->func1(Test->a);

		// [1-2]. strcpy, swap, 메모리 할당방법
		// [1-2-1]. stringcopy
		char string1[10] = "string1";
		char string2[10];
		StringCpy(string2, string1);

		cout<< string1 << "    "<< string2 << "\n";

		// [1-2-2]. swap
		char s1='1';
		char s2='2';
		Swapping(&s1,&s2);
		cout<< s1 << s2<< "\n";

		// [1-2-3]. 메모리 할당방법
		// 1. 메모리 할당법 1
		head = MemoryAlloc1();
		strcpy(head->name,"2pst");
		strcpy(head->number,"2");
		cout << head->name << head->number << "\n";
		free(head);

		// 2. 메모리 할당법 2
		MemoryAlloc2(&head);
		strcpy(head->name,"1pst");
		strcpy(head->number,"1");
		cout << head->name << head->number << "\n";
		free(head);

		// 3. 메모리 할당법 3
		// 3-1. G_Record 를 10 개정도 할당하기.
```

```
        head = (G_Record *)malloc(sizeof(G_Record)* 10);

        for( index =0; index <10; index++)
              head[index].index = index;

        free(head);

        // 3-2. 문자열  640*400+1 bytes 를  할당하는  방법
        temp =(char*) malloc(640*400+1);

        StringCpy(temp,"hi, hello son");

        free(temp);


// [2]. 1-dimension pointer array, 2 차원  포인터  변수.
// [2-1].
Test->PointPoint();

// [2-2]. G_Record 를  10 개정도  할당하기.
head = (G_Record *)malloc(sizeof(G_Record)* 10);
m_pphead = &head;

for( index =0; index <10; index++)
{
        head[index].index = index;
        cout<< "m_pphead" << index, m_pphead[0][index].index;

        name[0] = '0'+index;
        name[1] = NULL;
        StringCpy(head[index].name,name);

}
cout<<"end";

free(head);

//[2-3].2 dimension pointer array
m_pphead = (G_Record **)malloc(sizeof(G_Record*)*10);

for( index =0; index <10; index++)
{
        m_pphead[index] = (G_Record *)malloc(sizeof(G_Record));
        m_pphead[index]->index = index;

        name[0] = '0'+index;
        name[1] = NULL;
        StringCpy(m_pphead[index]->name,name);

}

for( index =0; index <10; index++)
        free(m_pphead[index]);
free(m_pphead);

// [3-1]. 포인터  함수
m_pFunc = test;
m_pFunc(Test->a);

//[3-2]. macro   함수
```

```
    macro_func(Test->a);
    delete Test;
}
```

2.3.생성자- Pointer::Pointer(char p[10])분석

1.기능:
1) 입력받은 문자열(10 바이트내)을 char a[10]에 복사
단)strcpy 함수는 10 바이트 모두를 복사하는 것이 아니라 NULL('\0')을 만날때까지의 문자만 복사한다.
따라서 source 문자열이 "abc"라면 3 바이트만 복사되고 NULL 이 문자열 끝에 삽입된다.
2) char *m_pArray[5];를 10 바이트 array 로 메모리 할당.
m_pArray[0] 에는 "abcde"로 값을 넣는다.
m _pArray[1]에는 "fghij"로 값을 넣는다.
m_pArray[2]에는 "klmno"로 값을 넣는다.
m_pArray[3]에는 "pqrst"로 값을 넣는다.
m_pArray[4]에는 "uvwxy"로 값을 넣는다.

2.필요한 데이터 :

3.필요한 함수

4. input

5. output

6. algorithm

7. 참고
Pointer::Pointer(char p[10])로 함수정의 하는 것은 Pointer::Pointer(char *p)와 동일하다.
Char p[10]에서 p 는 포인터처럼 행동한다.
단 **이런 동작은 char p[10]과 같은 변수가 함수 내에서 parameter(argument)로 사용될 때만 그런다.**
본질적으로 array 와 pointer 는 다르다.
Pointer : 1 차원 array 를 가리키는 포인터(주소값: memory address reference).
Array :1 차원 배열.

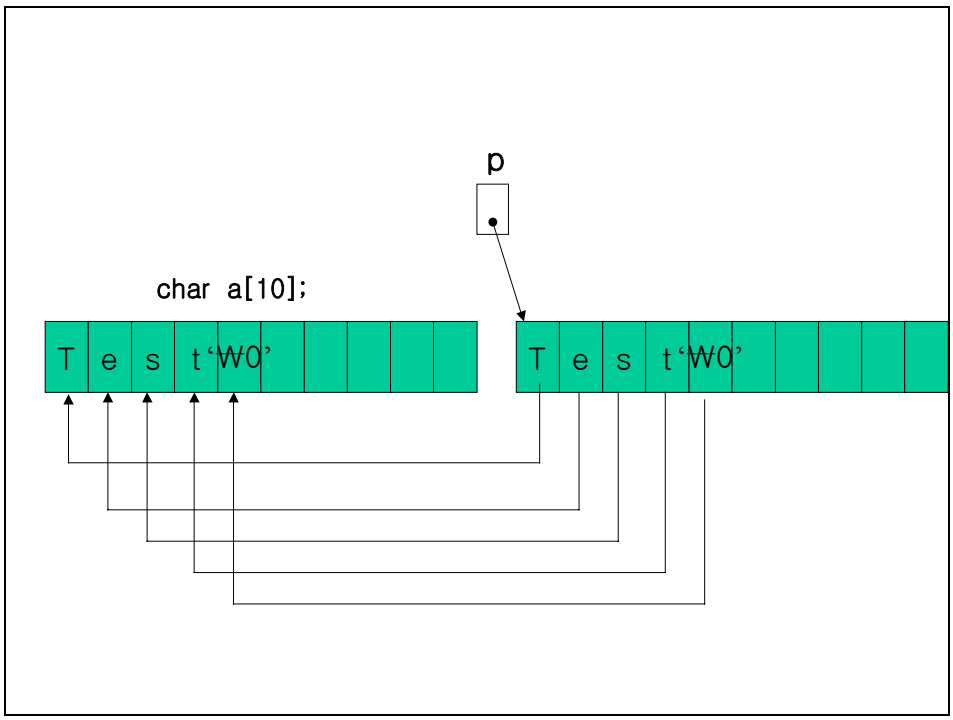
```
void main()
{

    Pointer *Test = new Pointer("Test"); //생성자 Pointer::Pointer(char p[10]); 자동 호출된다.

        ~~~; //  이러쿵 저러쿵 사용
        ~~~;// 이러쿵 저러쿵 사용중

}
```

```
Pointer::Pointer(char p[10])
{
    // 1.
    StringCpy (a,p);

    // 2. 1 차원 포인터 배열 초기화
    for(int i =0; i<5; i++)
    {
        m_pArray[i]= (char *)malloc(10);
    }
```

```
    strcpy(m_pArray[0],"abcde");
    strcpy(m_pArray[1],"fghij");
    strcpy(m_pArray[2],"klmno");
    strcpy(m_pArray[3],"pqrst");
    strcpy(m_pArray[4],"uvwxy");
}
```

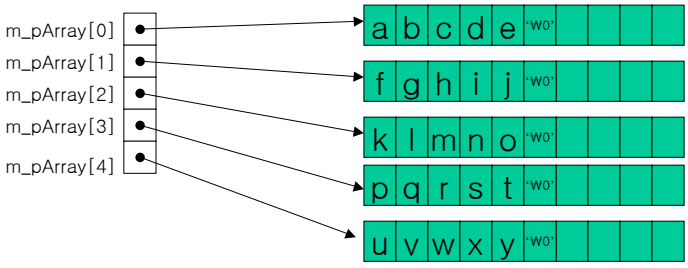
2.3.1. StringCpy (a,p);strcpy(a,p)로 사용해도 가능하다.

=> strcpy 와 동일소스이다.



2.3.2. 1 차원 포인터 배열 초기화

```
char *m_pArray[5];
for(int i =0; i<5; i++)
    m_pArray[i]= (char *)malloc(10);

strcpy(m_pArray[0],"abcde");
strcpy(m_pArray[1],"fghij");
strcpy(m_pArray[2],"klmno");
strcpy(m_pArray[3],"pqrst");
strcpy(m_pArray[4],"uvwxy");
```



2.4.소멸자

Pointer::~Pointer()
1.기능:
2) char *m_pArray[5]; 메모리 해제.

2.필요한 데이터 :

3.필요한 함수

4. input

5. output

6. algorithm

2.5.call by reference of parameter

void Pointer::func1(char *p)과 void Pointer::func2(char p[10])는

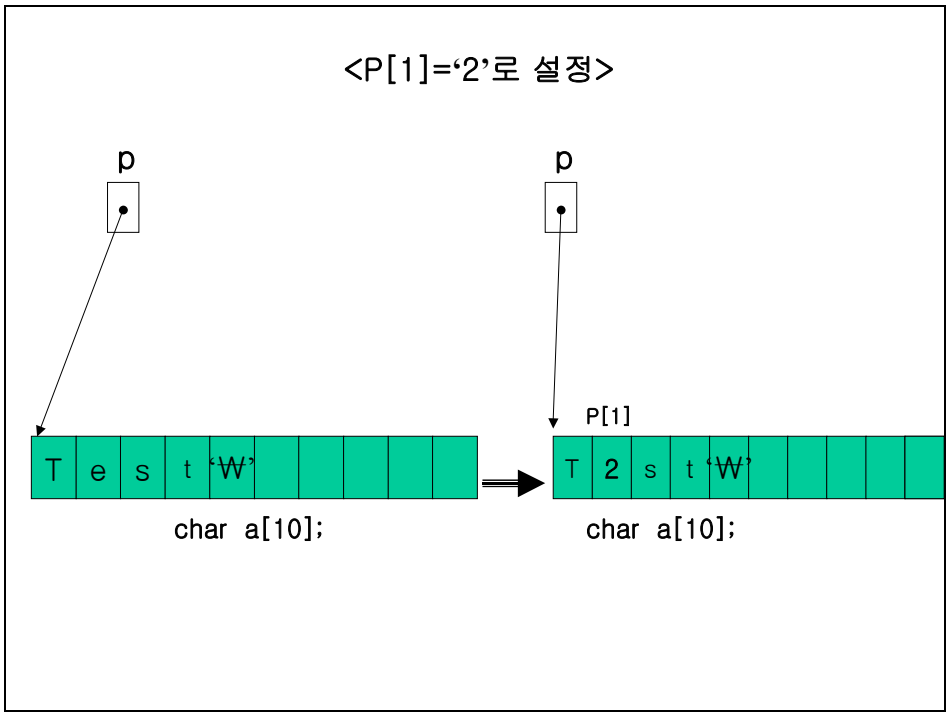
paremeter 타입이 틀리지만 **char p[10]은 마치 char *p** 처럼 포인터로 행동한다는 것을 알 수가 있다.
단 char p[10]이 함수의 parameter 로 사용될 때만 그러하다.

C 에서 함수 선언 : return_type function_name (parameters);
- return_type : int, char etc..,
- parameters : int destiny, int source etc…,

1.기능:
p 가 가리키는 문자열(parameter 로 받은 pointer)중 p[1]의 값을 '1'또는 '2'로 변경한다.
2.필요한 데이터 :
3.필요한 함수
4. input
5. output
6. algorithm

=>Source

```
Void main()
{

   ~~~~~; // 모시라고 코드 사용(중략)

   // [1-1].call by refrence
   Test->func2(Test->a);
   Test->func1(Test->a);

   ~~~~~; // 모시라고 코드 사용(중략)
}
```



2.6.string copy 와 swapping

2.6.1.StringCpy(): strcpy()함수

void StringCpy(char *dest, char *source)
1.기능:
source 가 가리키는 문자열을 dest 포인터가 가리키는 character array 에 복사.
표준 strcpy()함수와 동일 소스이다.

2.필요한 데이터 :

3.필요한 함수

4. input

5. output

6. algorithm
1) **방법 1**. 포인터 증감을 통한 복사법

2) **방법 2**: 포인터를 배열처럼 인식하여 복사법

```
/********
(1) 기능 : source 를  dest 로 string copy
********/
void StringCpy(char *dest, char *source)
{
	// 방법 1. 포인터를 이용한 방법
	while((*dest = *source) != '\0')
	{
		dest ++;
		source++;
	}

	// 방법 2. 배열을 이용한 방법.
	/**********************************
	int i;

	i =0;
	while((dest[i] = source[i]) != '\0')
	{
		i++;
	}
	**********************************/

}
```

2.6.1.1.example

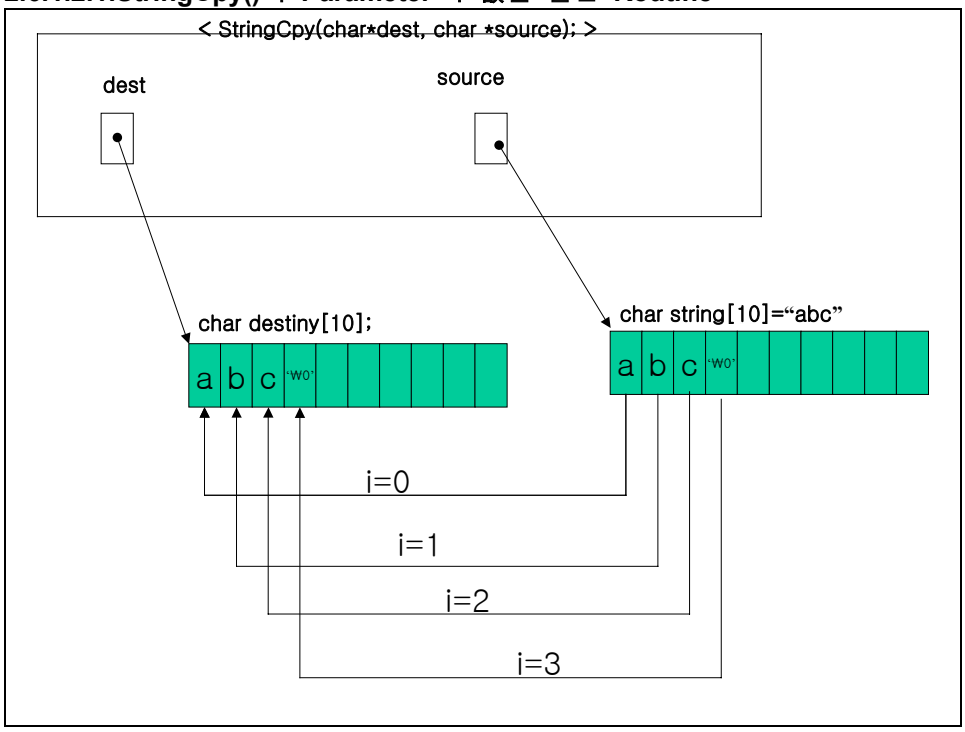
Ex)

```
char destiny[10];
char string[10] = “abc”
StringCpy(dest, string);
```

2.6.1.2.example(2.6.1.1.) trace

void StringCpy(char *dest, char *source)의 dest pointer 는 char destiny[10];의 처음 시작 포인터를 주소로 받는다. Suorce pointer 도 string[10]의 처음 시작 포인터를 주소로 받는다.

2.6.1.2.1.StringCpy()의 Parameter 가 값을 받는 Routine



2.6.1.2.2.StringCpy()실행되는 과정

1) StrCpy()내부의 변수 trace

<단 표내 약어 설명>

- dest 와 source : StrCpy()내부변수
- 회수(i)에서 i 는 방법 2. 배열을 이용한 방법에서 사용된 Local variable int i 이다>

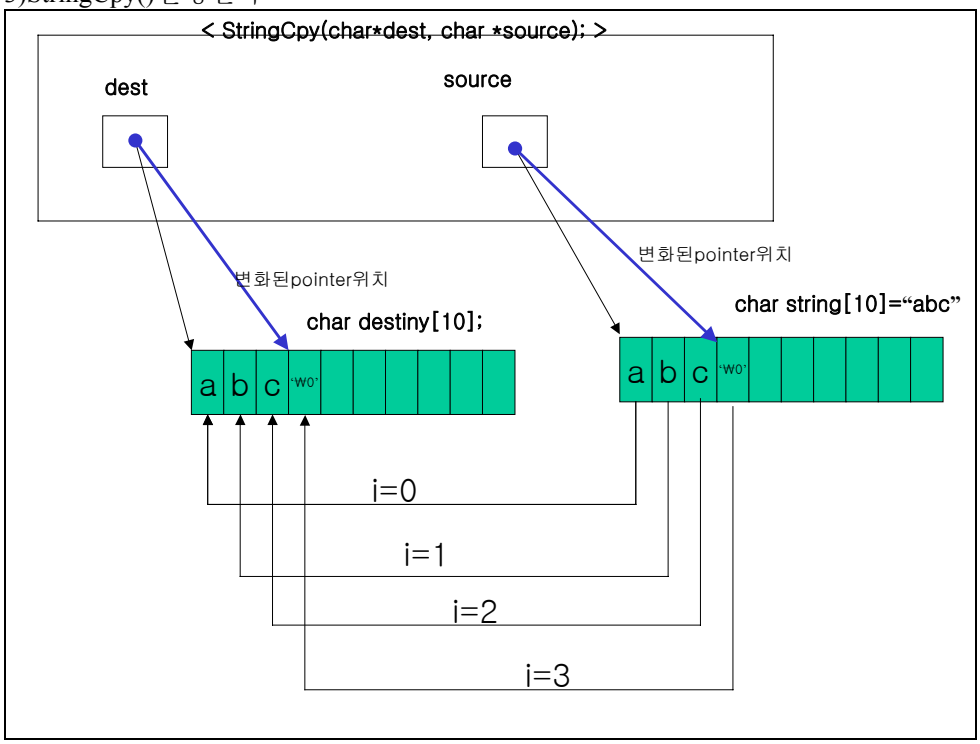
	같은 행에서 분석순서 =>				
회수 (i)	dest	Source	while((*dest = *source) != '\0')	dest ++ 실행후	source++ 실행후
1(0)	&destiny[0]	&string [0]	while((**destiny[0] = source[0]**)!=”\0”)	&destiny[1]	&string[1]
2(1)	&destiny[1]	&string [1]	while((**destiny[1]** = source[1])!=”\0”)	&destiny[2]	&string [2]
3(2)	&destiny[2]	&string [2]	while((**destiny[2]** = source[2])!=”\0”)	&destiny[3]	&string [3]
4(3)	&destiny[3]	&string [3]	while((**destiny[3]** = source[3])!=”\0”)		

2) while**((*dest = *source)** != '\0')의 확장.

	같은 행에서 분석순서 =>	
회수(i)	*source 의 변환된 값	**(*dest = *source)**
1(0)	source[0]==‘a’	**destiny** [0]=source[0]==‘a’
2(1)	source[1]==‘b’	**destiny** [1]=source[1]==‘b’
3(2)	source[2]==‘c’	**destiny** [2]=source[2]==‘c’

4(3)	source[3]==NULL==’\0’	**destiny** [3]=source[3]==NULL==’\0’

3)StringCpy()실행결과



2.6.2.swapping

void Swapping(char *a, char *b)
1.기능:
a 가 가리키는 문자 하나를 b 가 가리키는 문자 하나와 바꾸기.

2.필요한 데이터 :

3.필요한 함수

4. input

5. output

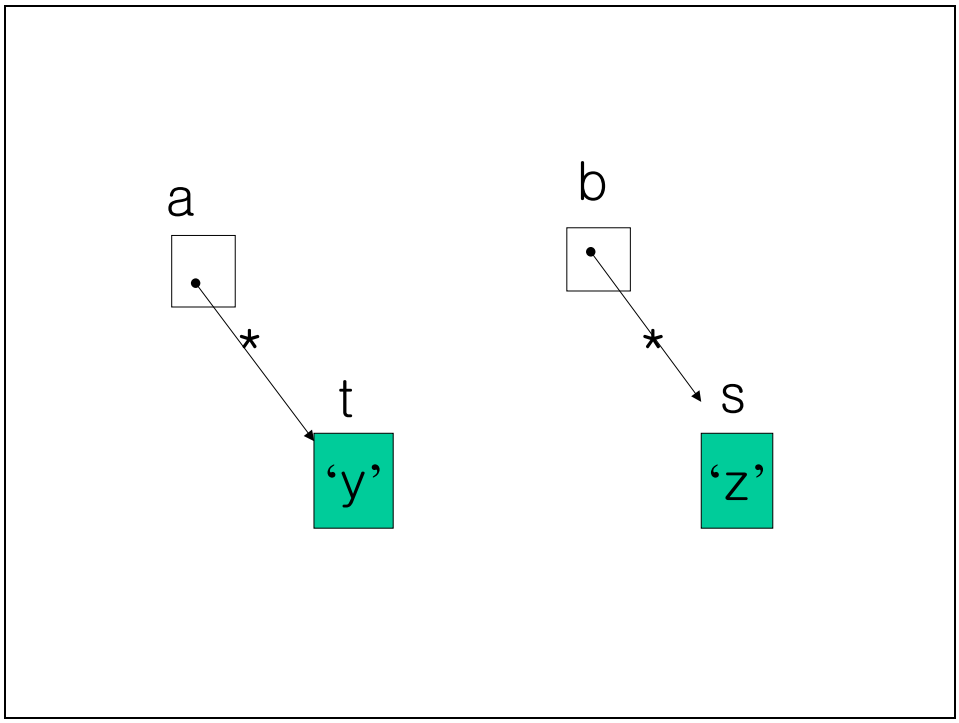
6. algorithm

7. 참고

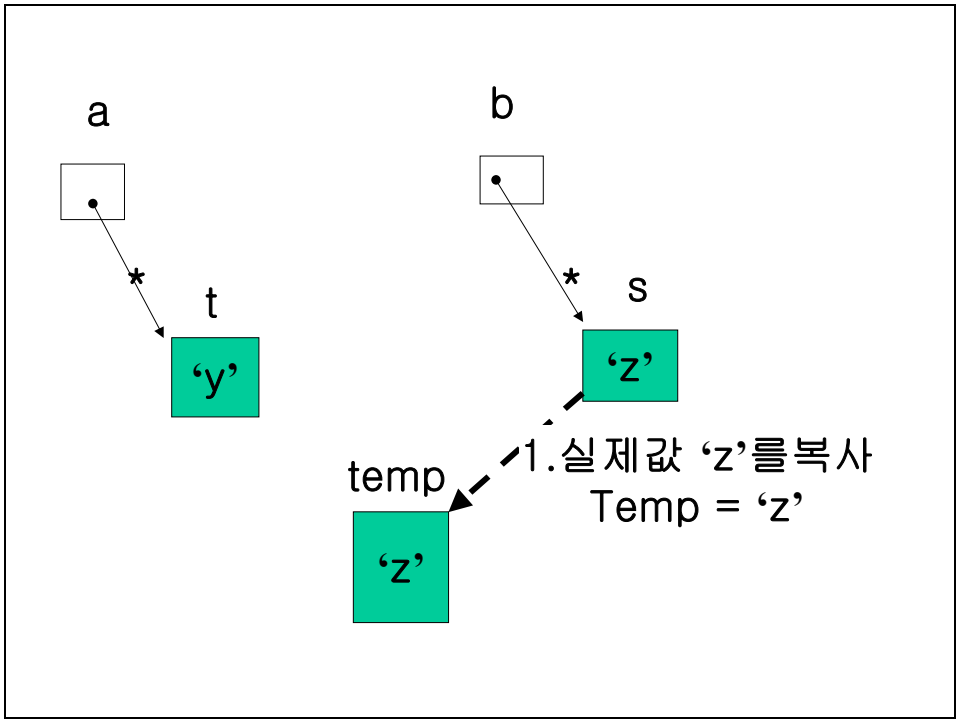
2.6.2.1.example1

Ex)
char t = ‘y’;
char s = ‘z’;
Swapping(t,s);

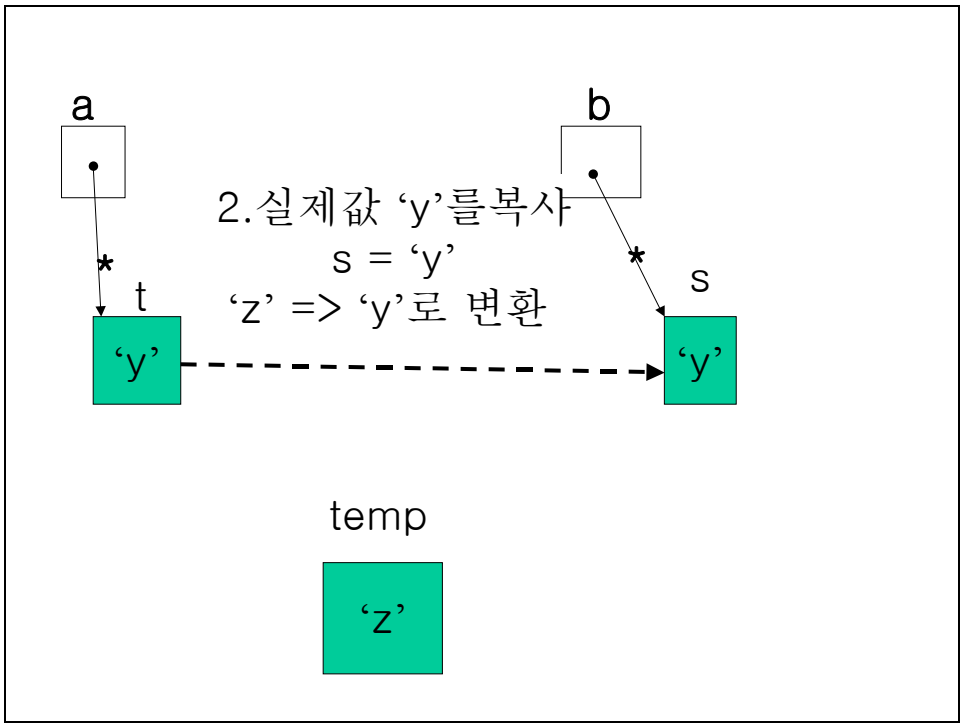
1)상태 1



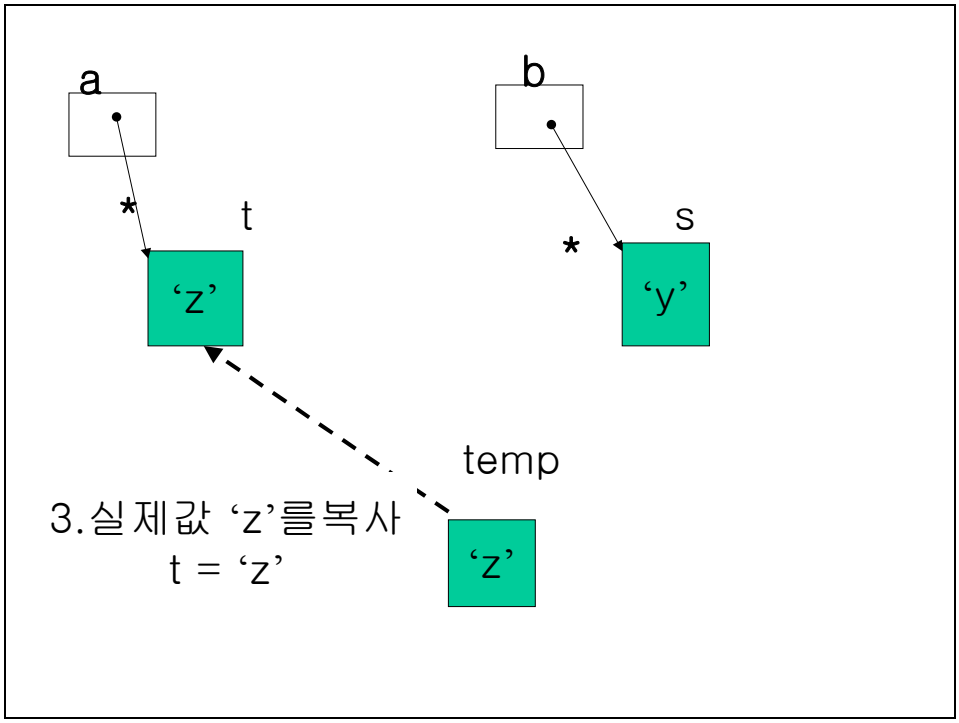
2) 상태 2


a
b
*
t
*
s
'y'
'z'
temp
1.실제값 'z'를복사
Temp = 'z'
'z'

3) 상태 3


a
b
2.실제값 'y'를복사
s = 'y'
'z' => 'y'로 변환
*
t
*
s
'y'
'y'
temp
'z'

4) 상태 4



2.7. memory allocation 방법

< 2.2.2 의 사용예제 소스>

```
void main
{
        G_Record *head = NULL;
        // [1-2-3]. 메모리 할당방법
        // 1. 메모리 할당법 1
        head = MemoryAlloc1();
        strcpy(head->name,"2pst");
        strcpy(head->number,"2");
        cout << head->name << head->number << "\n";
        free(head);

        // 2. 메모리 할당법 2
        MemoryAlloc2(&head);
        strcpy(head->name,"1pst");
        strcpy(head->number,"1");
        cout << head->name << head->number << "\n";
        free(head);

        // 3. 메모리 할당법 3
        // 3-1. G_Record 를 10 개정도 할당하기.
        head = (G_Record *)malloc(sizeof(G_Record)* 10);

        for( index =0; index <10; index++)
              head[index].index = index;

        free(head);

        // 3-2. 문자열 640*400+1 bytes 를 할당하는 방법
        temp =(char*) malloc(640*400+1);

        StringCpy(temp,"hi, hello son");

        free(temp);
}
```

2.7.1.방법 1(return 방법): G_Record *MemoryAlloc1(void)

```
typedef struct Record{
      char number[10];
      char name[30];
      char *next;
}G_Record;
```

G_Record *MemoryAlloc1(void)
1.기능: sizeof(G_Record)만큼의 공간을 할당 합니다.

2.필요한 데이터 :

3.필요한 함수

4. input

5. output
sizeof(G_Record)만큼의 공간을 할당한 후 그 포인터를 리턴합니다.

6. algorithm

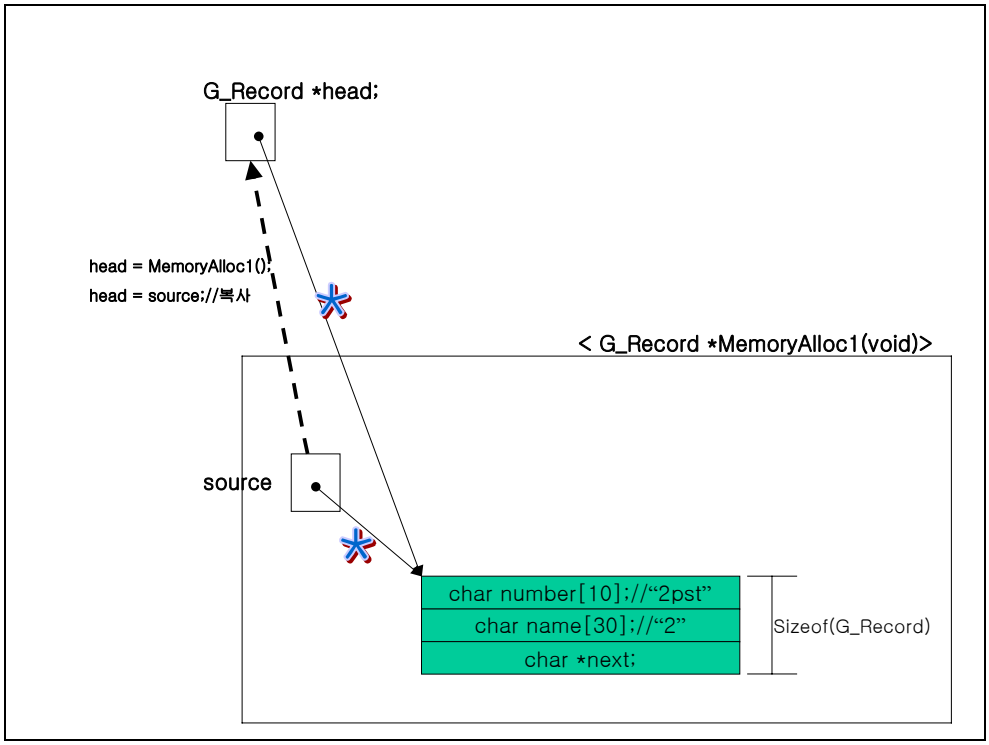
7. 참고

8.source

```
G_Record *MemoryAlloc1(void)
{
        G_Record * source ;
        source = (G_Record *)malloc(sizeof(G_Record));
        return source;
}
```

<예제. 2.2.2 소스>

```
void main
{
        G_Record *head = NULL;
        // 1. 메모리 할당법 1
        head = MemoryAlloc1();
        strcpy(head->name,"2pst");
        strcpy(head->number,"2");
        cout << head->name << head->number << "\n";
        free(head);
```



2.7.2.방법 2(parameter return 법)- G_Record *MemoryAlloc2()

```
Typedef struct Record{
        char number[10];
        char name[30];
        char *next;
}G_Record;
```

G_Record *MemoryAlloc2(G_Record **source)
1.기능: sizeof(G_Record)만큼의 공간을 할당 합니다.

2.필요한 데이터 :

3.필요한 함수

4. input(parameter)
1) G_Record **source : sizeof(G_Record)만큼의 공간을 할당한 후 포인터 source 의 값을 리턴합니다.

5. output
sizeof(G_Record)만큼의 공간을 할당한 후 포인터 *source 의 값을 리턴합니다

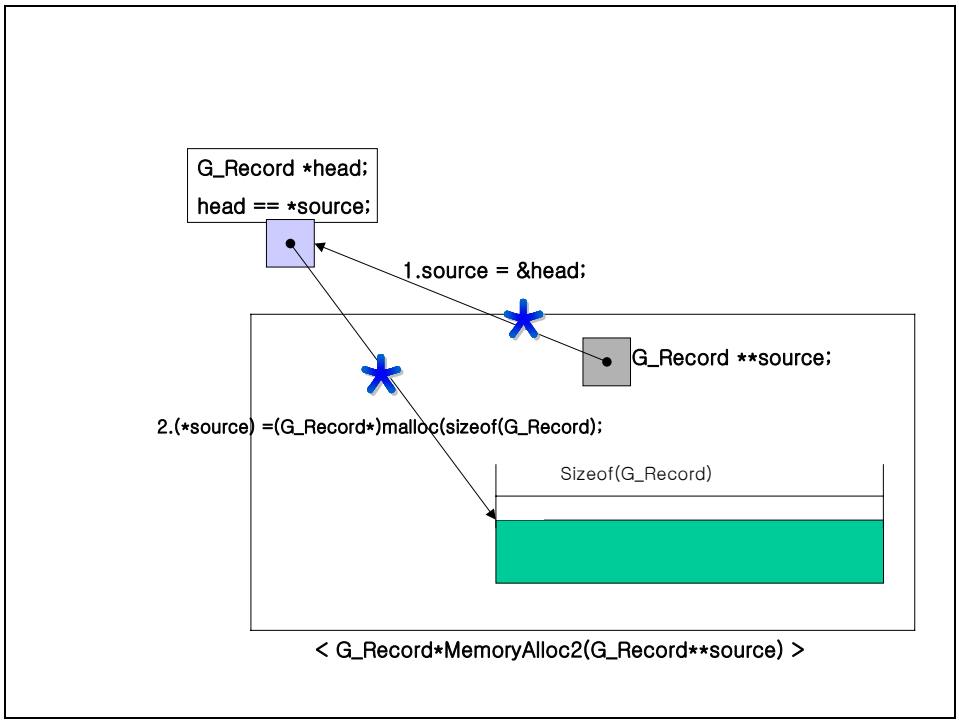
6. algorithm

7. source

```
G_Record *MemoryAlloc2(G_Record **source)
{
        *source = (G_Record *)malloc(sizeof(G_Record));
        return *source;
}
```

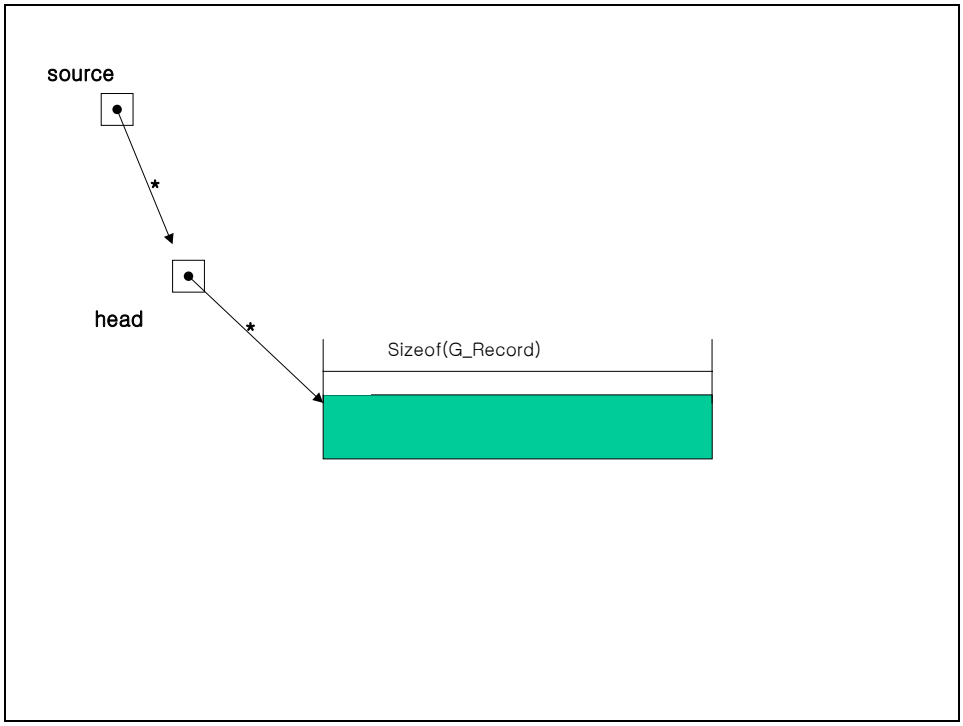
< 2.2.2 의 사용예제 소스>

```
void main
{
        G_Record *head = NULL;
        // 2. 메모리 할당법 2
        MemoryAlloc2(&head);
        strcpy(head->name,"1pst");
        strcpy(head->number,"1");
        cout << head->name << head->number << "\n";
        free(head);
}
```



2.7.2.1. 예제 설명

Ex)
G_Record *head;
MemoryAlloc2(&head);



1. MemoryAlloc2(&p);

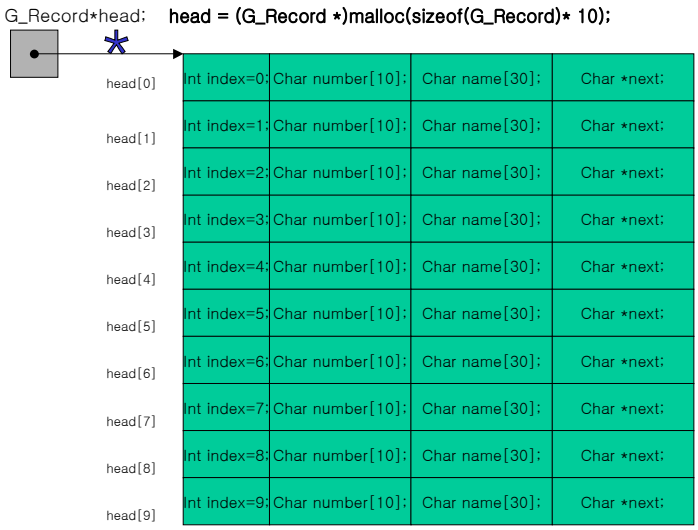
1) 원래 의도하고자 하는 목적
G_record *head = (G_Record *)malloc(sizeof(G_Record);혹은 head = (G_Record *)malloc(sizeof(G_Record);

2) 소스루틴 분석
G_Record ** source;에서 source = &head;가 복사된다.
그러면 source 변수 자체의미는 한번 가리키고(*) 또 한번 가리킨곳(*)에 **실제 값**(pointer 가 아닌 값)이 있다는 의미이므로
source == &head;
(*source) == head; (source 가 가리키는 곳의 실제값은 head 와 동일하다)
따라서
(*source) = (G_Record *)malloc(sizeof(G_Record);의미는 아래와 동일한 의미가 된다.
head = (G_Record *)malloc(sizeof(G_Record);

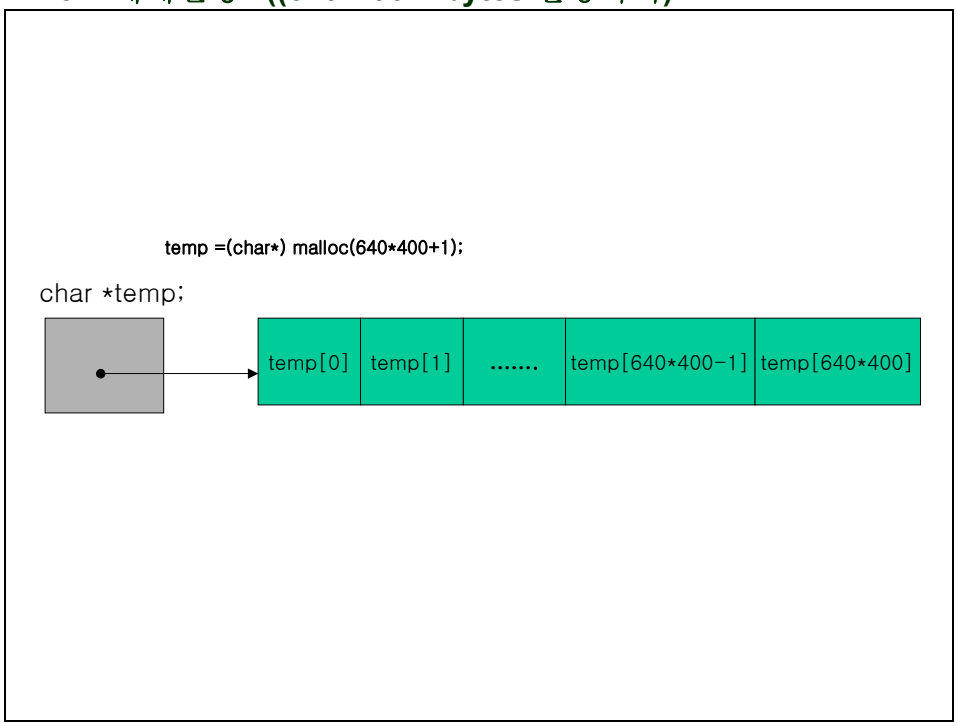
2.7.3.방법 3(일반적 malloc 함수 이용법)

```
void main
{
        G_Record *head = NULL;

        // 3. 메모리 할당법 3
        // 3-1. G_Record 를 10 개정도 할당하기.
        head = (G_Record *)malloc(sizeof(G_Record)* 10);

        for( index =0; index <10; index++)
              head[index].index = index;

        free(head);

        // 3-2. 문자열 640*400+1 bytes 를 할당하는 방법
        temp =(char*) malloc(640*400+1);

        StringCpy(temp,"hi, hello son");

        free(temp);
}
```

2.7.3.1 예제설명 1(G_Record 데이터를 10 배열공간잡기)



2.7.3.2.예제설명 2((640*400+1bytes 할당하기)



2.8.이중(2 차원) 포인터변수와 (1 차원)포인터 array

2.8.1.예제 1

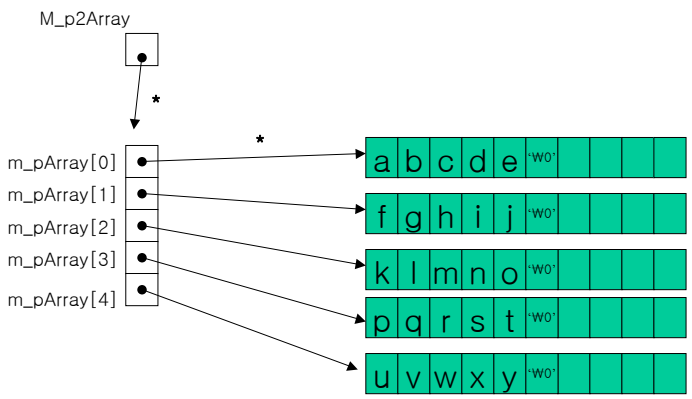
void Pointer::PointPoint()
1.기능:
char **m_p2Array;에 char *m_pArray[5];의 주소를 가리킵니다.
2.필요한 데이터 :

3.필요한 함수
4. input
5. output
6. algorithm
7. 참고
이중포인터 변수든 포인터 변수든 간에 모두 1 차원 어레이를 가리킨다.

8. source
< 2.2.2. source>

```
void Pointer::PointPoint()
{
        // 2 가지 방법
        // m_p2Array = m_pArray;
        m_p2Array = &m_pArray[0];

        for(int i =0; i< 5; i++)
                cout << m_p2Array[i] << "\n";

}
```

<참고. 2.3.생성자 : m_pArray 의 실제 data 가 초기화 되어있다.>



<기호>

= : means copy
== : means equal

2.8.1.1.초기화 부분

초기화.	기능
m_p2Array = m_pArray == &m_pArray[0]으로 초기화	m_pArray array 를 1 차원 배열로 인식하고 접근한다. 위 프로그램 예제에 근거한다면, m_p2Array 는 m_pArray 가 0 부터 4 까지 있는지를 모르므로, 단지 프로그래머가 4 까지만 접근하도록 제약해야한다. (단 이 예제 소스에서…)

2.8.1.2. 동일한 주소값

1.8.1.초기화한 후 성립되는 동일한 주소값

소스초기화 m_p2Array = &m_pArray[0] ; // 또는 m_p2Array = m_pArray;
m_p2Array == m_pArray == &m_pArray[0];
m_p2Array[0] == m_pArray[0] == “abcde”를 가리키는 주소값(&m_pArray[0][0] == &m_p2Array[0][0]) &m_p2Array[0][1] == &m_pArray[0][1] == “bcde”를 가리키는 주소값 &m_p2Array[0][2] == &m_pArray[0][2] == “cde”를 가리키는 주소값 &m_p2Array[0][3] == &m_pArray[0][3] == “de”를 가리키는 주소값 &m_p2Array[0][4] == &m_pArray[0][4] == “e”를 가리키는 주소값
m_p2Array[1] == m_pArray[1] == “fghij”를 가리키는 주소값(&m_pArray[1][0] == &m_p2Array[1][0]) &m_p2Array[1][1] == &m_pArray[1][1] == “ghij”를 가리키는 주소값 &m_p2Array[1][2] == &m_pArray[1][2] == “hij”를 가리키는 주소값 &m_p2Array[1][3] == &m_pArray[1][3] == “ij”를 가리키는 주소값 &m_p2Array[1][4] == &m_pArray[1][4] == “j”를 가리키는 주소값
m_p2Array[2] == m_pArray[2] == “klmno”를 가리키는 주소값(&m_pArray[2][0] == &m_p2Array[2][0]) &m_p2Array[2][1] == &m_pArray[2][1] == “lmno”를 가리키는 주소값 &m_p2Array[2][2] == &m_pArray[2][2] == “mno”를 가리키는 주소값 &m_p2Array[2][3] == &m_pArray[2][3] == “no”를 가리키는 주소값 &m_p2Array[2][4] == &m_pArray[2][4] == “o”를 가리키는 주소값
m_p2Array[3] == m_pArray[3] == “pqrst”를 가리키는 주소값(&m_pArray[3][0] == &m_p2Array[3][0]) &m_p2Array[3][1] == &m_pArray[3][1] == “qrst”를 가리키는 주소값 &m_p2Array[3][2] == &m_pArray[3][2] == “rst”를 가리키는 주소값 &m_p2Array[3][3] == &m_pArray[3][3] == “st”를 가리키는 주소값 &m_p2Array[3][4] == &m_pArray[3][4] == “t ”를 가리키는 주소값
m_p2Array[4] == m_pArray[4] == “uvwxy”를 가리키는 주소값(&m_pArray[4][0] == &m_p2Array[4][0]) &m_p2Array[4][1] == &m_pArray[4][1] == “vwxy”를 가리키는 주소값 &m_p2Array[4][2] == &m_pArray[4][2] == “wxy”를 가리키는 주소값 &m_p2Array[4][3] == &m_pArray[4][3] == “xy”를 가리키는 주소값 &m_p2Array[4][4] == &m_pArray[4][4] == “y”를 가리키는 주소값

<기호>
= : means copy
== : means equal

2.8.1.3.동일한 실제값

=> 1.8.1.에서 초기화 한후 동일한 실제값들.

m_p2Array[0][0]== m_pArray[0][0] == 'a' m_p2Array[0][1]== m_pArray[0][1] == 'b' m_p2Array[0][2]== m_pArray[0][2] == 'c' m_p2Array[0][3]== m_pArray[0][3] == 'd' m_p2Array[0][4]== m_pArray[0][4] == 'e'	"abcde"의 각 바이트값을 접근
m_p2Array[1][0]== m_pArray[1][0] == 'f' m_p2Array[1][1]== m_pArray[1][1] == 'g' m_p2Array[1][2]== m_pArray[1][2] == 'h' m_p2Array[1][3]== m_pArray[1][3] == 'i' m_p2Array[1][4]== m_pArray[1][4] == 'j'	"fghij"의 각 바이트값을 접근
m_p2Array[2][0]== m_pArray[2][0] == 'k' m_p2Array[2][1]== m_pArray[2][1] == 'l' m_p2Array[2][2]== m_pArray[2][2] == 'm' m_p2Array[2][3]== m_pArray[2][3] == 'n' m_p2Array[2][4]== m_pArray[2][4] == 'o'	"klmno"의 각 바이트값을 접근
m_p2Array[3][0]== m_pArray[3][0] == 'p' m_p2Array[3][1]== m_pArray[3][1] == 'q' m_p2Array[3][2]== m_pArray[3][2] == 'r' m_p2Array[3][3]== m_pArray[3][3] == 's' m_p2Array[3][4]== m_pArray[3][4] == 't'	"pqrst"의 각 바이트값을 접근
m_p2Array[4][0]== m_pArray[4][0] == 'u' m_p2Array[4][1]== m_pArray[4][1] == 'v' m_p2Array[4][2]== m_pArray[4][2] == 'w' m_p2Array[4][3]== m_pArray[4][3] == 'x' m_p2Array[4][4]== m_pArray[4][4] == 'y'	"uvwxy"의 각 바이트값을 접근

2.8.2.예제 2- sizeof(G_Record) : G_Record 의 size

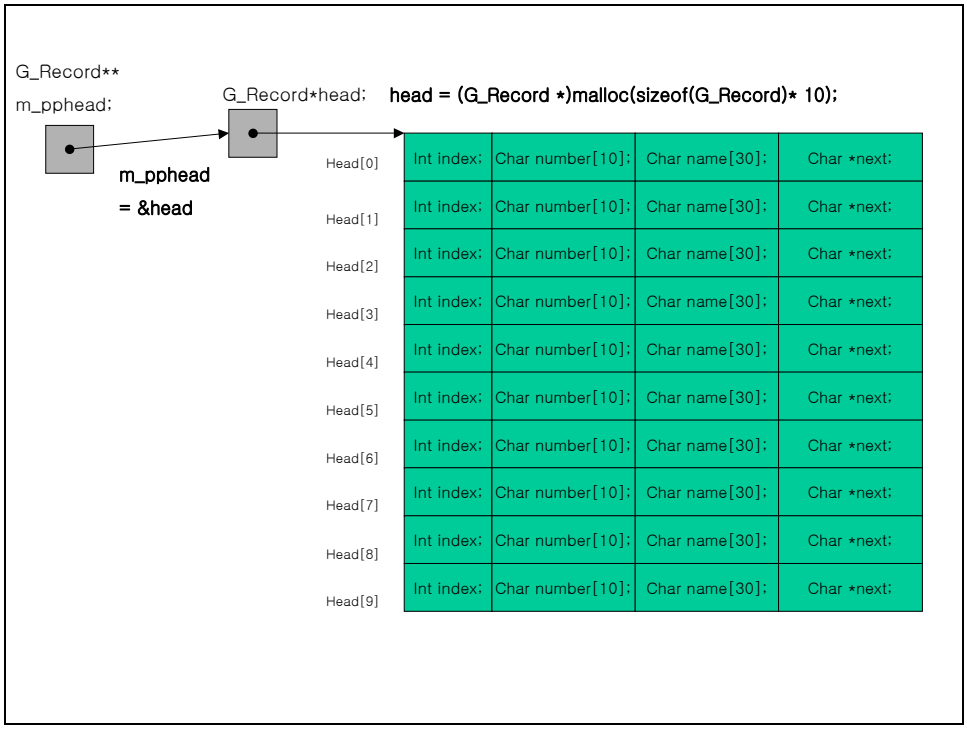
```
typedef struct Record
{
      char number[10];
      char name[30];
      char *next;
}G_Record;
```

<참고 2.2 소스 참고>

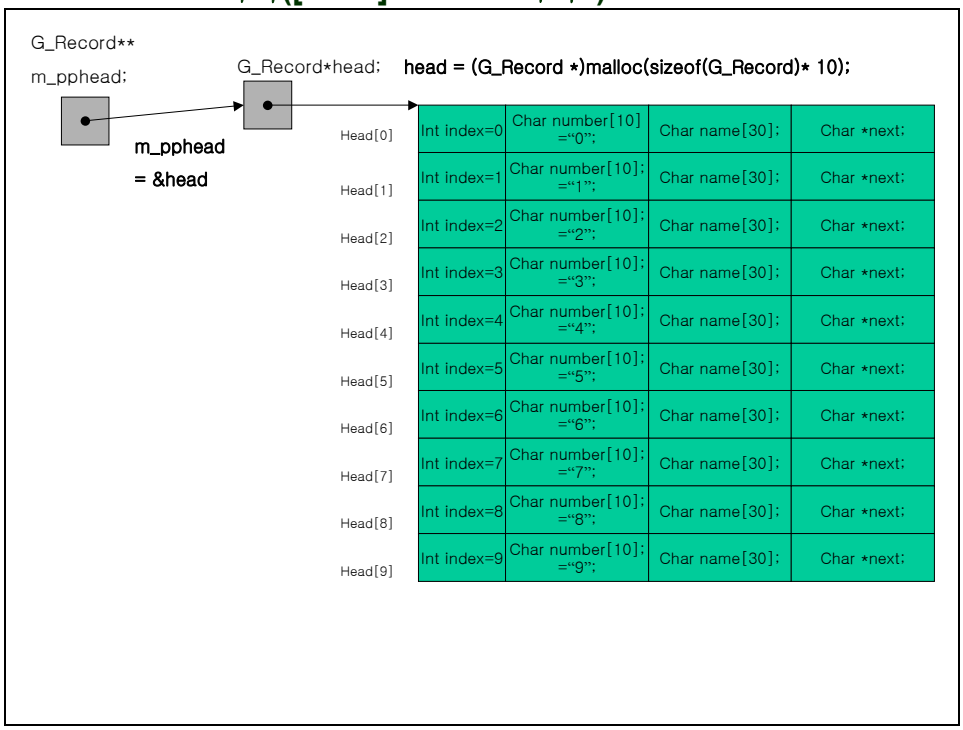
```
void main()
{
      G_Record **m_pphead = NULL;
      G_Record *head = NULL;

      // [2-2]. G_Record 를 10 개정도 할당하기.
      // [2-2-1]. 초기화부분
      head = (G_Record *)malloc(sizeof(G_Record)* 10);
      m_pphead = &head;

      // [2-2-2]. Data 초기화
      for( index =0; index <10; index++)
      {
            head[index].index = index;
            cout<< "m_pphead" << index, m_pphead[0][index].index;

            name[0] = '0'+index;
            name[1] = NULL;
            StringCpy(head[index].name,    name);
      }
      cout<<"end";
      free(head);
}
```

2.8.2.1.초기화부분

```
head = (G_Record *)malloc(sizeof(G_Record)* 10);
m_pphead = &head;
```


G_Record**
m_pphead;
G_Record*head;
head = (G_Record *)malloc(sizeof(G_Record)* 10);
m_pphead
= &head
Head[0]
Head[1]
Head[2]
Head[3]
Head[4]
Head[5]
Head[6]
Head[7]
Head[8]
Head[9]
Int index;
Char number[10];
Char name[30];
Char *next;

2.8.2.2.data 초기화([2-2-2]. Data 초기화)



2.8.2.3.동일한 data,pointer

2.8.2.3.1.동일한 record 단위의 주소값, atrribute 주소값

가. 초기화

소스초기화 m_pphead = &head ;

나 동일한 record 단위 주소값

m_pphead[0] == **&m_pphead[0][0]** == &(*(*(m_pphead+0)+0))==***(m_pphead+0)** == head == &head[0] == &(*(head+0)) == **head[0]의 int index=0 을 가리키는 주소값**
m_pphead[0]+1== &m_pphead[0][1] == &(*(*(m_pphead+0)+1))==***(m_pphead+0)+1** ==head+1== &head[1] == &(*(head+1)) == **head[1]의 int index=1 을 가리키는 주소값**
m_pphead[0]+2==&m_pphead[0][2] == &(*(*(m_pphead+0)+2))==***(m_pphead+0)+2** ==head+2== &head[2] == &(*(head+2)) == **head[2]의 int index=2 을 가리키는 주소값**
……..
m_pphead[0]+9==&m_pphead[0][9] == &(*(*(m_pphead+0)+9))==***(m_pphead+0)+9** ==head+9== &head[9] == &(*(head+9)) == **head[9]의 int index=9 을 가리키는 주소값**

[참고 : +1, +9, -1, -9 등의 의미] (n: 정수)

+n 의 의미는 m_pphead[0], m_pphead[0][j]가 담고 있는 값이 주소값(pointer) 혹은 실제값이냐에 따라 해석되는 의미가 다르다.

=> m_pphead[0]+n == **m_pphead[0]이 담고 있는 값(==주소값, pointer)** + n***sizeof(G_Record)**의 의미이다.

즉 현재 레코드로부터 n 번째 record 를 가리키는 포인터로 이동하라는 의미.

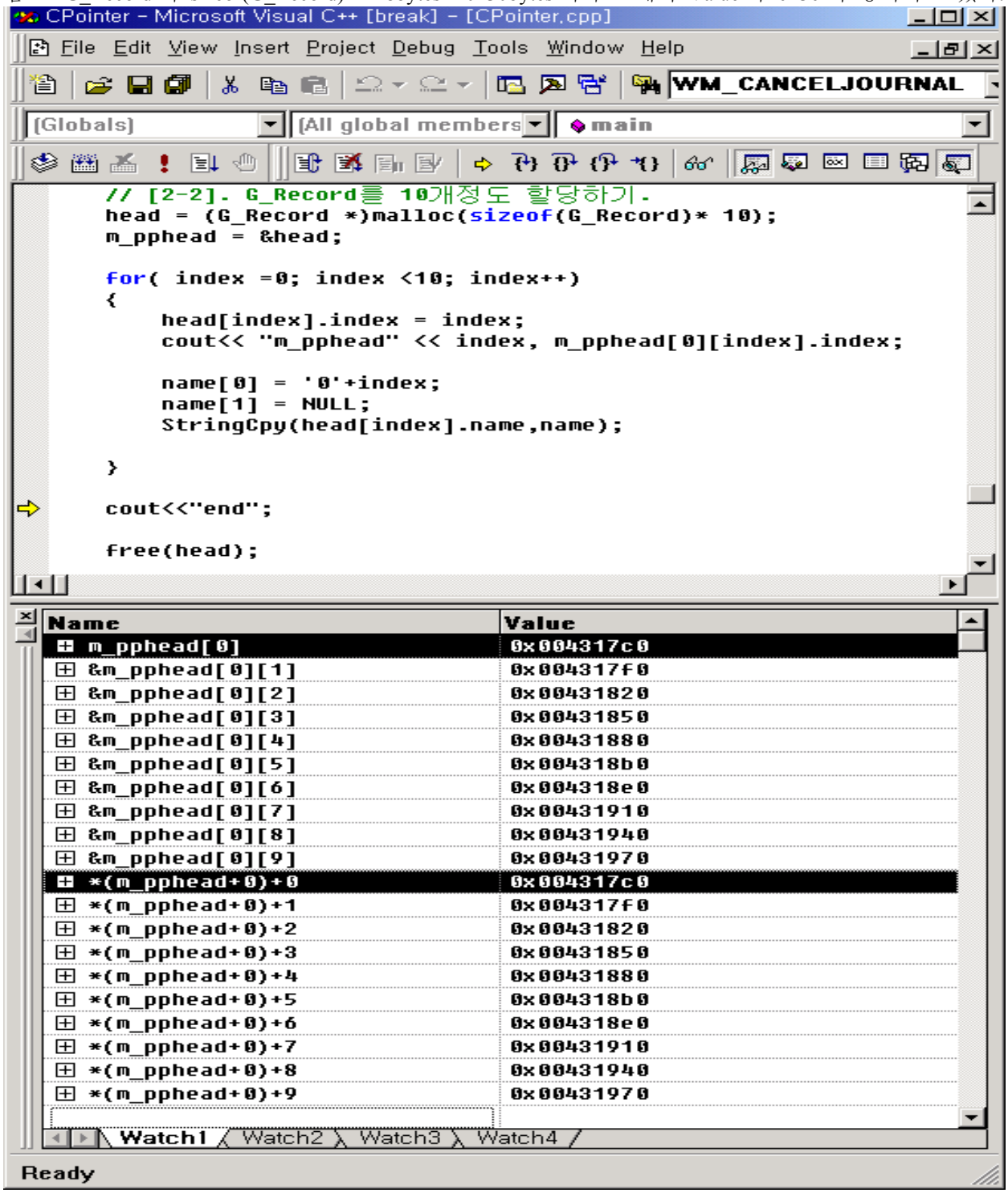
=> m_pphead[0][j].index +n == m_phead[0][j].index 가 담고 있는 값(==실제값)에 + n 을 해준다. (j : 0 ~9)

다. Debugging 화면

- Name : pointer

- Value : pointer 값(주소값)

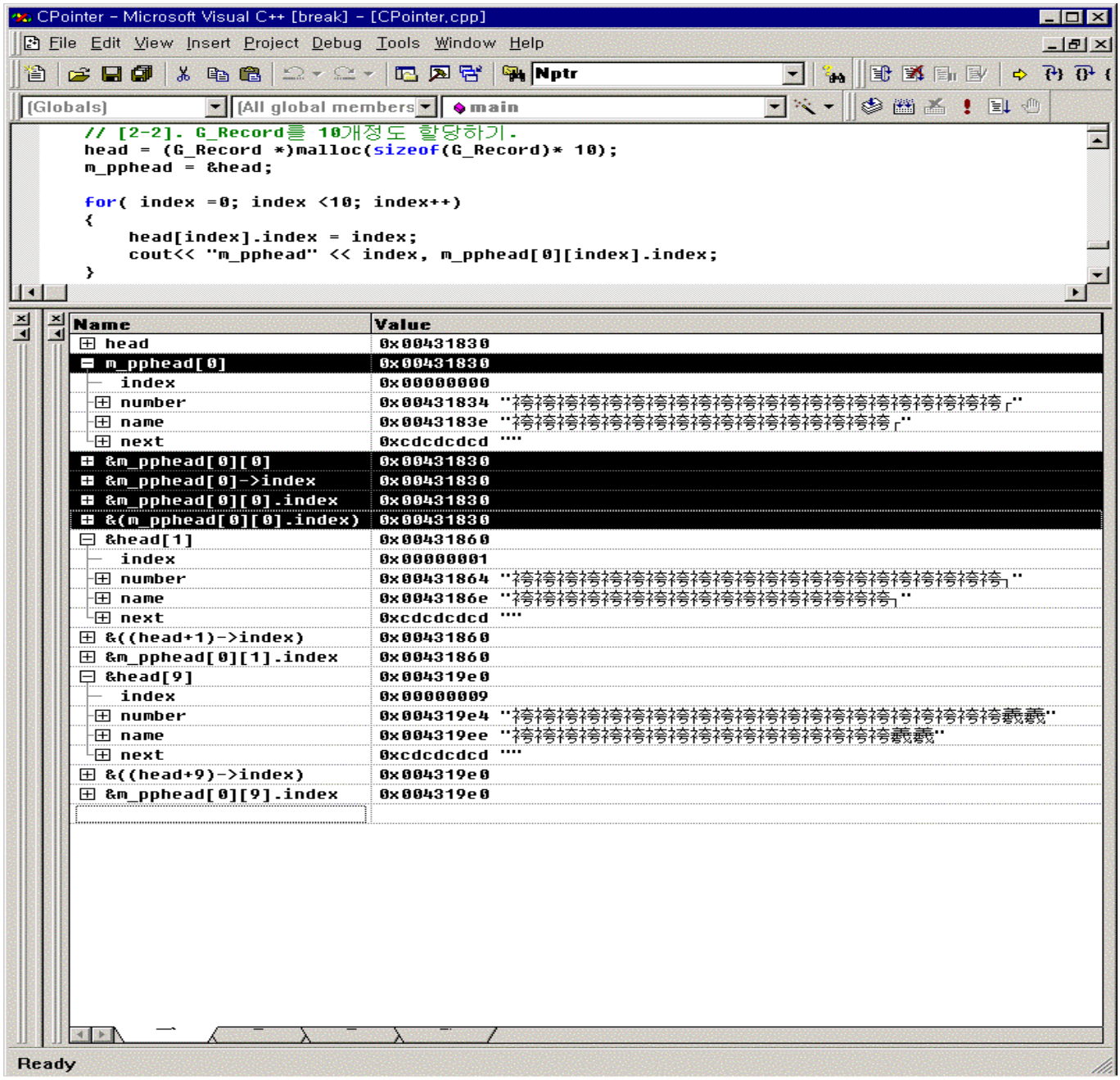
참고> G_Record 의 sizeof(G_Record)== 48bytes==0x30bytes 이다 그래서 Value 가 **0x30** 씩 증가하고 있다.



라. 동일한 record 의 attribute 주소값(많이 사용하는문법형태는 굵은 글씨)

1. index 의 주소값
Head == &head[0]==m_pphead[0] == &m_pphead[0][0] ==0th record 를 가리키는 구조체포인터의 1st attribute 를 가리키는 pointer **==&(m_pphead[0]->index) == &(m_pphead[0][0].index) == &m_pphead[0][0].index** == **&(head ->index)== &(head[0].index)** == &((*(head+0)).index) == head[0]의 int index=0 을 가리키는 주소값
Head+1==&Head[1]==m_pphead[0]+1==&m_pphead[0][1]== 1th record 를 가리키는 구조체포인터의 1st attribute 를 가리키는 pointer **==&(m_pphead[0][1].index) == &m_pphead[0][1].index** == &((head+1)->index)== **&(head[1].index)** == &((*(head+1)).index) == head[1]의 int index=1 을 가리키는 주소값
……..
Head+9==&head[9]==m_pphead[0]+9==&m_pphead[0][9]== 9th record 를 가리키는 구조체포인터의 1st attribute 를 가리키는 pointer **== &(m_pphead[0][9].index) == &m_pphead[0][9].index** == &((head+9)->index)== **&(head[9].index)** == &((*(head+9)).index) == head[9]의 int index=9 을 가리키는 주소값

<그림 G_Record 의 index 라는 attribute 의 주소값 >

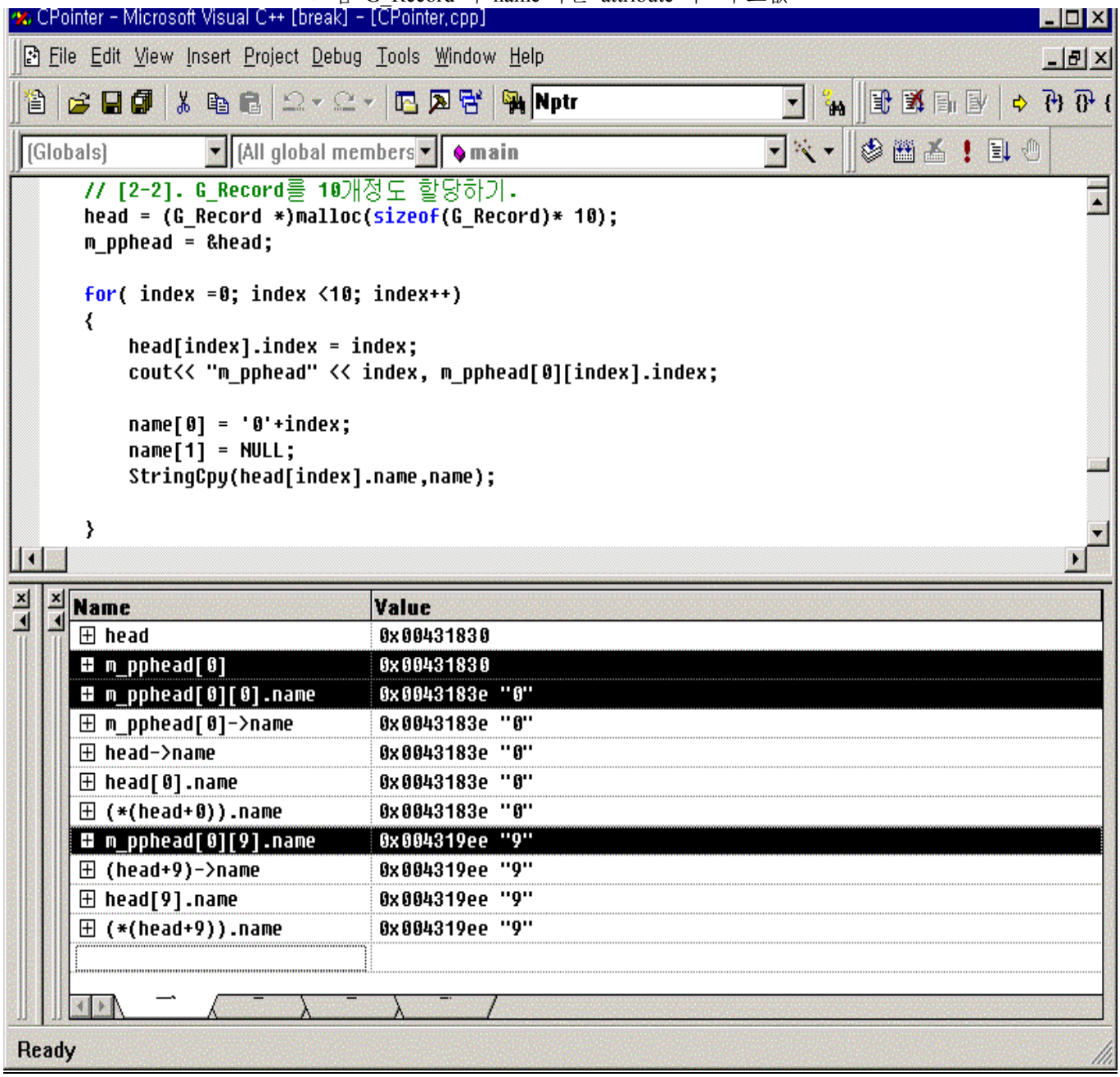


2. G_Record 의 char name[30]을 가리키는 주소값
M_pphead[0]->name == m_pphead[0][0].name == **head ->name==** **head[0].name** == (*(head+0)).name == head[0]의 char name[30]==”0”을 가리키는 주소값
M_pphead[0][1].name == (head+1)->name== **head[1].name** == (*(head+1)).name == head[1]의 char name[30]==”1”을 가리키는 주소값
……
M_pphead[0][9].name == (head+9)->name== **head[9].name** == (*(head+9)).name == head[9]의 char name[30]==”9”을 가리키는 주소값

[참고]
- char name[30];에서 name==&name[0]이라는 주소값을 의미한다.
- char name[30];은 **G_Record 구조체** 내에서 상대적 주소(offset)가
  14bytes(0x0Ebytes)==sizeof(int)+sizeof(char number[10])이다.

[참고 : +1, +9, -1, -9 등의 의미] (n: 정수)
+n 의 의미는 m_pphead[0], m_pphead[0][j]가 담고 있는 값이 주소값(pointer) 혹은 실제값이냐에 따라 해석되는 의미가 다르다.
=> m_pphead[0]+**n** == **m_pphead[0]이 담고 있는 값(==주소값, pointer) + n*sizeof(G_Record)**의 의미이다.
즉 현재 레코드로부터 n 번째 record 를 가리키는 포인터로 이동하라는 의미.
=> m_pphead[0][j].index +n == m_phead[0][j].index 가 담고 있는 값(==실제값)에 + n 을 해준다. (j : 0 ~9)

<그림 G_Record 의 name 라는 attribute 의 주소값>



2.8.2.3.2.동일한 record 단위 실제값

가. 초기화

소스초기화 m_pphead = &head ;

나 동일한 실제값

m_pphead[0][0] == *(*(m_pphead+0)+0)==*(m_pphead[0]**+0)** == *(m_pphead+0)[0] head[0] == *(head+0) **== head[0]이라는 G_Record 구조체의 실제값**
m_pphead[0][1] == *(*(m_pphead+0)+1)==*(m_pphead[0]**+1)** == *(m_pphead+0)[1] head[1] == *(head+1) **== head[1]이라는 G_Record 구조체의 실제값**
m_pphead[0][2] == *(*(m_pphead+0)+2)==*(m_pphead[0]**+2)** == *(m_pphead+0)[2] head[2] == *(head+2) **== head[2]이라는 G_Record 구조체의 실제값**
m_pphead[0][9] == *(*(m_pphead+0)+9)==*(m_pphead[0]**+9)** == *(m_pphead+0)[9] head[9] == *(head+9) **== head[9]이라는 G_Record 구조체의 실제값**

[참고 : +1, +9, -1, -9 등의 의미] (n: 정수)

+n 의 의미는 m_pphead[0], m_pphead[0][j]가 담고 있는 값이 주소값(pointer) 혹은 실제값이냐에 따라 해석되는 의미가 다르다.

=> m_pphead[0]+n == **m_pphead[0]이 담고 있는 값(==주소값, pointer)** + n***sizeof(G_Record)**의 의미이다.

즉 현재 레코드로부터 n 번째 record 를 가리키는 포인터로 이동하라는 의미.

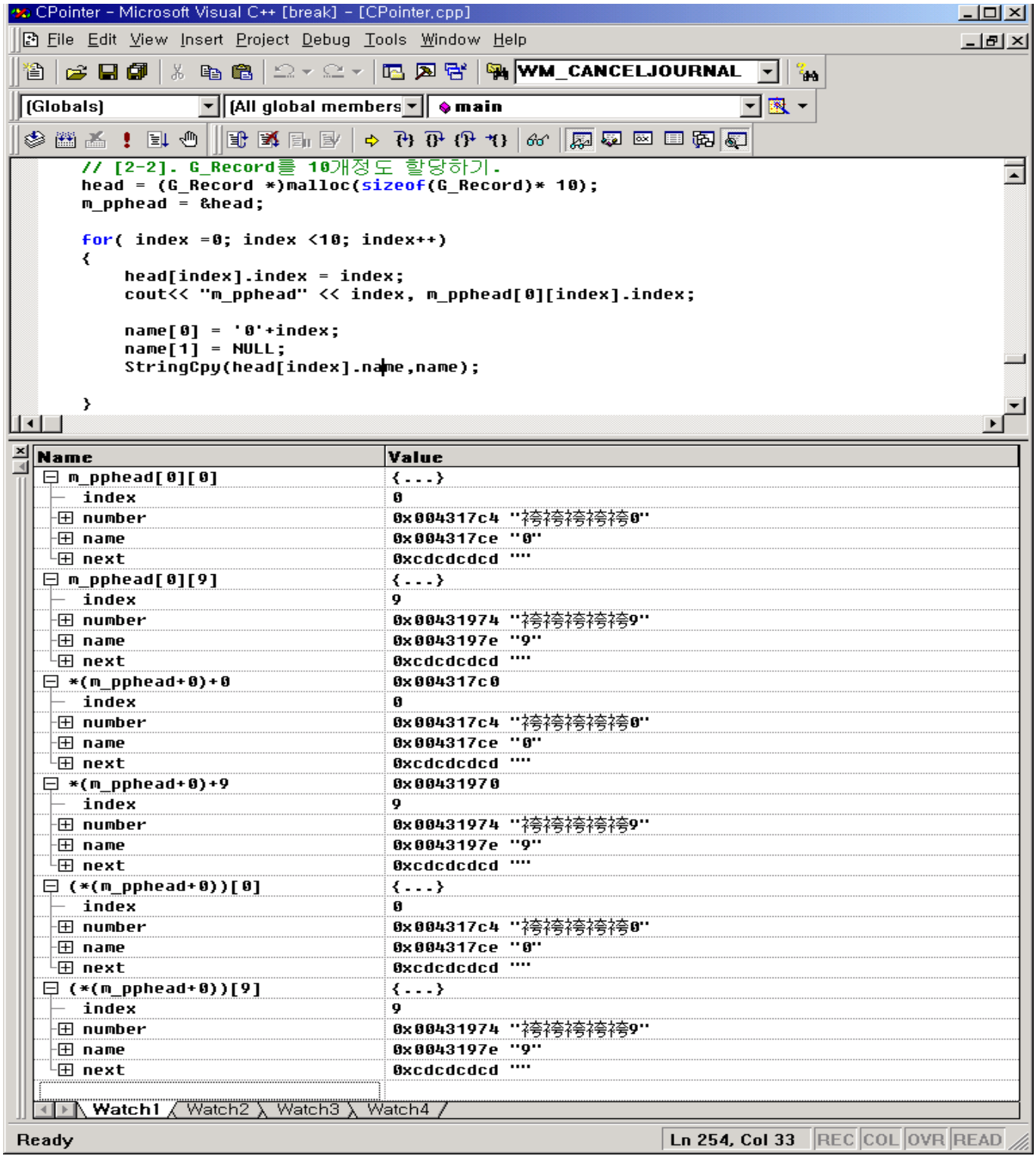
=> m_pphead[0][j].index +n == m_phead[0][j].index 가 담고 있는 값(==실제값)에 + n 을 해준다. (j : 0 ~9)

다. Debugging 화면

- Name : pointer
- Value : pointer 가 가리키고 있는 실제값

참고> G_Record 의 sizeof(G_Record)== 48bytes==0x30bytes 이다 그래서 Value 가 **0x30** 씩 증가하고 있다.

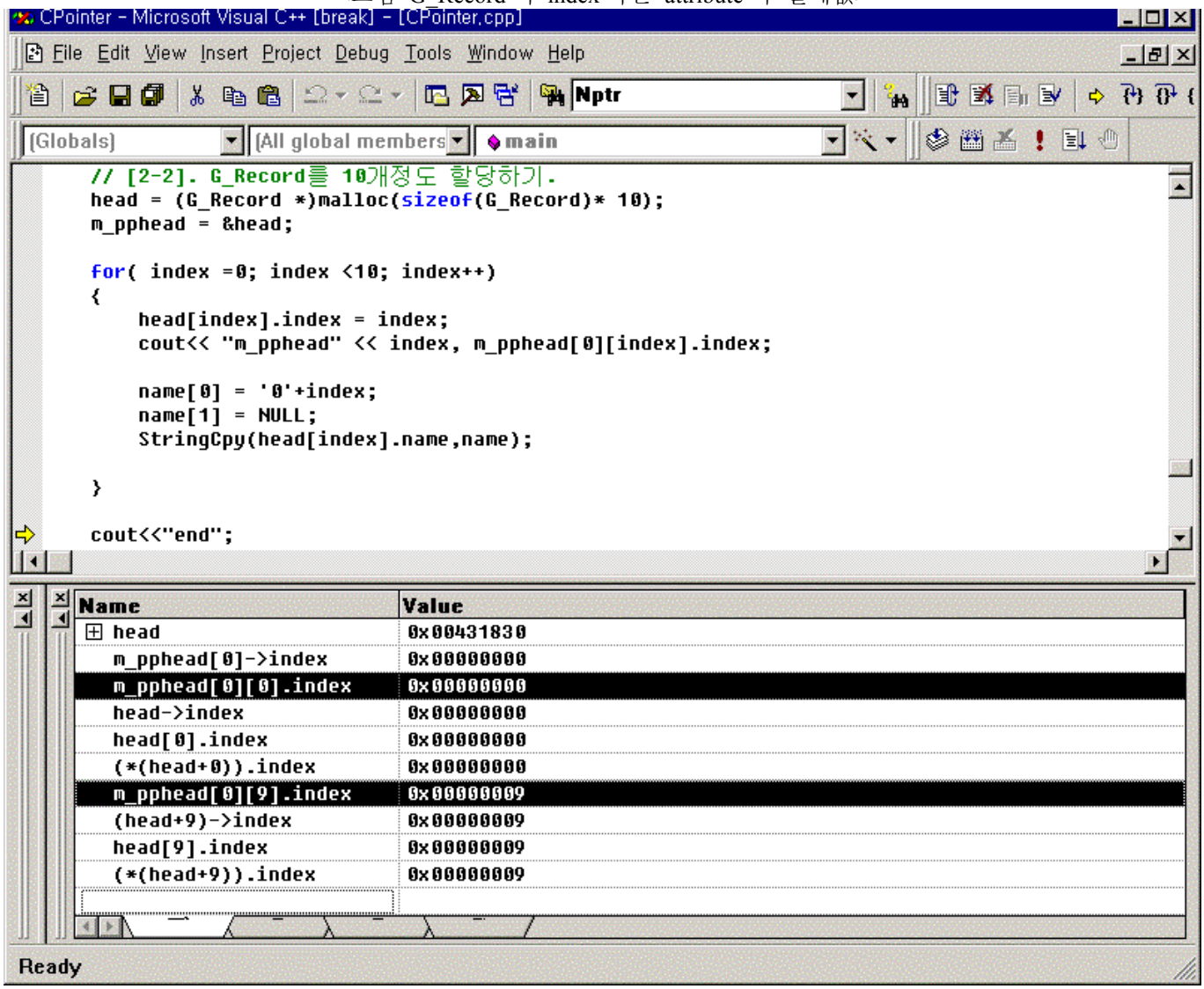
Index 값의 변화를 보라.



라. 동일한 record 의 attribute 실제값(많이 사용하는문법형태는 굵은 글씨)

1. index 의 실제값
m_pphead[0]->index== m_pphead[0][0].index == **head ->index== head[0].index** == (*(head+0)).index == head[0]의 int index=0 을 가리키는 실제값
m_pphead[0][1].index == (head+1)->index== **head[1].index** == (*(head+1)).index == head[1]의 int index=1 을 가리키는 실제값
……
m_pphead[0][9].index == (head+9)->index== **head[9].index** == (*(head+9)).index == head[9]의 int index=9 을 가리키는 실제값

<그림 G_Record 의 index 라는 attribute 의 실제값>



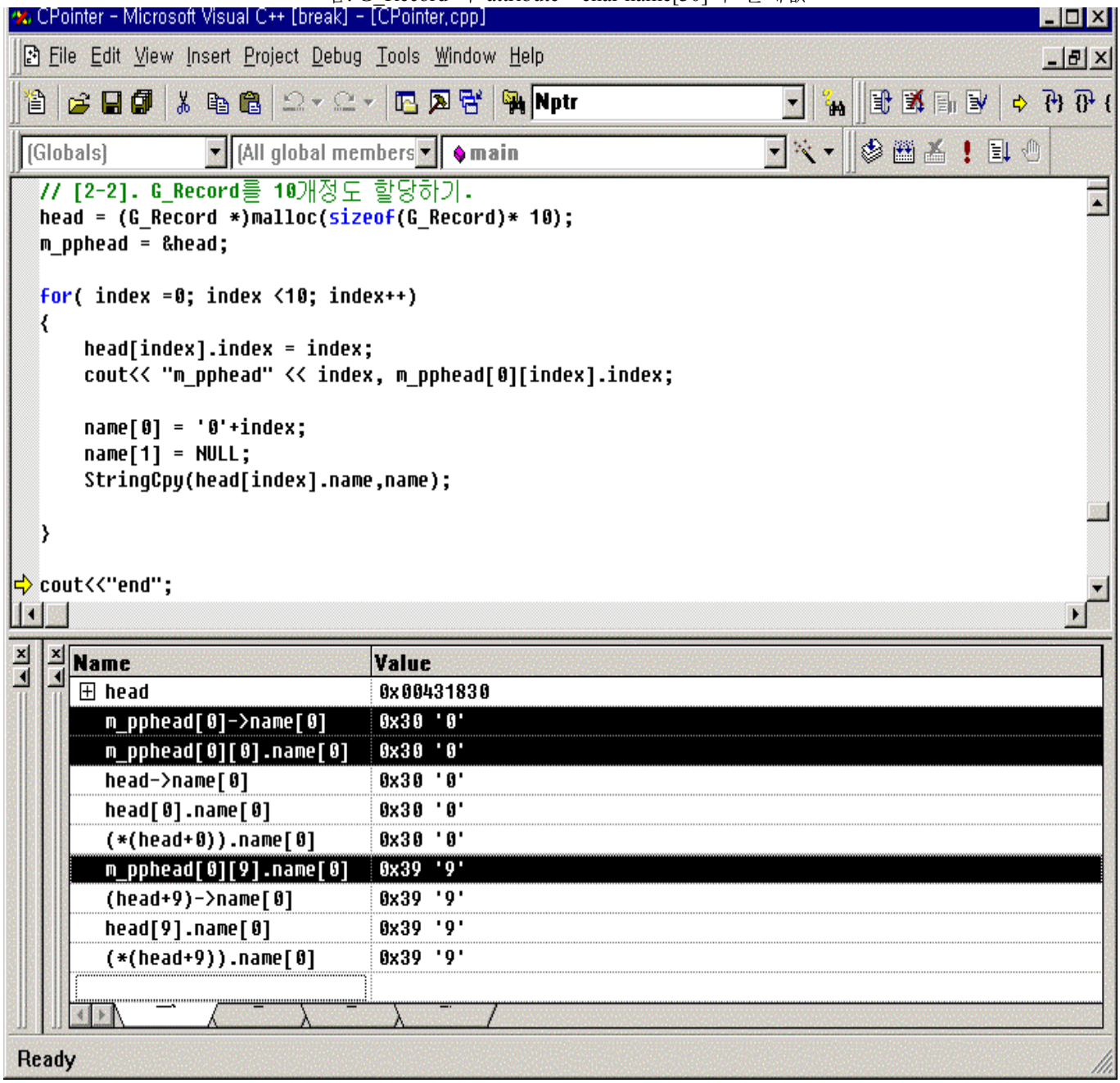
2. char name[30]의 실제값
m_pphead[0]->name[0]== m_pphead[0][0].name[0] == **head ->name[0]== head[0].name[0]** == (*(head+0)).name[0] == head[0]의 name[0]==”0”을 가리키는 실제값
m_pphead[0][1].name[0] == (head+1)->name[0]== **head[1].name[0]** == (*(head+1)).name[0] == head[1]의 name[0]==”1”을 가리키는 실제값
……..
m_pphead[0][9].name[0] == (head+9)->name[0]== **head[9].name[0]** == (*(head+9)).name[0] == head[9]의 name[0]==”9”을 가리키는 실제값

m_pphead[0]->name[1]== m_pphead[0][0].name[1] == **head ->name[1]== head[0].name[1]** == (*(head+0)).name[1] == head[0]의 name[1]을 가리키는 실제값
m_pphead[0][1].name[1] == (head+1)->name[1]== **head[1].name[1]** == (*(head+1)).name[1] == head[1]의 name[1]을 가리키는 실제값
……..
m_pphead[0][9].name[1] == (head+9)->name[1]== **head[9].name[1]** == (*(head+9)).name[1] == head[9]의 name[1]을 가리키는 실제값

…….
……..중략……

m_pphead[0]->name[30]== m_pphead[0][0].name[30] == **head ->name[30]== head[0].name[30]** == (*(head+0)).name[30] == head[0]의 name[0]을 가리키는 실제값
m_pphead[0][1].name[30] == (head+1)->name[30]== **head[1].name[30]** == (*(head+1)).name[30] == head[1]의 name[30]을 가리키는 실제값
……..
m_pphead[0][9].name[0] == (head+9)->name[30]== **head[9].name[30]** == (*(head+9)).name[30] == head[9]의 name[30]을 가리키는 실제값

<그림. G_Record 의 attribute char name[30]의 실제값 >



2.8.3.예제 3 - sizeof(G_Record *) : 주소값 4bytes

```
typedef struct Record
{
	char number[10];
	char name[30];
	char *next;
}G_Record;
```

```
void main()
{
	G_Record **m_pphead = NULL;
	char name[30];
	int index =0;

	// [1]. Malloc m_pphead
	m_pphead = (G_Record **)malloc(sizeof(G_Record*)*10);

	for( index =0; index <10; index++)
	{
		m_pphead[index] = (G_Record *)malloc(sizeof(G_Record));
		m_pphead[index]->index = index;

		name[0] = '0'+index;
		name[1] = NULL;
		StringCpy(m_pphead[index]->name,name);

	}

	for( index =0; index <10; index++)
		free(m_pphead[index]);
	free(m_pphead);
}
```

- sizeof(G_Record *)의 의미

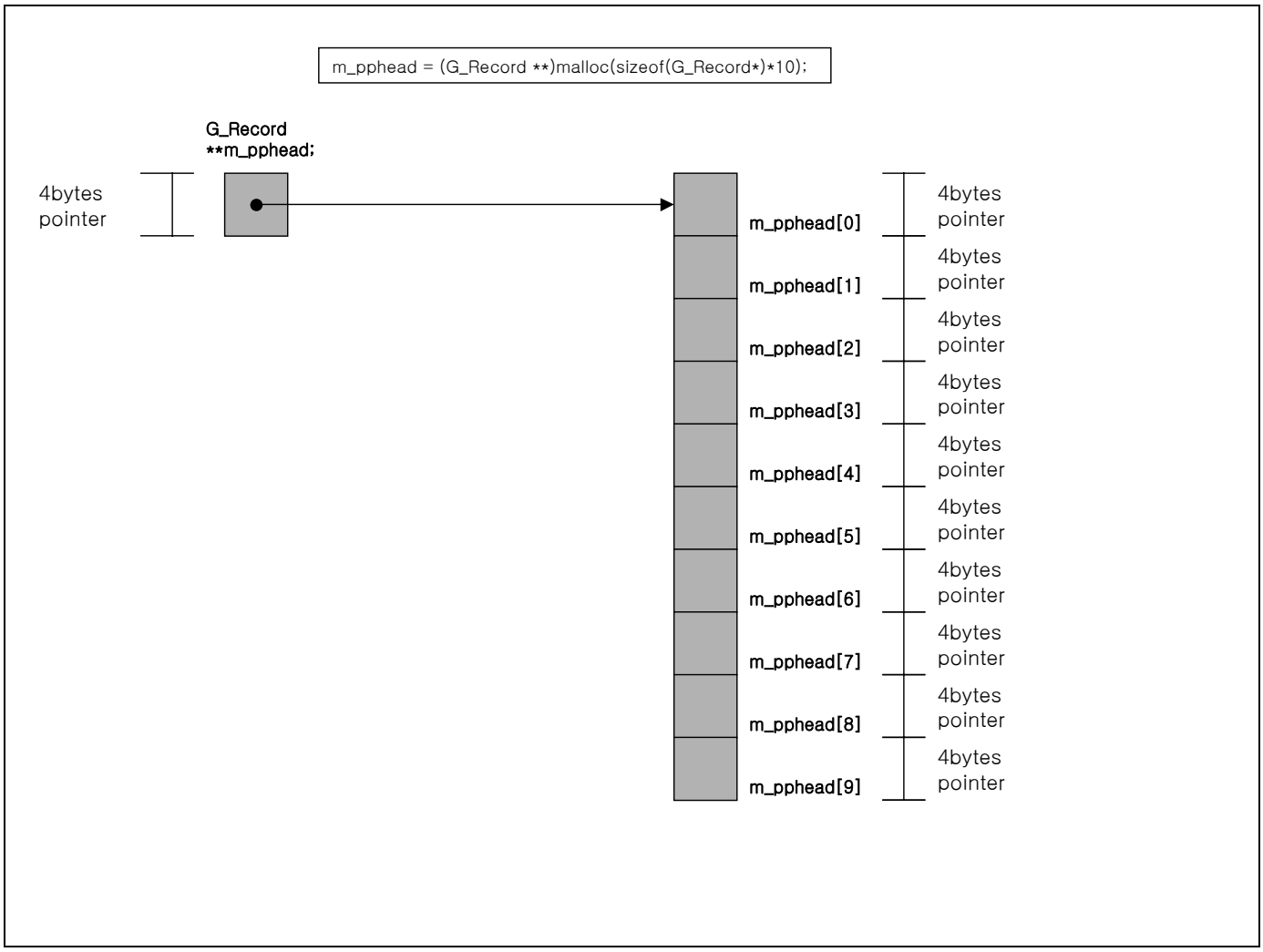
  :"G_Record data type 을 가리키는(Point 하는) 변수"의 사이즈

  : pointer 를 담을 곳이므로 **4bytes** 의 pointer 이다.

- Sizeof(G_Record)

  G_Record 의 사이즈.

2.8.3.1. two dimension array pointer 초기화



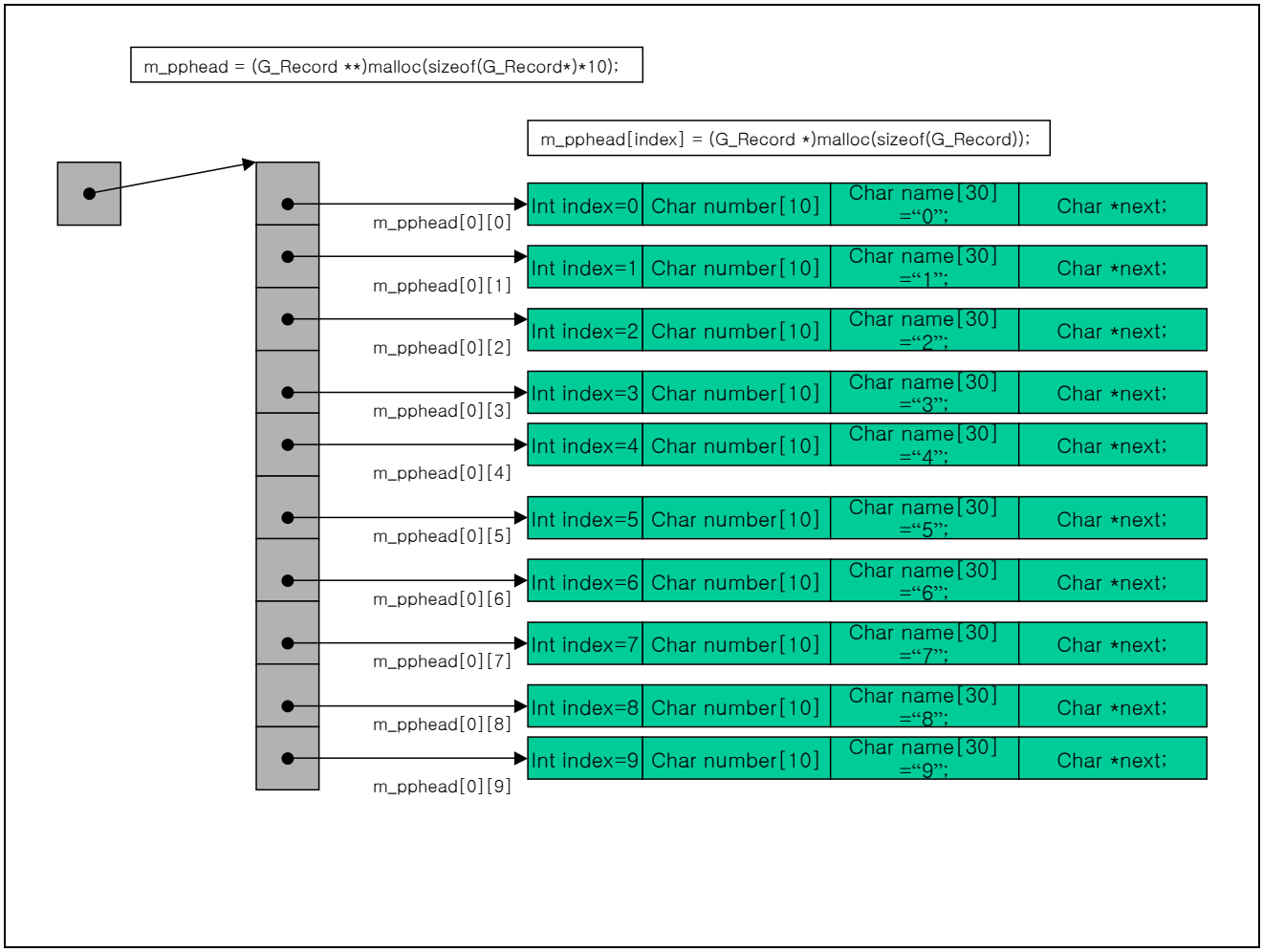
=>[m_pphead = **(G_Record **)** malloc(sizeof(G_Record *) * 10);]에 대한 설명

- **sizeof(G_Record *)**는 G_Record data type 을 가리키는 포인터의 크기 즉 4bytes(pointer 는 무조건 4bytes)
- **malloc(sizeof(G_Record *) * 10)** : pointer 변수 10 개를 array 로 할당 즉 40bytes
- **(G_Record **)** malloc(sizeof(G_Record *) * 10) : 40bytes(pointer 10 개의 array)할당 받은 곳을 가리키는 주소값을 담고 있는 m_pphead 는 **(G_Record **) data type 임을 "cast 연산자"로 data type 을 변화시켜준다.**
- cast 연산자 : data type 을 변화 시킨다. [사용법 : **(data_type)** variable]

Ex) int a ;
char b =5;
a = **(int)** b; // or a = b; b 가담고있는 값을 int 형으로 변화	

Char * p;
P = **(char *)**malloc(100); // 100byte 할당받은곳이 char data type
// char 형 data Array 를 가리키는 pointer p	

2.8.3.2. 1 차원 포인터 초기화&데이터 초기화



2.9.함수 포인터(Function Pointer)와 Macro function

```
void main()
{


	// [3-1]. 포인터 함수
	// c function Pointer
	m_pFunc = test;
	m_pFunc(Test->a);

	// [3-2]. C++ Function Pointer
	Test->FuncPoint(Test->a);

	// [3-2]. macro 함수
	macro_func(Test->a);Void test(char *p);



}
```

2.9.1 함수 포인터(Function Pointer)

2.9.1.1.C++ FuncPoint()

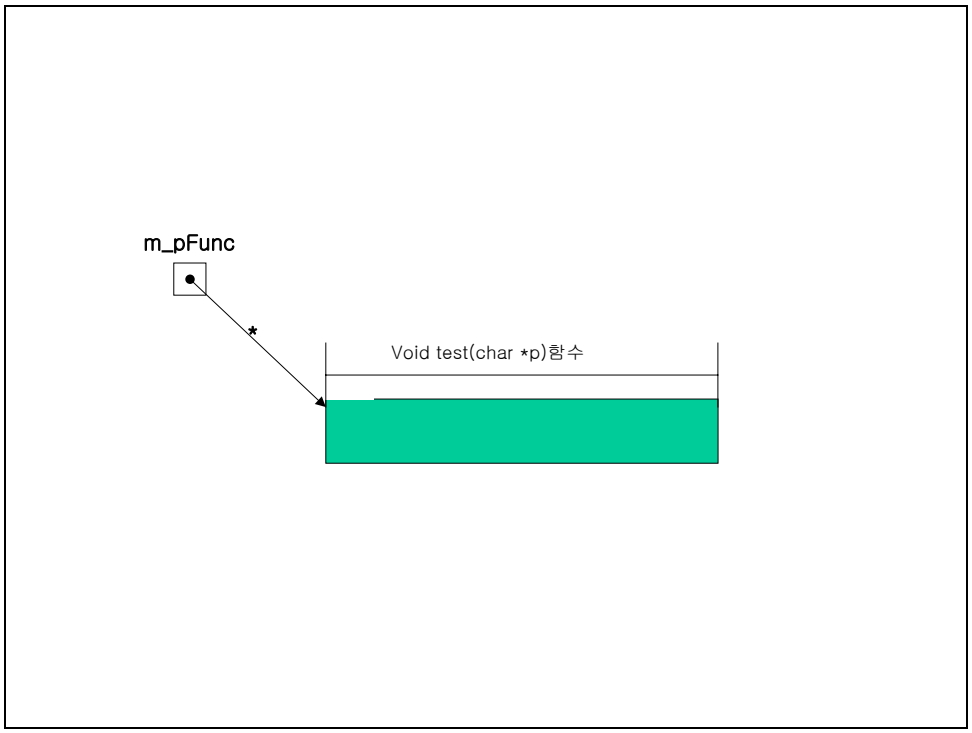
: class 멤버함수를 함수포인터로 넘기지 못하게 되어있는 것 같다.

```
void Pointer::FuncPoint()
{
	// [1]. class member 간 함수 포인터 안됨
	// pFunc = func1;

	// [2].class member 를 c 함수 포인터로 못넘김.
	// pFunc1 = func1;
}
```

2.9.1.2.C 함수포인터

2.9.1.2.1.예제 1

```
int (*m_pFunc)(char *p);

int test(char *p)
{
	cout << "function pointer test :" << p <<"\n";
	return 0;
}

void main()
{
	Pointer *Test = new Pointer("Test");

	// [3-1]. 포인터 함수
	m_pFunc = test;
	m_pFunc(Test->a);

}
```



2.9.1.2.1.예제 2(Q&A 에 오른 예제)

Bubble sort 를 이용하여 ascending/descending

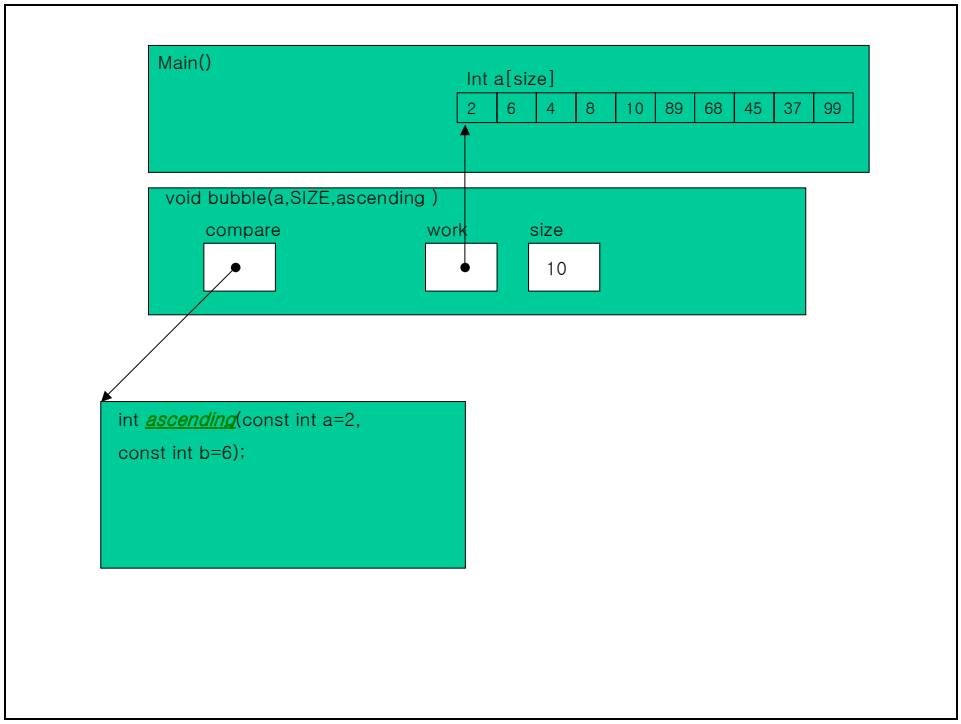
2.9.1.2.1.1.소스

```
//긁어서 컴파일 해보쇼...

#include<stdio.h>
#define SIZE 10

void swap(int *, int *);
void bubble (int *, const int, int (*compare) (const int,const int));
int ascending(const int, const int);
int descending(const int, const int);

main()
{
	int a[SIZE]={2, 6, 4, 8, 10, 89, 68, 45,37,99};
	int counter;

	bubble(a, SIZE, ascending);
	printf("\n 오름차순으로 정렬된 데이터\n");

	for(counter=0; counter<=SIZE-1;counter++)
		printf("\n%4d", a[counter]);

	bubble(a, SIZE, descending);
	printf("\n 내림차순으로 정렬된 데이터 \n");

	for(counter=0; counter<=SIZE-1;counter++)
		printf("\n%4d", a[counter]);
```

```
	printf("\ntest");

	return 0;
}

void bubble(int *work, const int size, int(*compare)(int, int))
{
	int pass, count;

	for(pass=0;pass<size-1;pass++) //  여기도
		for(count=0; count<size-1; count++) //  여기도

			// 조건에 따른 ascending & descending
			if(compare(work[count], work[count+1]))
				swap(&work[count], &work[count+1]);
}

void swap(int	*element1ptr, int	*element2ptr)
{
	int temp;
	temp=*element1ptr;
	*element1ptr=*element2ptr;
	*element2ptr=temp;
}


// ascending(오름차순) : 1,2,3,4,56
int ascending(const int a, const int b)
{
	return a>b; //  여기
}

// descending(내림차순) : 56,4,3,2,1
int descending(const int a, const int b)
{
	return b>a; //  여기
}
```

2.9.1.2.1.2. bubble(a, SIZE, *ascending*);분석

1.초기부분

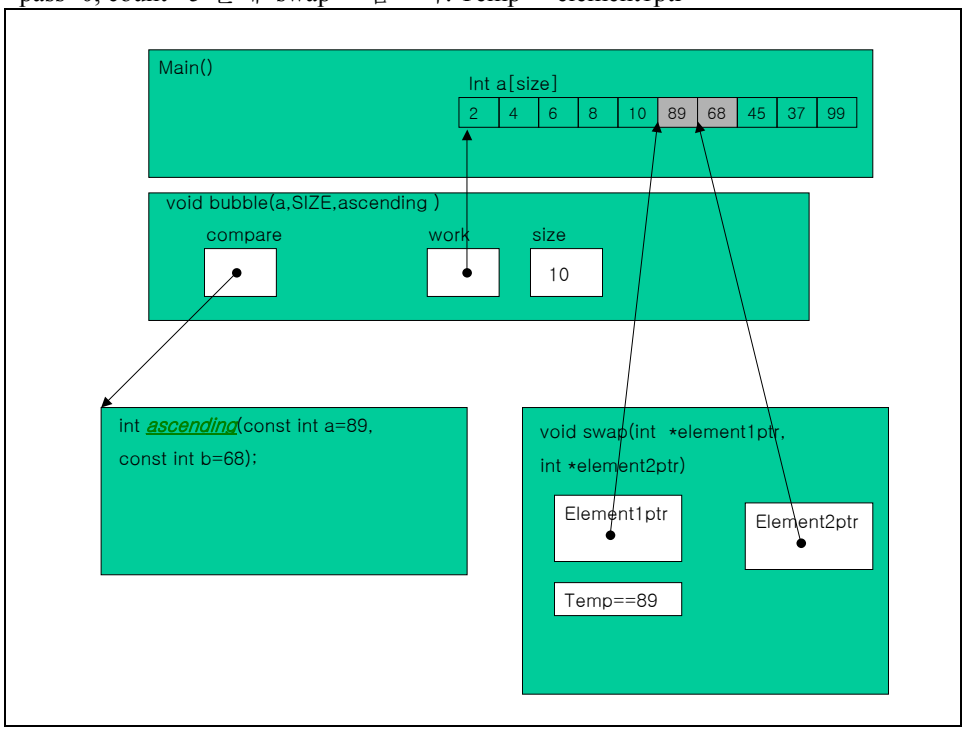


2. state:

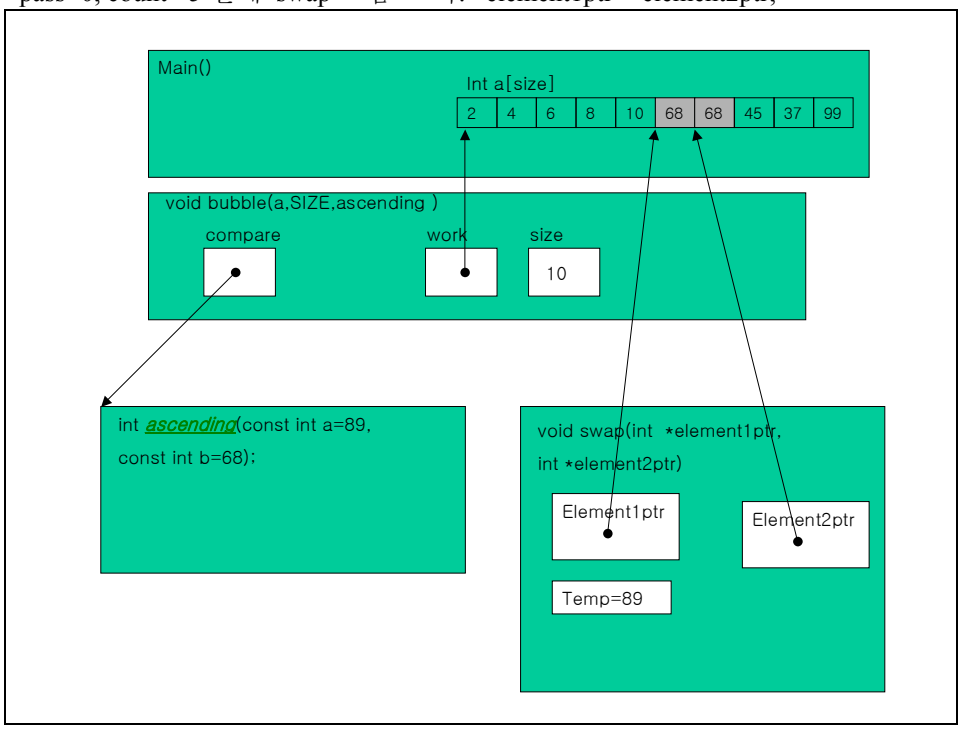
	같은 행에서 분석순서 =>				
pass	count	work[count]	work[count+1]	**compare**(work[count], work[count+1])	swap(&work[count], &work[count+1])
0	0	2	6	ascending(2,6); 0(FALSE)	int a[10] **2** **6** 4 8 10 89 68 45 37 99 => **2** **6** 4 8 10 89 68 45 37 99 (X)
	1	6	4	ascending(6,4); 1(TRUE)	Int a[10]결과 2 **6** **4** 8 10 89 68 45 37 99 => 2 **4** **6** 8 10 89 68 45 37 99 (X)
	2	6	8	ascending(6,8); 0(FALSE)	int a[10] 2 4 **6** **8** 10 89 68 45 37 99 => 2 4 **6** **8** 10 89 68 45 37 99 (X)
	3	8	10	ascending(8,10); 0(FALSE)	int a[10] 2 4 6 8 10 89 68 45 37 99 => 2 4 6 8 10 89 68 45 37 99 (X)
	4	10	89	ascending(10,89); 0(FALSE)	int a[10] 2 4 6 8 10 89 68 45 37 99 => 2 4 6 8 10 89 68 45 37 99 (X)
	5	89	68	ascending(89,68); 1(TRUE)	int a[10] 2 4 6 8 10 89 68 45 37 99 => 2 4 6 8 10 68 89 45 37 99 (X)
	6	89	45	ascending(89,45); 1(TRUE)	int a[10] 2 4 6 8 10 68 89 45 37 99 => 2 4 6 8 10 68 45 89 37 99 (X)
	7	89	37	ascending(89,37); 1(TRUE)	int a[10] 2 4 6 8 10 68 45 89 37 99 => 2 4 6 8 1 6 4 3 8 9

					0 8 5 7 9 9 (X)
	8	89	99	ascending(89,99); 0(FALSE)	int a[10] 2 4 6 8 10 68 45 37 89 99 => 2 4 6 8 10 68 45 37 89 99 (X)

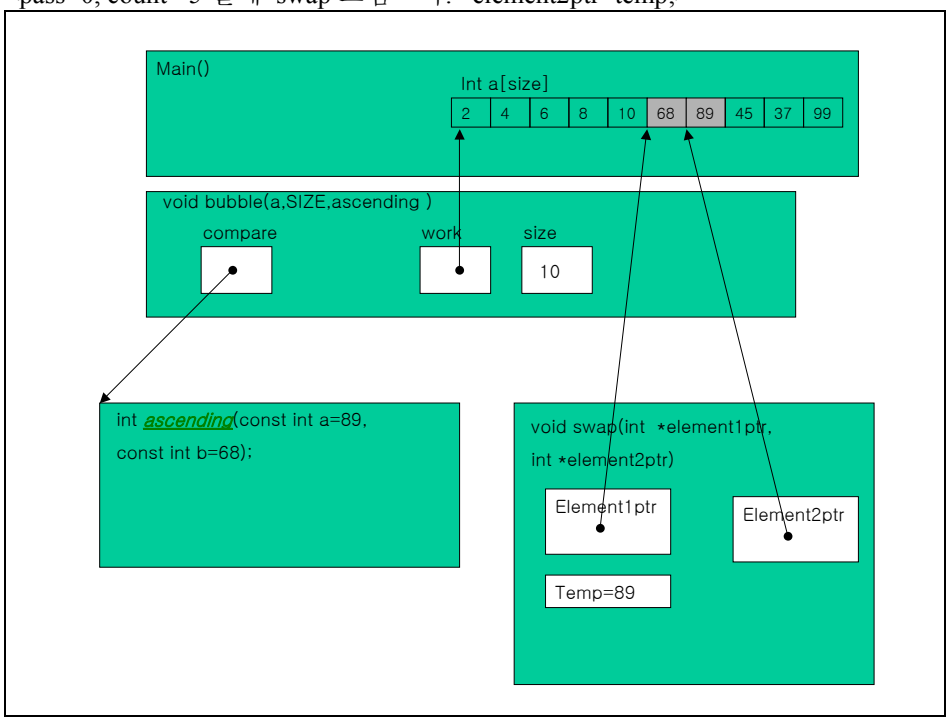
<pass=0, count =5 일 때 swap 그림 가. Temp= *element1ptr>



<pass=0, count =5 일때 swap 그림 나. *element1ptr=*element2ptr;>



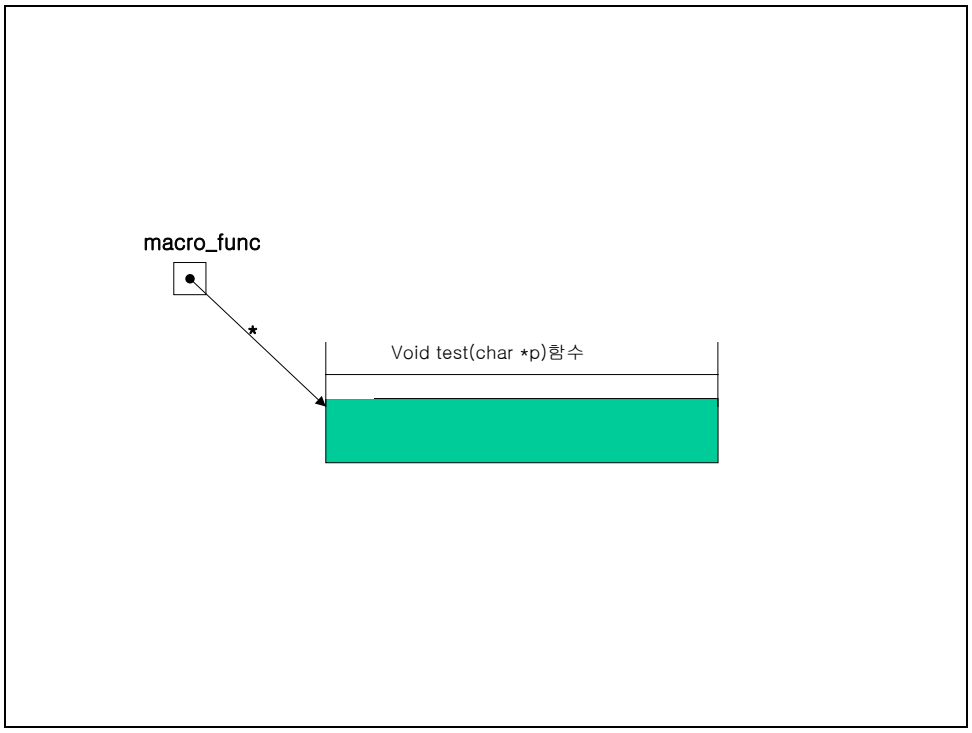
<pass=0, count =5 일때 swap 그림 다. *element2ptr=temp;>



2.9.2. Macro function

2.9.2.1. 예제 1:

```
// [5-2]. C 상태에서 함수 포인터 사용 가능
int test(char *p)
{
	cout << "function pointer test :" << p <<"\n";
	return 0;
}

#define macro_func test

void main()
{
	// [3-2]. macro 함수
	macro_func(Test->a);
}
```

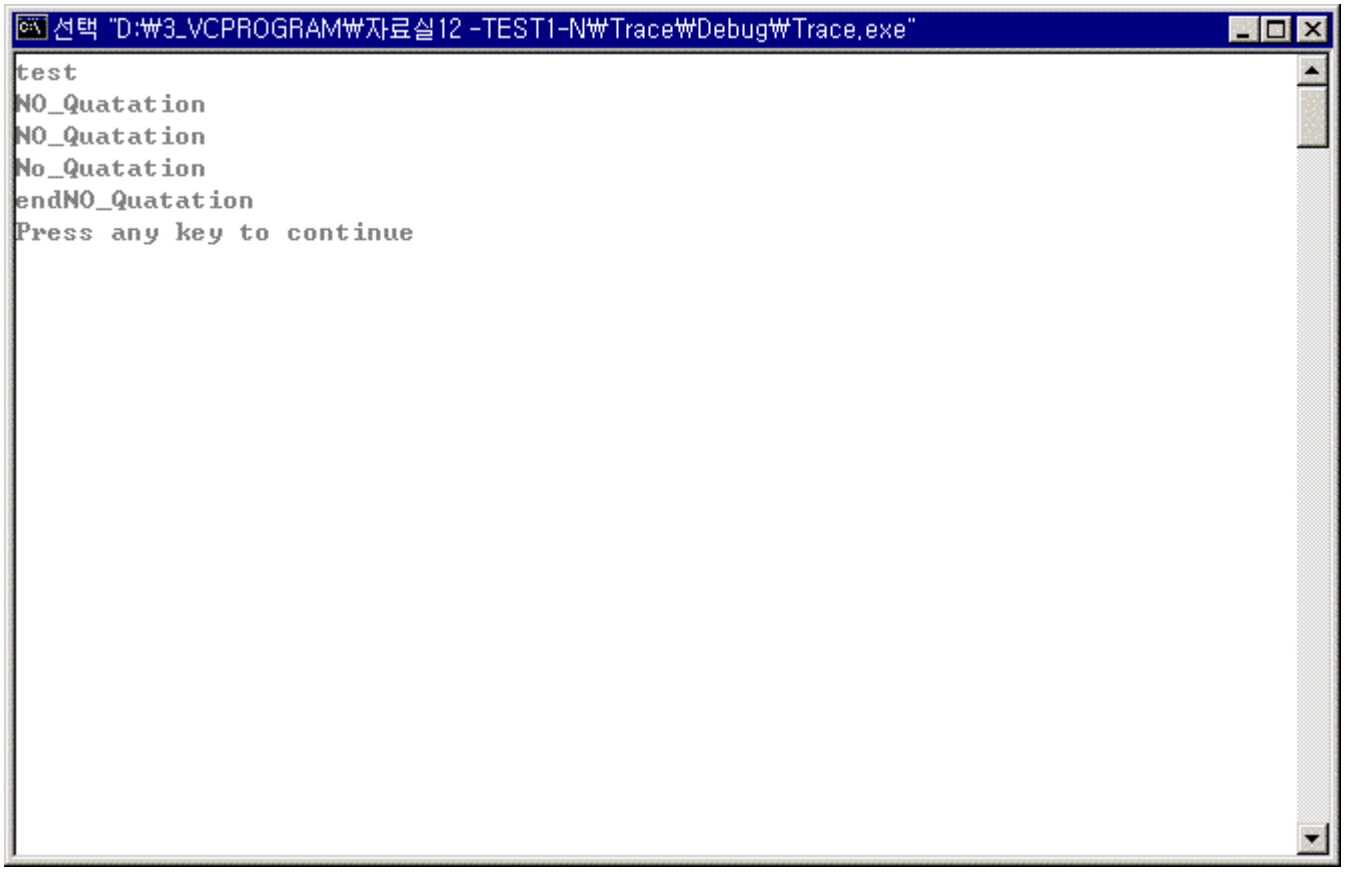


2.9.2.2.Macro Function 예제 2:#문자열 만들기(Define 문서에 기록)

```
#include <stdio.h>
#include <string.h>

// [1]. How to Printf, Using Macro Function
// 동일한 기능을 하는 3 종류의 TRACE()함수
#define TEST_3
#ifdef TEST_1
#define Trace(fmt,args) \
{ \
	int argp[] = { args }; \
	char temp[500]; \
	sprintf(temp,fmt, argp);\
	printf(temp);\
}

#elif TEST_2
#define Trace(fmt,args)\
{\
	char temp[500]; \
	sprintf(temp,fmt, args);\
	printf(temp);\
}
#endif

#ifdef TEST_3
#define Trace(fmt,args)\
{\
	printf(fmt, args);\
}
#endif

// [2]. Printf Parameter
/*********************************************
#Parameter means
MakeString(NO_Quatation) == "NO_Quatation"
*********************************************/
#define MakeString(Parameter)	(#Parameter)
#define PrintString(Parameter)	printf(#Parameter "\n")

/*********************************************
#Parameter means
Copy_String("NO_Quatatioin") == "NO_Quatation"
*********************************************/
// MakeString == Copy_String
#define Copy_String(Parameter)		(##Parameter)
#define Print_String(Parameter)		printf(##Parameter "\n")

// [3]. Make Character
/*****************************************
causes the statement
a = makechar(b);
to be expanded to
a = 'b';
*****************************************/
```

```
#define makechar(x)    #@x

void main()
{

	char stemp[500];
	char test;

	// [2-1]. Test 1
	strcpy(stemp, "test");

	Trace("%s\n",stemp);

	// [2-2]. Test 2
	strcpy(stemp, MakeString(NO_Quatation));
	Trace("%s\n",stemp);

	PrintString(NO_Quatation);

	// [2-3].
	strcpy(stemp, Copy_String("No_Quatation"));
	Trace("%s\n",stemp);

	PrintString(NO_Quatation);

	test = makechar(t); // test = 't'
}
```

=>실행 결과



2.9.3. String Parsing(C 함수포인터와 Macro Function 이용)

Maco Function(컴파일시 해당 소스로 변환 삽입된다)과 pointer 이용한 Function 의 차이점.

2.9.3.1.소스

```
#define MAIN1 // test1()을 하려면 MAIN1, test2()를 할려면 MAIN2
// [1]. I N C L U D E
#include <stdio.h>

// [2]. D E F I N E     C O N S T
#define MAX_LINE_LENGTH              256

// [3]. M A C R O       F U N C T I O N
// [3-1]. Is a value type(digit/hexa/EOL)?
/*******************************************************
1. 기능 : 변수 x 의 값이 문자로 아라비아 숫자('0' - '9')인가?
2. 필요한 데이타:
3. 필요한 함수 :
4. input
      1) x : 문자
5. output
      1) 1 : 아라비아 숫자
      2) 0 : 아라비아 숫자가 아닌경우.
6. algorithm
7. Module Test Example
*******************************************************/
#define IS_DIGIT(x)((x>='0')&&(x<='9'))
#define IS_HEXA(x)    (((x>='0')&&(x<='9'))||((x>='a')&&(x<='f'))||((x>='A')&&(x<='F')))
#define IS_ALPHA(x) (((x>='a')&&(x<='z'))||((x>='A')&&(x<='Z')))
#define IS_CHAR    IS_ALPHA

/*******************************************************
1. 기능 : 변수의 두 값이 EOL(end of Line: '\n', '^M'== 13, NULL== '\0'== 0)인가?
2. 필요한 데이타:
3. 필요한 함수 :
4. input
      1) x : 문자
5. output
      1) 1 : End of Line
      2) 0 :
6. algorithm

7. Module Test Example
*******************************************************/
#define IS_EOL( c, d)    ( c=='\n'|| c==0 || (c==13 && d=='\n'))

/* is Null ? 0 == '\0' */
#define IS_NULL(c)          (c==0)

// [3-2]. Get Line(string)
/*******************************************************
1. 기능 : 문자열(file)로부터 한 라인의 크기와 라인 문자열을 얻는다.
참고)라인의 구분자(delimeter) '\n', NULL, ^M'== 13
2. 필요한 데이타:
3. 필요한 함수 :
```

```
4. input
1) line    : source(hay_stack)에서 처음 발견한 line 을 저장할곳.
2) line_length : line length(단위 byte)
   즉) line[*index]
3) source : hay_stack(건초더미,여러라인이 들어있음)문자열을 가리키는 pointer
4) source_length :source 의 문자열 크기(단위 byte)

5. output
6. algorithm
7. Module Test Example
*****************************************************/
void    GET_LINE(char *line,int *line_length,char *source,int source_length)
{
      int j=0;
      *line_length=0;

      // EOL 문자가 아니고 && source_length 이하일동안 line 에 문자를 삽입
      while( !IS_EOL(source[*line_length],source[(*line_length)+1])
                  &&(*line_length < source_length)&&( (*line_length) < MAX_LINE_LENGTH))
      {
            line[j++] = source[(*line_length)++];
      }

      line[j] = 0; // 문자열 끝에 NULL 을 삽입한다.

      // 수정 요망....
      if(source[*line_length]=='\n') //"\n"
            (*line_length)++;
      else
            (*line_length)+=2; // "^M\n"
}

// [3-3]. Parsing
// [3-3-1].
// is a skip character?
#define IS_SKIP(x) (x==' '||x=='\t')

#define IS_DELI(x)    (x=='/'||x==' '||x==':'||x==','||x==';'||x=='='||x=='('||x==')'|| x=='\t' )

// [3-3-2]. Get a Digit Token from a Line(string)
/*****************************************
1. 기능 : line 이라는 문자열에서 Delimeter(숫자외의 문자)를 구분자로 하여 "숫자 토큰"을 얻는 함수.
 단) space 와 '\t'==tab 문자는 토큰으로 설정하지 않는다.
 단) 문자열의 특정위치에 디지트가 있다는 것을 알고 있을때

2. 필요한 데이타:
3. 필요한 함수 :
1) IS_SKIP()
2) IS_DIGIT()
4. input
      1) line    : 문자열 시작 포인터(hay_stack)
      2) offset :  문자열 시작 포인터의 어레이 넘버.ex) line[0]..... line[index]
      3) token :  발견한 token( a needle)
      4) token_length : token 의 길이.( needle Size)
5. output
      SUCCESS : get a digit token
      FAILURE : don't get....
1) offset :
```

```
      - line 의 어레이에서 발견한 토큰의 문자열이 끝난곳 다음 1 바이트를 가리킨다.
      -목적: 새로운 토큰(needle)을 찾는 위치
ex) line = "hey    a    babo" => token = "hey", index ="hey"다음의   "space 문자" 위치(h 가   어레이번호로 0 이라면
              index 값은 3)
6. algorithm
7. Module Test Example
*****************************************/
#define TRUE          1
#define FALSE         0
#define SUCCESS    TRUE
#define FAILURE    -1
int GET_DIGIT(char *line,int *offset,char *token,int token_len)
{
      token_len=0;

      // [2-1]. space, '\t'내용은 skip
      for( ; IS_SKIP(line[*offset]) ; (*offset)++)
            ;

      // [2-2]. get digit value from [char* input]
      //    delimiter 는 아라비아 숫자.
      for(;IS_DIGIT(line[*offset]);(*offset)++)
            token[token_len++]=line[*offset];

      token[token_len] = NULL; //문자열 끝에 NULL

      // [2-3].
      if( token_len == 0)
            return FAILURE;
      else
            return SUCCESS;
}

// [3-3-3]. Get a Token( all type) from a line(string)
/*****************************************************
Get Token
1. 기능 : line 이라는 문자열에서 Delimeter 를 구분자로 하여 토큰을 얻는 함수.
 단) space 와 '\t'==tab 문자는 토큰으로 설정하지 않는다.
2. 필요한 데이타:
3. 필요한 함수 :
1) IS_DELI()
2) IS_EOL()
4. input
1) line :  문자열 시작 포인터.
2) offset: token 을 얻을 문자열 시작 포인터의 어레이 넘버.ex) line[0]..... line[offset]
3) token :  발견한 token
4) token_length : token 의 길이.

5. output
1) offset : line 의 어레이에서 발견한 토큰의 문자열이 끝난곳 다음 1 바이트를 가리킨다.
          - 목적: 새로운 토큰(needle)을 찾는 위치
ex) line = "hey    a    babo" => token = "hey", index ="hey"다음   "space 문자" 위치(h 가
       어레이번호로 0 이라면 index 값은 3)
6. algorithm

7. Module Test Example
*****************************************************/
void    GET_TOKEN(char *line,int* offset,char *token,int *token_length)
```

```
{
	*token_length=0;

	// [2-1]. space 나 tab 키는  skip 한다.
	for(; IS_SKIP(line[*offset])  ;(*offset)++)
		;

	// [2-2].Get Token while a character is not delimiter.
	do
	{
		token[(*token_length)++]=line[(*offset)++];

		if(IS_DELI(token[(*token_length)-1]))
			break;

	}while(!IS_DELI(line[*offset])&&!IS_EOL(line[*offset],line[(*offset)+1]));

	// [2-3]. token 끝에 NULL 문자 삽입
	token[*token_length]=0;
}

// Define_Get_Token
#define D_GET_TOKEN(line,offset,token,token_length)\
{\
	token_length=0;\
	for(;IS_SKIP(line[offset]);(offset)++)\
		;\
	do{\
		token[token_length++]=line[(offset)++];\
		if(IS_DELI(token[token_length-1]))\
			break;\
	}while(!IS_DELI(line[offset])&&!IS_EOL(line[offset],line[(offset)+1]));\
	token[token_length]=0;\
}

// [3-3-4]. Is a token a digit token(수) or a variable/function?
//  token 의 종류?
#define DIGIT_STRING		TRUE
#define NO_DIGIT_STRING	-1
/*****************************************************
1. 기  능 : 상수?인가?  변수 인가?
2. 필요한 데이타:
3. 필요한 함수 :
	1) IS_DIGIT()
4. input
1) token : 발견한 token

5. output
	- DIGIT_STRING :숫자로만 구성되어 있다.
	- NO_DIGIT_STRING

6. algorithm

7. Module Test Example
*****************************************************/
int IS_DIGIT_STRING(char *token)
{
	// [1]. Data
	int token_length;
```

```
	int loop;

	// [2].
	// [2-1].
	token_length = strlen(token);

	for(loop =0; loop< token_length ;loop++)
	{
		if(!IS_DIGIT( token[loop]))
			return NO_DIGIT_STRING;
	}

	return DIGIT_STRING;
}

/*******************************************************
1. 기   능 : Hexa or Decimal Digit?
2. 필요한 데이타:
3. 필요한 함수 :
	1) IS_HEXA()
4. input
1) token : 발견한 token

5. output
	- HEXA_DIGIT_STRING :숫자로만 구성되어 있다.
	- NO_HEXA_DIGIT_STRING

6. algorithm

7. Module Test Example
ex) token = "0x1234" or "0X1234"
*******************************************************/
#define HEXA_DIGIT_STRING		TRUE
#define NO_HEXA_DIGIT_STRING	-1
int IS_HEXA_STRING(char *token)
{
	// [1]. Data
	int token_length;
	int loop;

	// [2].
	token_length = strlen(token);

	// [2-1]. token[0]= '0' token[1] = 'x'?
	if( token[0] != '0' || (token[1] != 'x' && token[1] != 'X') )
		return NO_HEXA_DIGIT_STRING;

	// [2-2]. token [2, ....] is digits?
	for(loop =2; loop< token_length ;loop++)
	{
		if(!IS_HEXA( token[loop]))
			return NO_HEXA_DIGIT_STRING;
	}

	return HEXA_DIGIT_STRING;
}


#ifdef MAIN
void main()
```

```
#else
void test()
#endif
{
	// [1]. Data
	int	index =0;
	int	num_of_token =0;
	char token[50];
	int	token_length =0;

	char message[MAX_LINE_LENGTH] = "0x1234 int\tCString::GetString(char a)";

	// [2-1]. line 단위  파싱.
	while(message[index] != NULL)
	{
		// [2-1-1]. line 내에서  토큰을  얻는다.
		GET_TOKEN(message,&index,token,&token_length);
		printf("\nnum[%d] : token[%s]is a %s, message[index(%d)] token_length(%d)\n",
				num_of_token++,token,IS_HEXA_STRING(token)==HEXA_DIGIT_STRING?"hexa":"no" ,
				index,token_length);
	}



}
#ifdef MAIN2
void main()
#else
void test2()
#endif
{

	// [1]. Data
	int	temp =0;
	int	index =0;

	char token[50];
	int	token_length =0;

	char message[MAX_LINE_LENGTH] = "int\tCString::GetString(char a)";

	// [2-1]. line 단위  파싱.
	while(message[index] !=NULL)
	{
		D_GET_TOKEN(message,index,token,token_length);
		printf("\ntoken(%s) index(%d)    token_length(%d)\n",token, index, token_length);

	}
}
```

2.9.3.2.실행결과

가. test1()실행결과

-주어진 string : "0x1234 int\tCString::GetString(char a)"

```
num[0] : token[0x1234]is a hexa, message[index(6)] token_length(6)
num[1] : token[int]is a no, message[index(10)] token_length(3)
num[2] : token[CString]is a no, message[index(18)] token_length(7)
num[3] : token[:]is a no, message[index(19)] token_length(1)
num[4] : token[:]is a no, message[index(20)] token_length(1)
num[5] : token[GetString]is a no, message[index(29)] token_length(9)
num[6] : token[(]is a no, message[index(30)] token_length(1)
num[7] : token[char]is a no, message[index(34)] token_length(4)
num[8] : token[a]is a no, message[index(36)] token_length(1)
num[9] : token[]]is a no, message[index(37)] token_length(1)
Press any key to continue
```

나. test2()실행결과

-주어진 string : = "int\tCString::GetString(char a)";

```
token(int) index(3)    token_length(3)
token(CString) index(11)    token_length(7)
token(:) index(12)    token_length(1)
token(:) index(13)    token_length(1)
token(GetString) index(22)    token_length(9)
token(() index(23)    token_length(1)
token(char) index(27)    token_length(4)
token(a) index(29)    token_length(1)
token()) index(30)    token_length(1)
Press any key to continue
```

2.9.4. MFC DLL Function 호출(Function Pointer)

추후 예제 보여줌니다.

3.String 계열 함수 분석(신규추가)

3.1. StrLen(), strlen()

3.1.1.정의

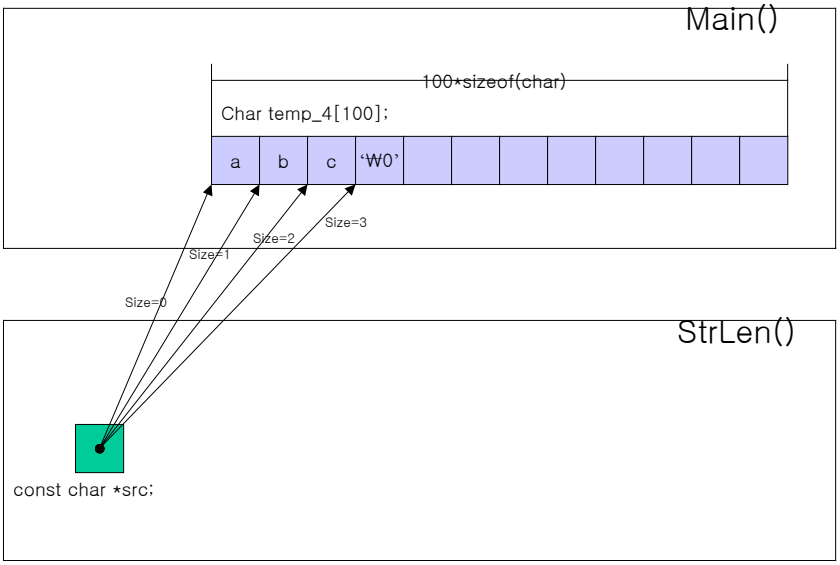
함수	
int StrLen(const char *src);	**1.기능** : NULL== 0== "\0"문자를 만날 때까지의 문자열의 크기를 구한다. **2.필요데이타** **3.필요한 함수.** **4. parameter** 1) const char *src : 문자열을 가리키는 pointer **5. return**: 문자열의 크기를 return 한다. **6. algorithm** **7.참고** Fast Strlen()방식 : 4 바이트 0x00ffffff, 0xff00ffff, 0xffff00ff, 0xffffff00 비트마스크를 이용하는 방식은 리눅스나 유닉스에서는 되지만 윈도우에서는 저장방식이 틀려서 안된다.

3.1.2.소스

```
int StrLen(const char *src)
{
	// [1]. Data
	int size =-1 ;

	// [2]. Algorithm
	// [2-1].
	while(src[++size] !=NULL)
		;

	// [2-2]. return size
	return size;
}
```

3.1.3. Analysis

```
void main()
{
	char temp_4[100]="abc";

	StrLen(temp_4);
}
```



	같은 행에서 분석순서 =>			
	src	size	Src[++size]	while(src[++size] !=NULL)
1 회	== temp_4 == &temp_4[0]	-1	Src[0] == 'a'	while(src[0] !=NULL)
2 회	== &temp_4[1]	0	Src[1] == 'b'	while(src[1] !=NULL)
3 회	== &temp_4[2]	1	Src[2] == 'c'	while(src[2] !=NULL)
4 회	== &temp_4[3]	2	Src[3] == '\0'	while(src[3] !=NULL)

3.2. StrCpy(), strcpy() – 2.6.1.StringCpy()함수 참조

3.3. StrCmp(), strcmp()

3.3.1.정의

함수	
int StrCmp (const char *p1, const char *p2)	**1. 기능 : Compare S1 and S2, returning less than, equal to or greater than zero if S1 is lexicographically less than, equal to or greater than S2.** **2. 필요한 데이타:** **3. 필요한 함수** **4. Parameter :** **5. Output :** **양수 : string p1 > string p2** **0 : string p1 == string p2** **음수 : string p1 < string p2** **6. algorithm** **7.참고 :순차적(선형적)으로 비교하는 알고리즘**

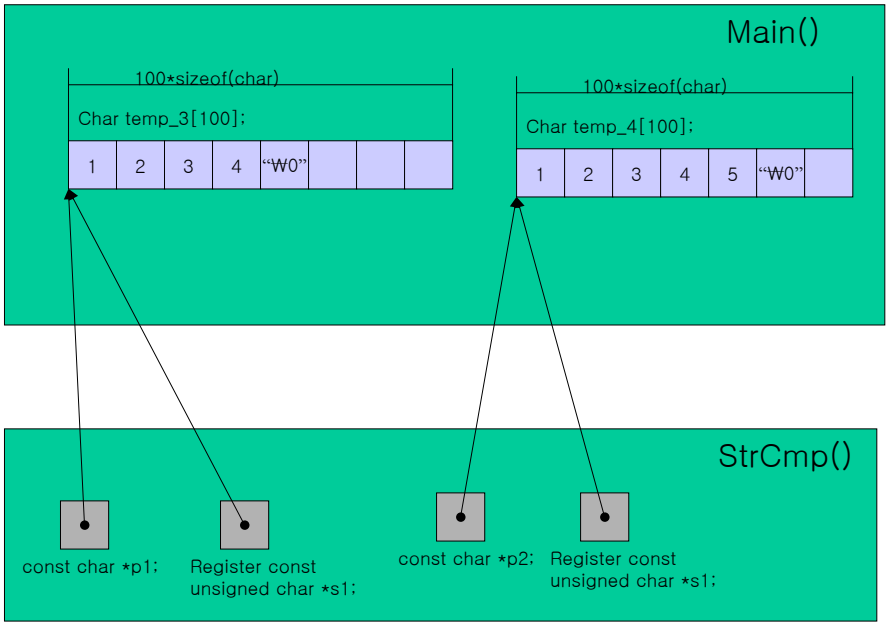
3.3.2.소스

```
int StrCmp (const char *p1, const char *p2)
{
	register const unsigned char *s1 = (const unsigned char *) p1;
	register const unsigned char *s2 = (const unsigned char *) p2;
	unsigned char c1, c2;

	do
	{
		c1 = (unsigned char) *s1++;
		c2 = (unsigned char) *s2++;

		if (c1 == '\0')
			return c1 - c2;
	} while (c1 == c2);

	return c1 - c2;
}
```

3.3.3.Analysis

```
void main()
{
	char temp_1[100]="1234";
	char temp_2[100]="12345";

	StrCmp(temp_1,temp_2);
}
```

3.3.3.1.Main()에서 StrCmp(temp_1, temp_2)의 Parameter 받는 모습



3.3.3.2.Main()에서 StrCmp(temp_1, temp_2)실행과정과 결과

3.3.3.2.1.TRACE 과정

<약어>

- UCHAR : unsigned char

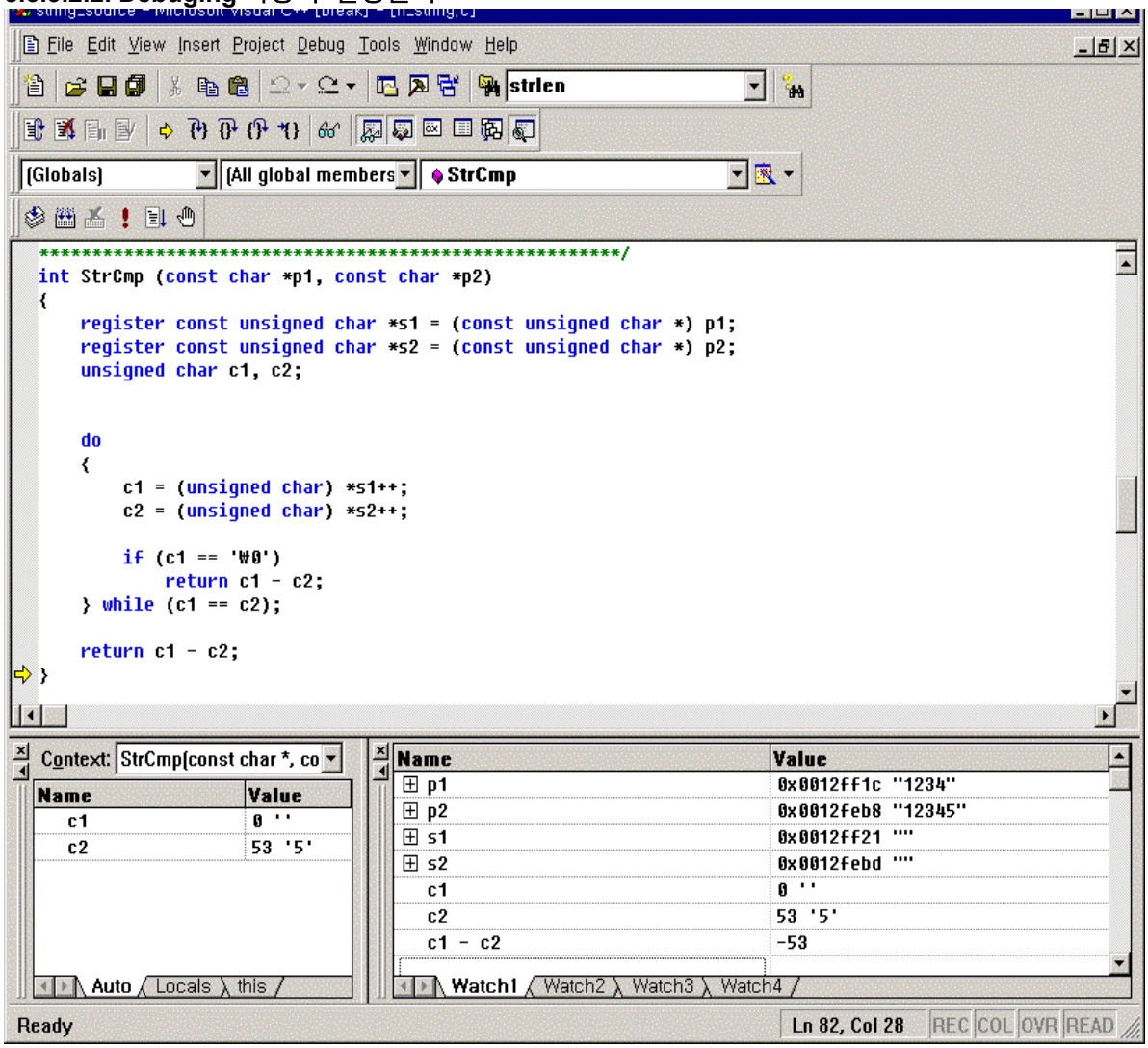
-TRUE : 1

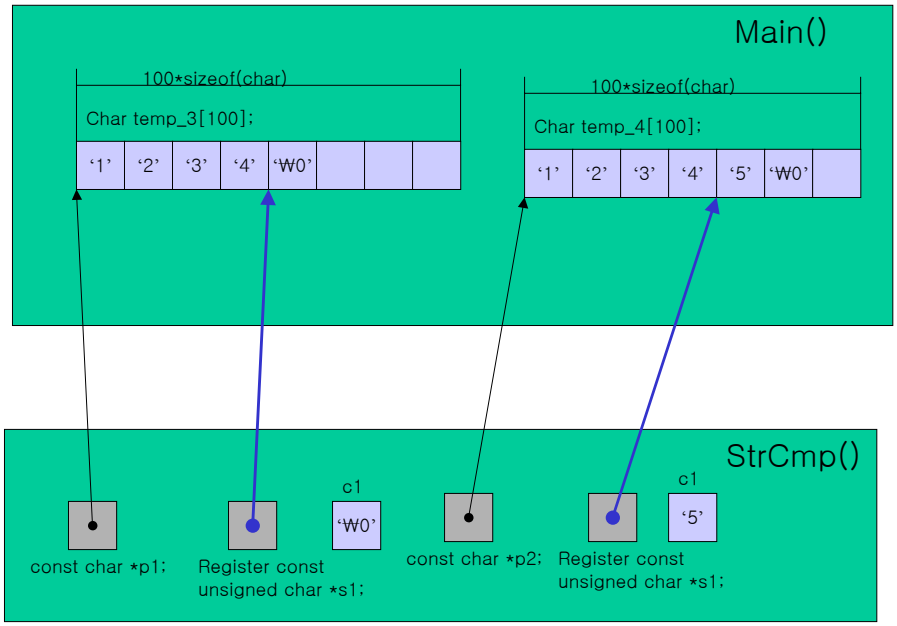
-FALSE : 0

	같은 행에서 분석순서 =>						
			do{				
회수	s1	s2	c1 = (UCHAR) *s1++;	c2 = (UCHAR) *s2++;	if (c1== '\0')	**} while(c1== c2);**	return c1-c2;
1	&temp_3[0]	&temp_4[0]	C1 = '1'	C2 = '1'	FALSE	TRUE	
2	&temp_3[1]	&temp_4[1]	C1 = '2'	C2 = '2'	FALSE	TRUE	
3	&temp_3[2]	&temp_4[2	C1 = '3'	C2 = '3'	FALSE	TRUE	

		]					
4	&temp_3[3]	&temp_4[3]	C1 = ‘4’	C2 = ‘4’	FALSE	TRUE	
5	&temp_3[4]	&temp_4[4]	C1 = ‘\0’	C2 = ‘5’	TRUE	FALSE	-53 == (‘5’의 음수)

3.3.3.2.2. Debuging 과정과 실행결과




Main()
100*sizeof(char)
Char temp_3[100];
'1' '2' '3' '4' '₩0'
100*sizeof(char)
Char temp_4[100];
'1' '2' '3' '4' '5' '₩0'
StrCmp()
const char *p1;
Register const
unsigned char *s1;
c1
'₩0'
const char *p2;
Register const
unsigned char *s1;
c1
'5'

3.3. StrCat()

3.3.1.정의

함수	
char *StrCat (char *dest, const char *src)	**1.기능** : NULL== 0== "\0"문자를 만나기 전까지 src 문자열을 Dest 문자열 끝에 append(이어서 붙여쓴다). **2.필요데이타** **3.필요한 함수.** **1)StrLen()** **2)StrCpy()** **4. parameter** 1) char *dest : append 될 곳 2) const char *src : **5. return**: Append 후 만들어진 문자열을 가리키는 dest 의 주소값을 return 한다. **6. algorithm** **7.참고**

3.3.2.소스

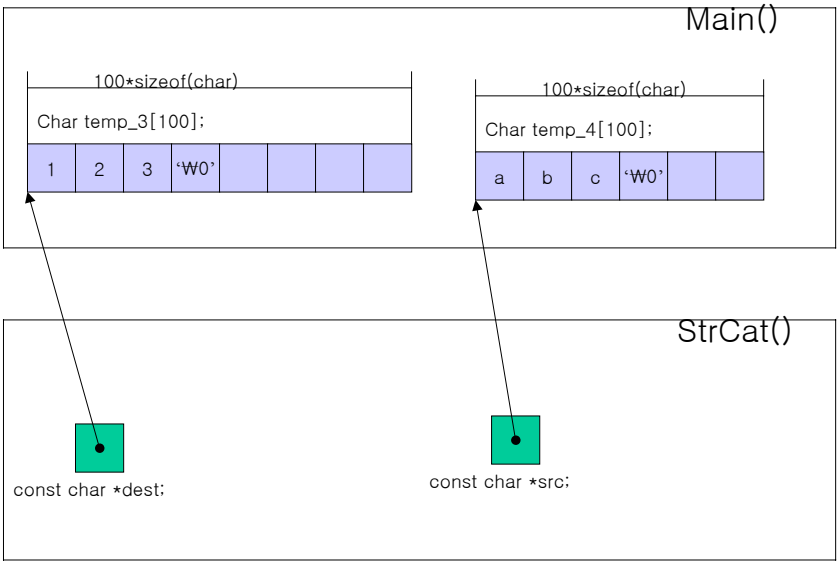
```
/*********************************************************
1. 기능 :  Append SRC on the end of DEST.
2. 필요한 데이타:
3. 필요한 함수
4. Parameter :
5. Output :
*********************************************************/
char *StrCat (char *dest, const char *src)
{
	// [1]. Data
	int dest_size;
	int src_size;
	char* destiny_copy;

	// [2].
	// [2-1]. How to make Case_1
#if  0 // #if 0 하면 #else 사이의 소스는 실행이 안된다.

	// [2-1-1]. Calculates Sizes of dest String.
	dest_size = StrLen(dest);

	// [2-1-2]. append == StrCpy()와 동일한 루틴.
	while(*src != NULL)
	{
		dest[dest_size++] = *src++;

	}

	dest[dest_size] = NULL;
```

```
#else

	// [2-2]. How to make Case_2
	// [2-2-1]. Calculates Sizes of dest String and src String.
	dest_size = StrLen(dest);
	src_size    = StrLen(src);

	// [2-2-2]. append with using strcpy();
	destiny_copy = dest;
	destiny_copy += dest_size;
	StrCpy(destiny_copy, src); // or memcpy(destiny, src,src_size+1);

#endif

	return dest;

}
```

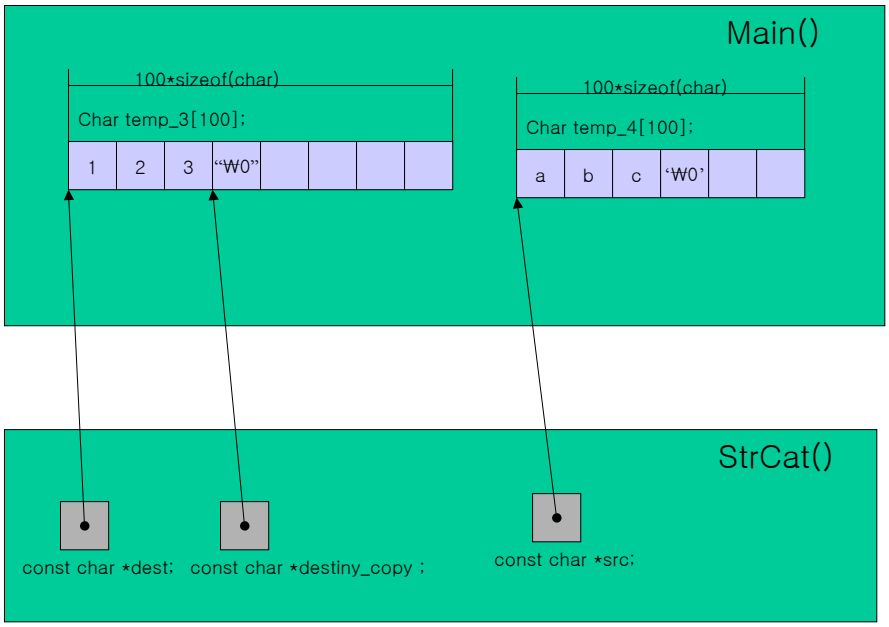
3.3.3.Analysis

```
void main()
{
	char temp_3[100]="123";
	char temp_4[100]="abc";

	StrCat(temp_3,temp_4);
}
```

3.3.3.1. 그림 3.3.3.가 StrCat(temp_3, temp_4);함수의 초기화면



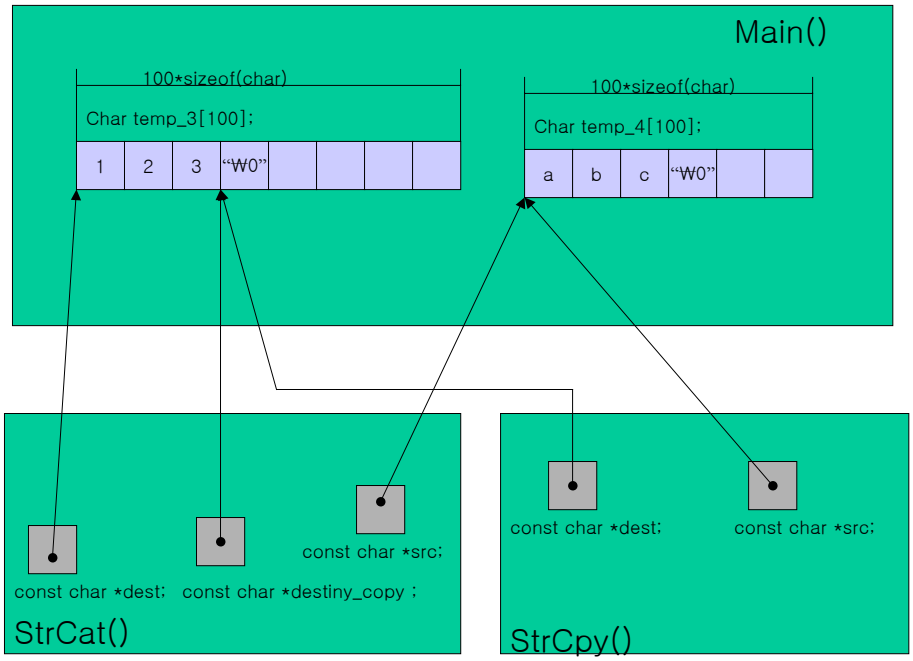
3.3.3.2. StrCat(temp_3, temp_4);함수내의 StrCpy(destiny_copy,src)실행전



< 그림 3.3.3.나.>

3.3.3.3. StrCat(temp_3, temp_4);함수내의 StrCpy(destiny_copy,src) Parameter 받는루틴

2.6.1.StrCpy()함수를 참조 하세요.



<그림 3.3.3.다.>

3.3.3.4.StrCpy 호출되는 루틴

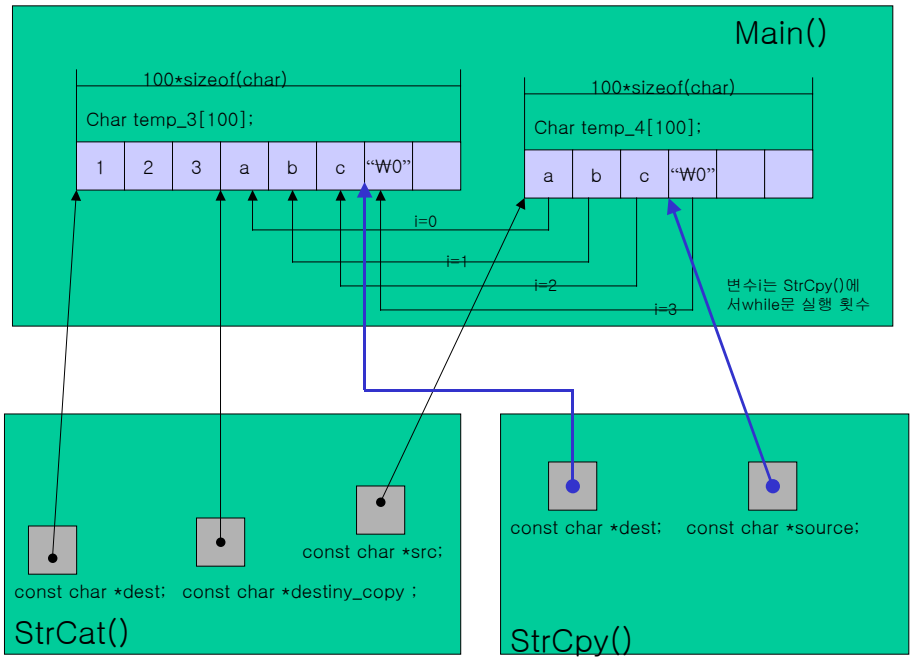
1) StrCpy()내부의 변수 trace
<단 표내 약어 설명>
- dest 와 source : StrCpy()내부변수
- destiny_copy 와 src : StrCat()내부변수입니다.
- Cat_dest : StrCat()내부변수 const char *dest 의 또다른 이름(alias)입니다.
- 회수(i)에서 i 는 방법 2. 배열을 이용한 방법에서 사용된 Local variable int i 이다>

	같은 행에서 분석순서 =>				
회수 (i)	Dest	Source	while((*dest = *source) != '\0')	dest ++ 실행후	source++ 실행후
1(0)	&destiny_copy[0]==&Cat_dest[3];	&src[0]	while((**destiny_copy[0] = source[0]**)!=”\0”)	&destiny_copy[1]	&src[1]
2(1)	&destiny_copy[1]==&Cat_dest[4];	&src[1]	while((**destiny_copy** [1] = source[1])!=”\0”)	&destiny_copy[2]	&src[2]
3(2)	&destiny_copy[2]==&Cat_dest[5];	&src[2]	while((**destiny_copy** [2] = source[2])!=”\0”)	&destiny_copy[3]	&src[3]
4(3)	&destiny_copy[3]==&Cat_dest[6];	&src[3]	while((**destiny_copy** [3] = source[3])!=”\0”)		

2) while**((*dest = *source)** != '\0')의 확장.

	같은 행에서 분석순서 =>	
회수(i)	*source 의 변환된 값	**(*dest = *source)**
1(0)	source[0]==‘a’	**destiny_copy** [0]=source[0]==‘a’
2(1)	source[1]==‘b’	**destiny_copy** [1]=source[1]==‘b’
3(2)	source[2]==‘c’	**destiny_copy** [2]=source[2]==‘c’
4(3)	source[3]==NULL==’\0’	**destiny_copy** [3]=source[3]==NULL==’\0’

3) StrCat(temp_3, temp_4);함수내의 StrCpy(destiny_copy,src)실행후
2.6.1.StrCpy()함수를 참조 하세요.



<그림 3.3.3.라.>

3.4.strtok(2.9.2.2.의 GetToken()함수와 비슷한 방법을사용)

3.5.StrChr()

3.5.1.정의

함수	
char ***StrChr** (const char *m_haystack, const char m_needle)	**1.기능** : phaystack 의 문자열에서 pneedle 문자와 동일한 문자가 시작하는 포인터를 return 한다 **2.필요데이타** **3.필요한 함수.** **4. parameter** 1) const char *o_haystack : Original Haystack 2) const char o_needle : Original needle **5. return**: Haystack(건초더미)에서 m_needle(바늘)이 발견된 곳의 Pointer 를 return 한다. **6. algorithm** **program 의 기본이 되는 Linear Search** **7.참고** **HayStack (건초더미)에서 a Needle(바늘)찾기프로그램**

3.5.2.소스

```
char *StrChr(const char *o_haystack, const char o_needle)
{
	// [1]. Data
	register const char *copy_haystack;

	// [2].
	//[2-1]. check Error
	if( o_haystack == NULL || o_needle == NULL
		|| *o_haystack == NULL )
		return NULL;

	// [2-2]. Linear Search
	copy_haystack = o_haystack;
	while(*copy_haystack++ != o_needle)
		;

	return --copy_haystack;
}
```

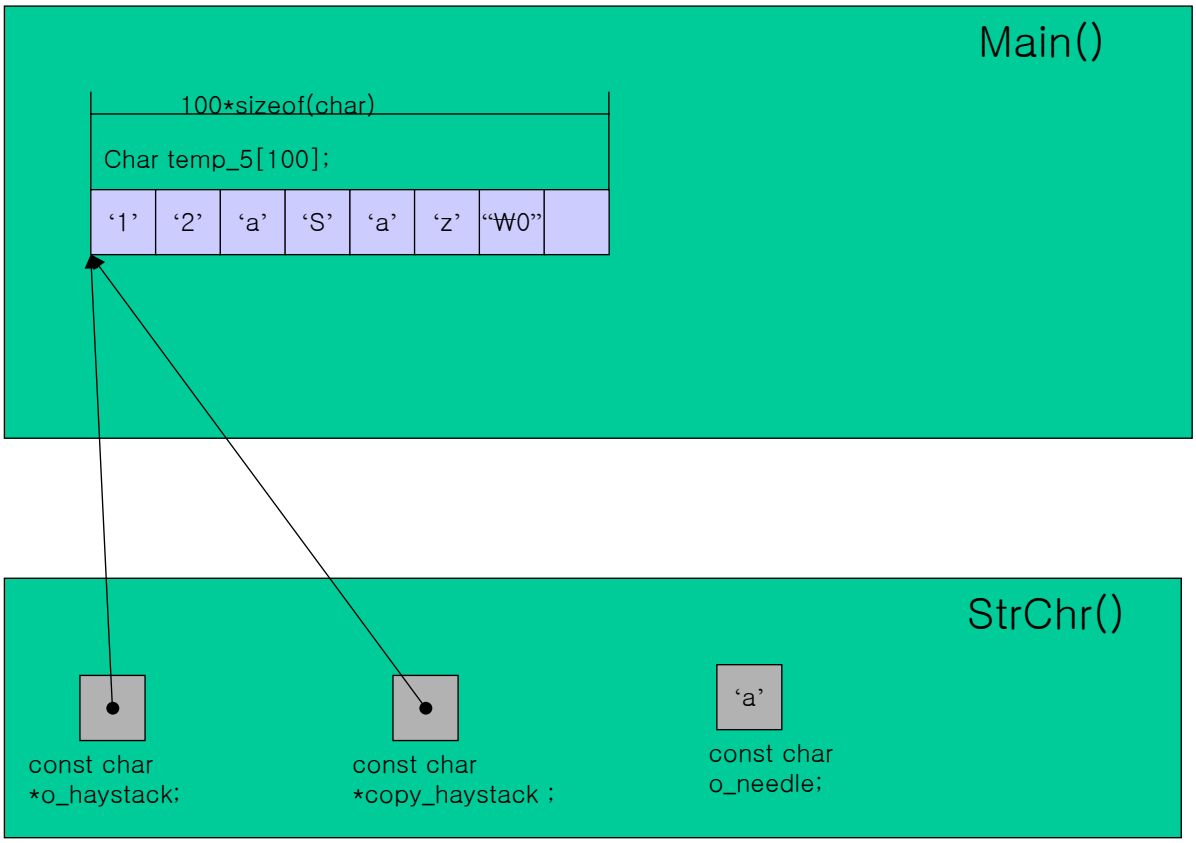
3.5.3.Analysis

-초기그림
-중간그림, Trace
-최종그림

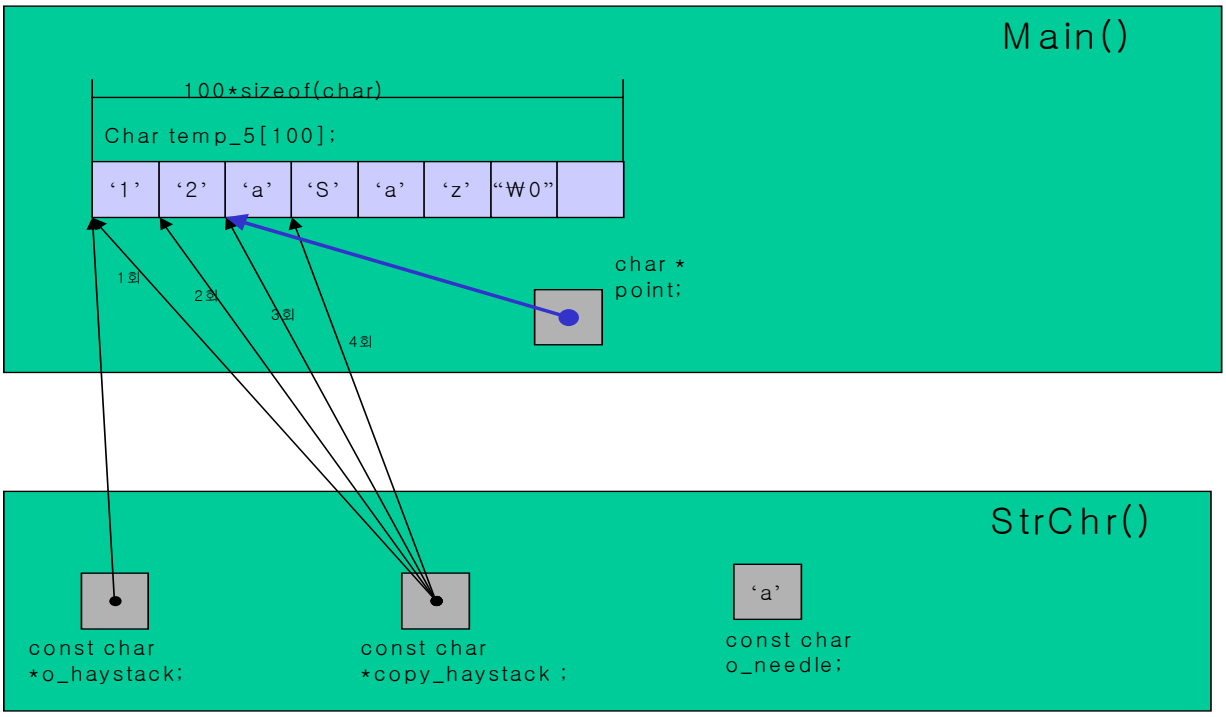
```
void main()
{
	char temp_5[100]="12asaz";
	char temp_7[100]="";
	char *point;

	// [1].
	point =StrChr(temp_5, 'a');

	// [2].
	point =StrChr(point+1, 'a');

	// [3]. error test
	point =StrChr(temp_7, "");

}
```

3.5.3.1. point =StrChr(temp_5, 'a');분석

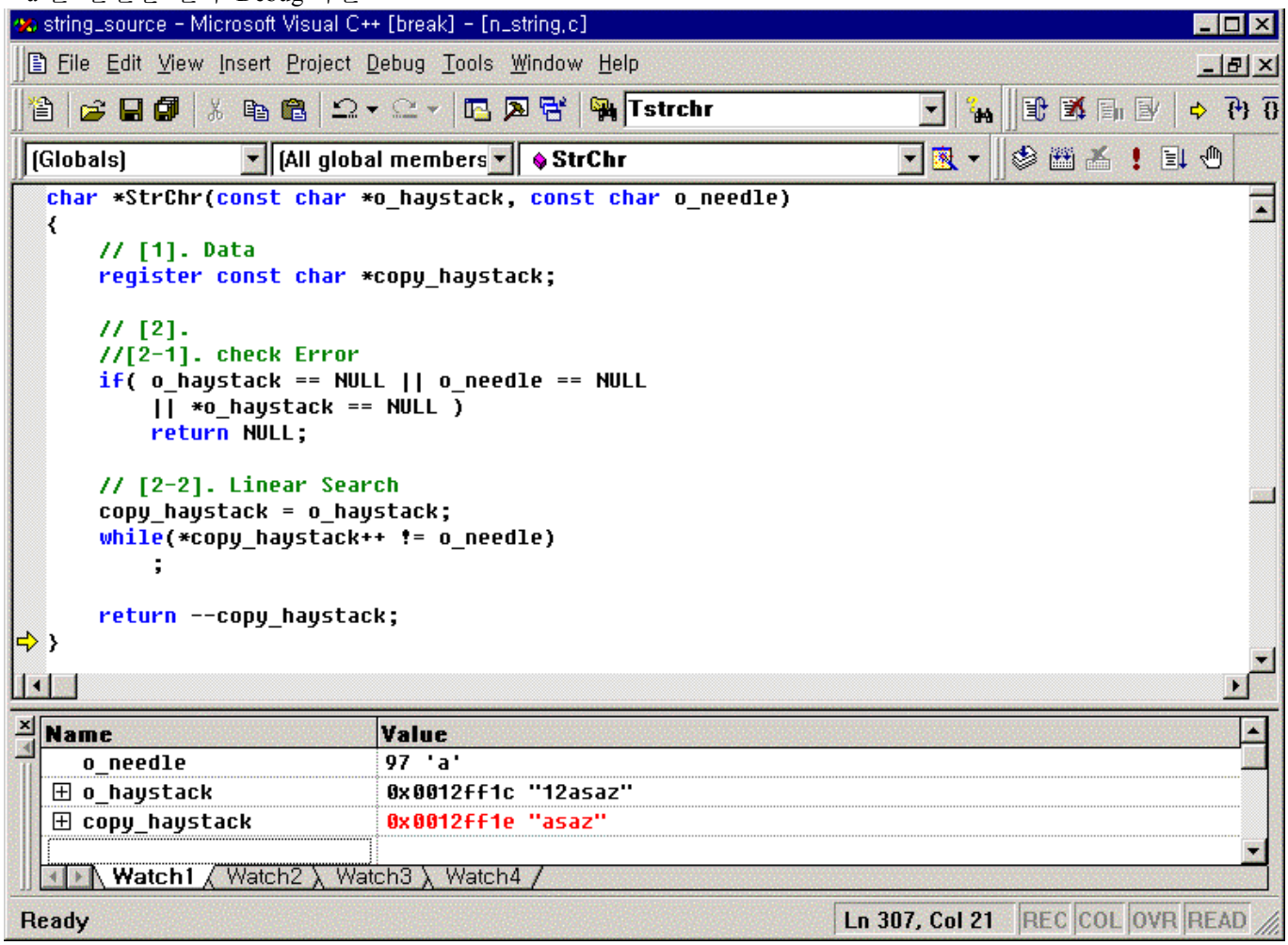
3.5.3.1.1.StrChr()의 초기 Parameter 넘긴상태



3.5.3.1.2.StrChr()를 Trace 한상태



<'a'를 발견한 결과 Debug 화면>



1. StrChr() Trace 한 내용

	같은 행에서 분석순서 =>			
횟수	copy_haystack	while(*copy_haystack++ != o_needle)	copy_haystack 변화	return --copy_haystack;
1	&temp_5[0]	While(temp_5[0] != 'a') TRUE	&temp_5[1]	
2	&temp_5[1]	While(temp_5[1] != 'a') TRUE	&temp_5[2]	
3	&temp_5[2]	While(temp_5[2] != 'a') FALSE	&temp_5[3]	&temp_5[2]

2. while(*copy_haystack++ != o_needle)구체적분석

	같은 행에서 분석순서 =>		
횟수	copy_haystack	while(*copy_haystack++ != o_needle) ;	
		*copy_haystack++ == *(copy_haystack++)	*copy_haystack++ != o_needle
1	&temp_5[0]	Temp_5[0] == '1'	temp_5[0] != 'a' TRUE
2	&temp_5[1]	Temp_5[1] == '2'	temp_5[1] != 'a' TRUE
3	&temp_5[2]	Temp_5[2] == 'a'	temp_5[2] != 'a' FALSE

3.6.strstr()

3.6.1.정의

함수	
char *StrStr (const char *phaystack, const char *pneedle)	**1.기능** : phaystack 의 문자열에서 pneedle 의 문자열과 동일한 문자열로 시작하는 포인터를 return 한다 **2.필요데이타** **3.필요한 함수.** **StrChr();** **4. parameter** 1) const char *phaystack : Original Haystack 2) const char pneedle : Original needle **5. return**: Haystack(건초더미)에서 m_needle(바늘)이 발견된 곳의 Pointer 를 return 한다. **6. algorithm** **haystack 에서 Linear 서치하다가 틀리면, 틀린 현 위치에서 needle 을 다시 서치한다.** **7.참고** **HayStack (건초더미)에서 a Needle(바늘)찾기프로그램**

3.6.2.소스

```
char *StrStr (const char *phaystack, const char *pneedle)
{
	// [1]. Data
	// haystack data
	register const unsigned char *n_haystack; // new haystack
	unsigned char *d_haystack ; // Dig up    haystacks

	// needle data
	register const unsigned char *s_needle; // start_position of needle
	unsigned char *d_needle, *e_needle; // Dig up haystacks, end_position of needle
	int pneedle_size = StrLen(pneedle); // needle size

	// initiate data
	n_haystack     = d_haystack  = (const unsigned char *) phaystack;
	s_needle  = d_needle            = (const unsigned char *) pneedle;
	e_needle  = s_needle + pneedle_size-1;

	// [2].
	// [2-1]. Check Error
	if(phaystack == NULL || pneedle == NULL
		|| *phaystack == NULL    || *pneedle == NULL )
		return NULL;

	// [2-2].match *d_needle(== d_needle[0]) with phaystack ?
	if( ( n_haystack = d_haystack = StrChr(d_haystack, *s_needle)) == NULL )
```

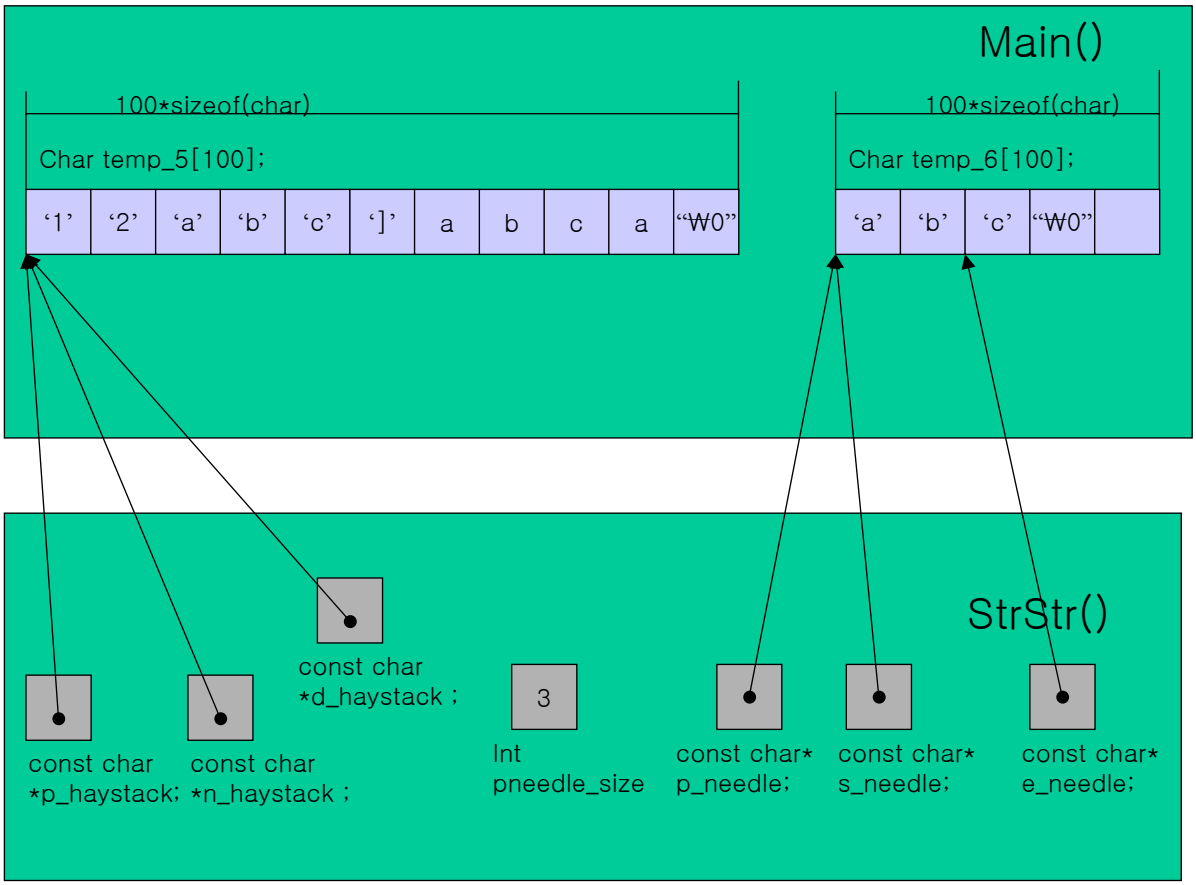
```
            return NULL;

        // [2-3].
        while(*d_haystack !=NULL)
        {
            // [2-3-1]. The Work digging up haystack
            while(++d_needle <= e_needle && ++d_haystack != NULL)
            {
                // if not find
                if(*d_haystack != *d_needle )
                {
                    d_needle = s_needle;
                    break; // Exit    of [2-3-1]. while()
                }

            }

            // [2-3-2]. if find a needle
            if(--d_needle == e_needle && *d_haystack == *e_needle )
                return    (d_haystack -(pneedle_size-1));

            // [2-3-3]. Fail to find a needle & Find a needle from new haystack
            else
            {
                d_needle = s_needle;

                // match *s_needle ?
                if( ( n_haystack = d_haystack = StrChr(d_haystack, *s_needle)) == NULL )
                    return NULL;
            }
        }

        return NULL;
}
```

3.6.3.Analysis

```
void main()
{
	char temp_5[100]="12abc]abca";
	char temp_6[100]="abc";
	char *point;

	point =StrStr(temp_5,temp_6);
	point =StrStr(point+1,temp_6);

}
```

3.6.3.1.StrStr(temp_5, temp_6)의 Parameter 넘기는 초기화면

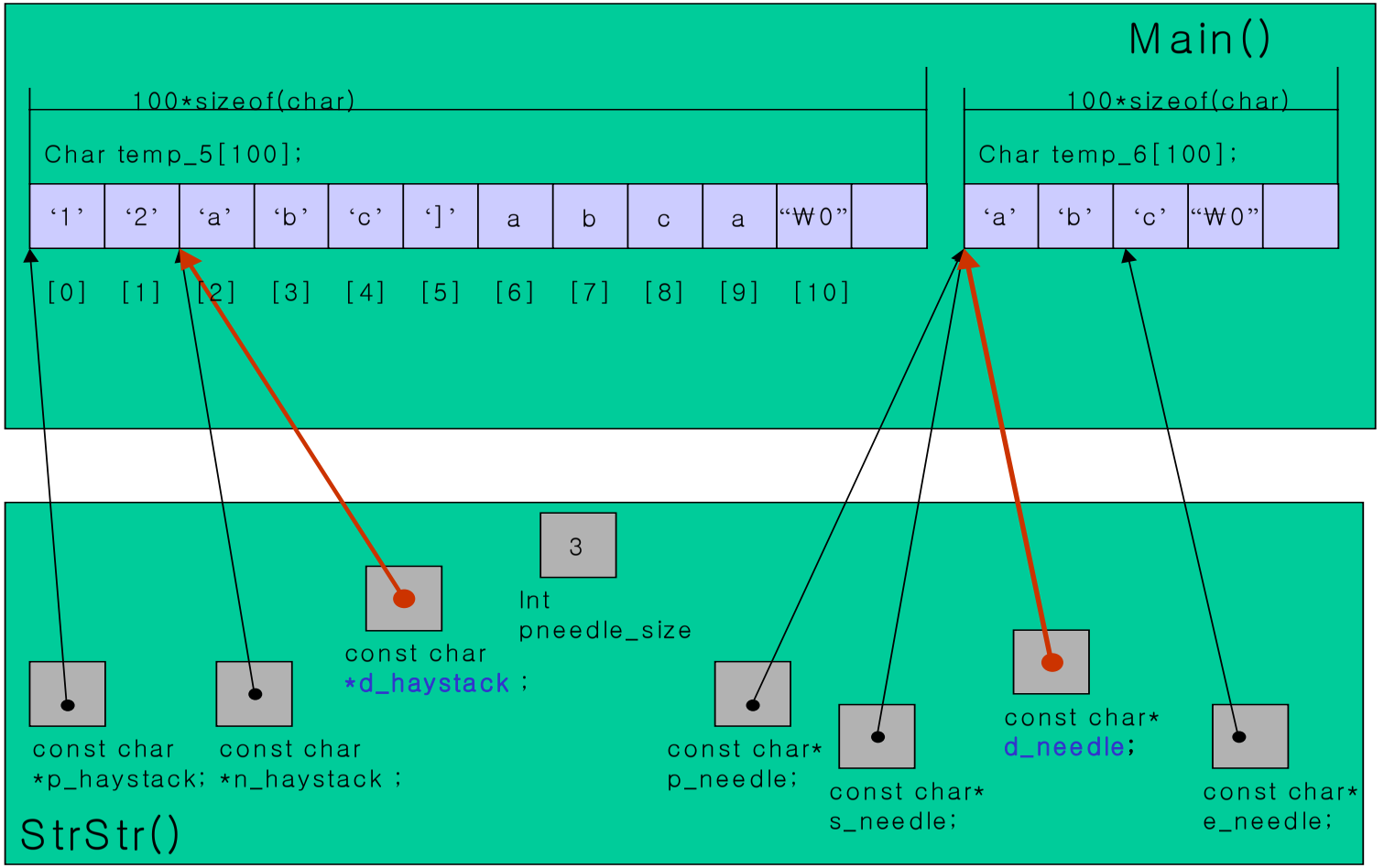
StrStr();에서 [2-1]. Check Error 까지 실행된 모습



3.6.3.2.1. StrStr(temp_5, temp_6)을 Trace & Debugging 화면

3.6.3.2.1.1.StrChr(d_haystack, *s_needle)

횟수	d_haystack	*s_needle	n_haystack = d_haystack = StrChr(d_haystack, *s_needle)
1	&temp_5[0]	‘a’	n_haystack = d_haystack = &temp_5[2]


Main()
100*sizeof(char)
Char temp_5[100];
'1' '2' 'a' 'b' 'c' ']' a b c a "₩0"
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10]
100*sizeof(char)
Char temp_6[100];
'a' 'b' 'c' "₩0"
3
Int
pneedle_size
const char
*d_haystack ;
const char
*p_haystack;
const char
*n_haystack ;
const char*
p_needle;
const char*
s_needle;
const char*
d_needle;
const char*
e_needle;
StrStr()

3.6.3.2.1.2.Find a Needle

3.6.3.2.1.2.1.StrStr() Trace Routine

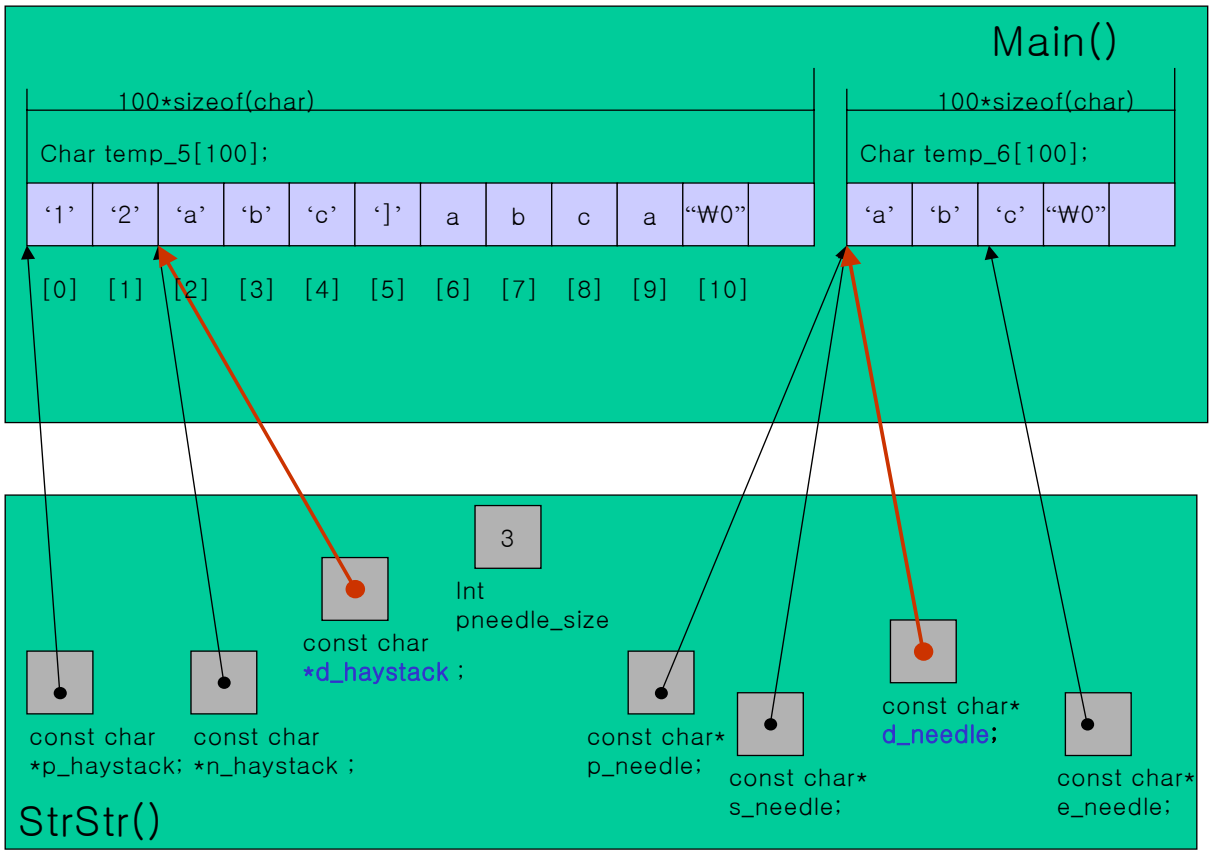
횟수	while(*d_haystack !=NULL)	while(++d_needle <= e_needle && ++d_haystack != NULL)			if(*d_haystack != *d_needle)		if(--d_needle == e_needle && *d_haystack == *e_needle)				else	
		++d_needle	e_needle	++d_haystack	*d_haystack	*d_needle	--d_needle	e_needle	*d_haystack	*e_needle	d_needle = s_needle;	if((n_haystack =d_haystack= StrChr(d_haystack, *s_needle)) == NULL)
1	*d_haystack = temp_5[2]	&temp_6 [1]	&temp_6 [2]	&temp_5[3]	temp_5 [3]	temp_6 [1]						
2		&temp_6 [2]	&temp_6 [2]	&temp_5[4]	temp_5 [4]	temp_6 [2]						
3		&temp_6 [3]	&temp_6 [2]	앞에서 탈출			&temp_6 [2]	&temp_6 [2]	temp_5 [4]	temp_6 [2]		
							return (d_haystack -(pneedle_size-1)); **== &temp_5[2]**					

- (주의점) 컴파일러 실행 순서

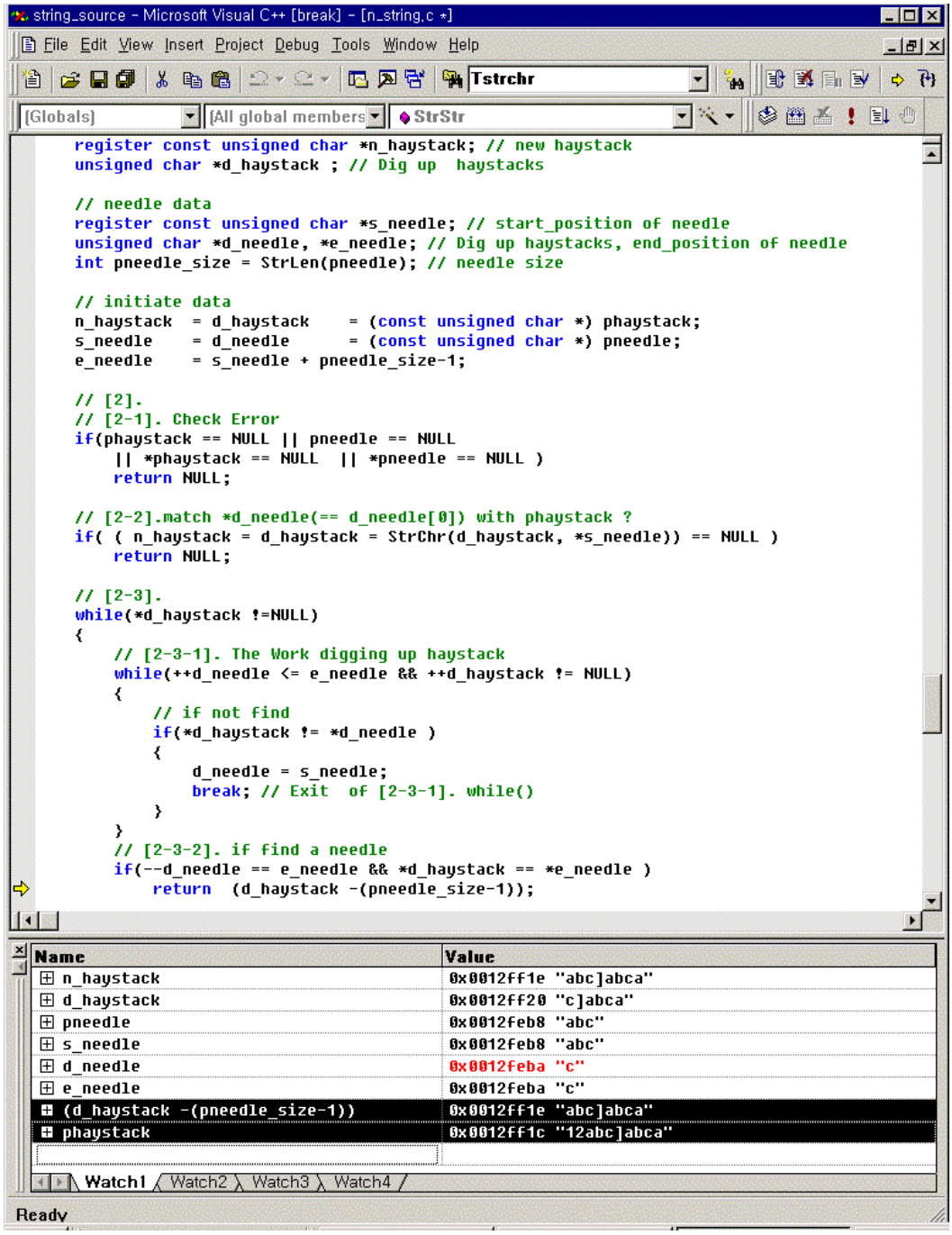
```
while(++d_needle <= e_needle && ++d_haystack != NULL)
```

1. ++d_needle <= e_needle 를 먼저 실행한다.
2. ++d_haystack != NULL 를 실행한다.

3.6.3.2.1.2.2. StrStr() Trace Routine 그림과 Debug Routine



3.6.3.2. point =StrStr(point+1,temp_6);



횟수	while(*d_haystack !=NULL)	while(++d_needle <= e_needle && ++d_haystack != NULL)			if(*d_haystack != *d_needle)		if(--d_needle == e_needle && *d_haystack == *e_needle)			
		++d_needle	e_needle	++d_haystack	*d_haystack	*d_needle	--d_needle	e_needle	*d_haystack	*e_needle
1	*d_haystack = temp_5[2]	&temp_6 [1]	&temp_6 [2]	&temp_5[3]	temp_5 [3]	temp_6 [1]				
2		&temp_6 [2]	&temp_6 [2]	&temp_5[4]	temp_5 [4]	temp_6 [2]				
3		&temp_6	&temp_6	앞에서 탈출			&temp_6	&temp_6	temp_5	temp_6

		[3]	[2]				[2]	[2]	[4]	[2]	
4											

3.7.Reverse

4. Linked List 소스 분석

4.1. Singled Linked List

4.2.Doubly Linked List

5. for()while()문의 특징.

5.1.Break 탈출경우-LinuxKernel 의 이해

5.1.1.example1

5.1.1.1.소스

```
#include <stdio.h>


void main()
{

	int loop;

	for(loop = 0 ; loop <10; loop++)
	{

		if(loop ==5)
		{
			printf("%d\n",loop);
			break;
		}
	}

	printf("%d",loop);

}
```

=>결과

```
5
```

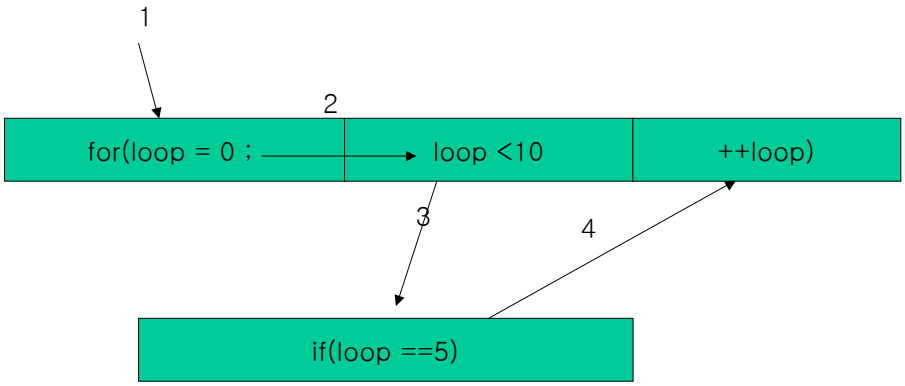
5.1.1.2.Debug

break;가 들어있는 for 까지만 효력이 발생한다.

[표 5.1.1.2.가]

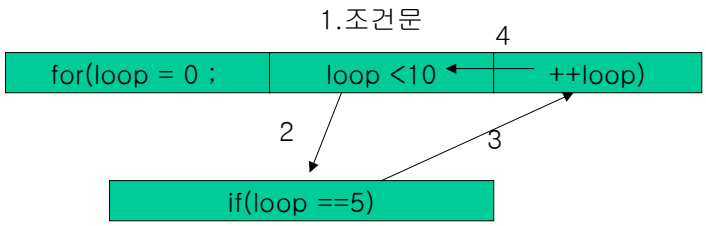
			1 회	2 회	3 회	4 회	5 회
1.초기값	loop =0;		1.Loop=0				
2 조건문	loop<10		2.True	1.True	1.True	1.True	True
3.증감	loop++		4.Loop = 1	3.Loop = 2	3.Loop = 3	3.Loop = 4	3.Loop = 5
	if(loop == 5)		3.False	2.False	2.False	2.False	2.False
		Break					break; for 문탈출 =>loop =5 이다.

[처음 FOR문이 실행될때-1회]




```
printf("%d₩n",loop);
break;
```

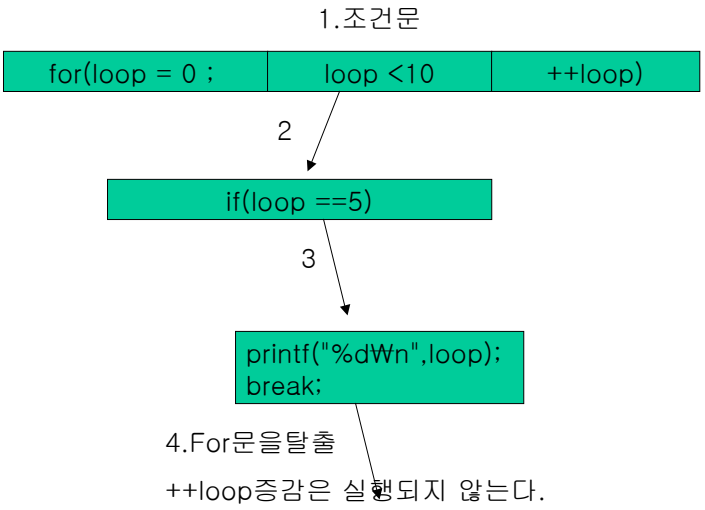
[FOR문이 실행될때-2~4회:for문의 수행횟수]




```
printf("%d₩n",loop);
break;
```

1->2->3->4를
반복수행한다

[FOR문이 실행될때-5회] loop ==5;



5.1.2.Example2

5.1.2.1.소스

```
#include <stdio.h>
void main()
{
	int index;
	int loop;
	for(index = 0; index < 5; index++)
		for(loop = 0 ; loop <10; ++loop) // break 가 영향받는 곳.
		{

			if(loop ==5)
			{
				printf("%d\n",loop);
				break;
			}

		}
}
```

=>결과

```
5
5
5
5
5
```

5.1.2.2.Debug

break;가 들어있는 for 까지만 효력이 발생한다.

ANNEX A. Version 관리

이 문서는 Free 입니다.

단 이 문서의 저작권은 scwpark@hananet.net에게 있습니다.

일 시	변경코드	작성자	주 석
2001/10/22	1.0	박성태	Pointer 문서 1.0
2001/10/25	1,1	박성태	Pointer 문서 1.0
2001/11/11	1.5	박성태	2.9.함수포인터 추가.
2001/11/15	1.51	박성태	2.3.1.strcpy 그림수정, 2.5 그림 추가.
2001/12/14	1.52		2 ~2.8 까지 전체적 추가
2001/12/20	1.52@		전체적 오타수정
2001/12/21	1.53		1.1.2 수정, 2.9.1.2.2.그림 수정
2002/1/31			1.1.3 추가,1.3, 1.5 추가 3 장추가
2002/4/1	1.54		1.6 장 Attribute 의 offset 구하기.
2002/4/11	1.54@		Parsing 수정(미완성)

이 문서가 나올쯤에는 더 업그레이드 된 문서가 작성되고 있을 겁니다.

여러분 많은 기대 바랍니다.

혹시 이문서가 괜찮다면 혼자만 보지 마시고, 이 글을 쓴 저자에 취지에 맞게(당근 기술을 공유하여 Win-Win), 주변분들에게도 이문서를 공유하여 주세요.)

그리고 협력하여 공부하는 법을 배우세요. 프로그래머의 단점이 협력 공부입니다.

협력하여 Win-Win 하세요.