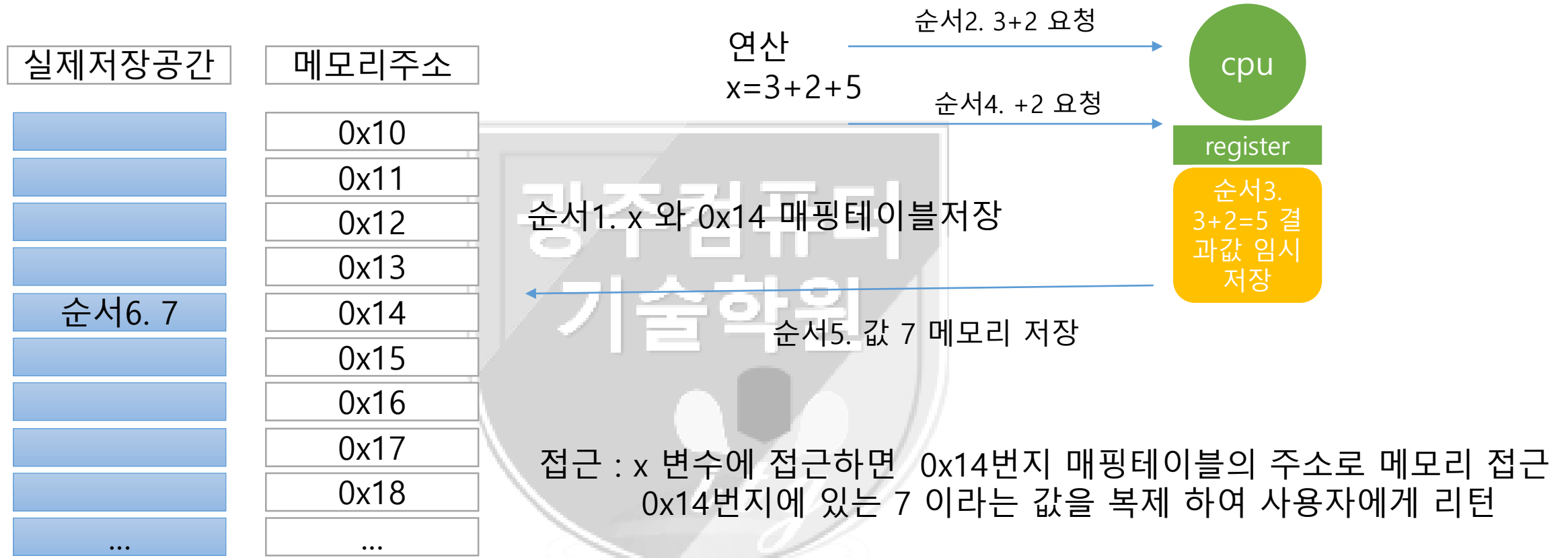


프로그래밍 기초

C 언어

메모리



메모리 저장공간에 저장 형태


1. 실제 값을 이진수로 저장
2. 메모리 주소를 이진수로 저장

프로그래밍에서는 빠른속도를 위해 16진수로 연산 접근

메모리 저장공간에 저장 형태

메모리 접근시 실제 값이 나온다면 직접주소접근방식

메모리 접근시 메모리 주소가 나온다면 간접주소접근방식



데이터를 만들어 내는 과정

컴퓨터 내부로 저장되어지는 데이터는 이진수 이다

사용자가 입력하는 데이터는 10진수 문자열 등 다양하다.

16진수 10진수 8진수 전환은 진법변환으로 가능하나 문자열은 진법 변환이 불가능 하므로
아스키코드 테이블을 만들어 숫자로 전환후 이진수로 전환하는 방법을 제공한다.

물론 색상또한 색상 코드테이블을 두어 이진수 전환이 가능함

컴파일러

컴파일류 언어는 모든 코드 작성후 컴파일 링크 단계를 거쳐 기계에서 실행가능한 기계어로 바뀐다.

C 언어, 자바, 파이썬등은 컴파일 언어이다.

c 언어 기초

특징 : 고급언어 종류에 속하지만 어셈블리어 수준으로 하드웨어를 제어할 수 있는 저급언어의 특징이 있다.

절차 지향 프로그래밍 언어이다. 자료형 타입이 엄격하며 대소문자를 구분한다.

구조

```
#include <stdio.h> //외부 모듈을 불러온다.
```

```
#define TEXT "Welcome" //TEXT 타입으로 문자를 정의한다.
```

```
int main(){ // 메인 쓰레드 코드 블록 , 프로세스 쓰레드를 발생시킨다
```

```
    printf(TEXT); //stdio.h 의 명령어인 printf 함수를 불러온다.
```

```
    return 0; //리턴값이 int 타입이므로 0을 리턴 시켜준다.
```

```
}
```

문법 : 대소문자 구분, 명령문의 마침표는 ; , 블럭등은 생략가능(; 은 반드시 작성해야 오류가 나지 않는다.)

주석 : 한줄 주석은 // 여러줄 주석은 /* 문장 */ (주석문은 실행시 영향을 주지 않는 문장입니다.)

printf() – 표준 출력함수

```
printf("서식을 지정할 수 있습니다. \n 줄바꿈을 합니다.")
```

```
printf("다음 숫자는 %d 입니다.",10)
```

자주쓰이는 이스케이프 시퀀스

:: 텍스트 문자열 내부에서 사용

\' : ' 출력

\\" : " 출력

\? : ? 출력

\\ : \ 출력

\\n : 줄바꿈

\\r : carriage return

\\t : tab 입력

자주쓰이는 포맷 서식지정자

:: 텍스트 문자열 내부에서 사용

%c : 한문자

%s : 문자열

%d : 부호있는 10진 정수

%i : %d 와동일

%f : 실수(고정소숫점 6자리)

%o : 8진정수

%x,%X : 16진 정수

%% : % (퍼센트)기호출력

```
printf("%%c 를 사용한 결과 : %c \n",'a'); // 한문자 character 는 ' 로 감싸주며 아스키코드표와 대칭됩니다.
```

A 는 정수 65 a는 정수 97

```
printf("%%f 사용결과 : %f\n",0.123456);
```

```
printf("%%s 사용결과 : %s\n","정보처리기사")
```

```
printf("%%x를 사용한 결과 : %x\n", 123); // => 7,11 => 7b
```

```
printf(" %%+7d를 사용한 결과 : |%+7d|\n", 123); // %+7d를 사용한 결과 : | +123|
```

```
printf(" %%-7d를 사용한 결과 : |%-7d|\n", 123);// %-7d를 사용한 결과 : |123 |
```

```
printf(" %%7.2f를 사용한 결과 : |%7.2f|\n", 1.23);// %7.2f를 사용한 결과 : | 1.23|
```

입력함수

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

```
int main(void)
{
    int num01, num02;

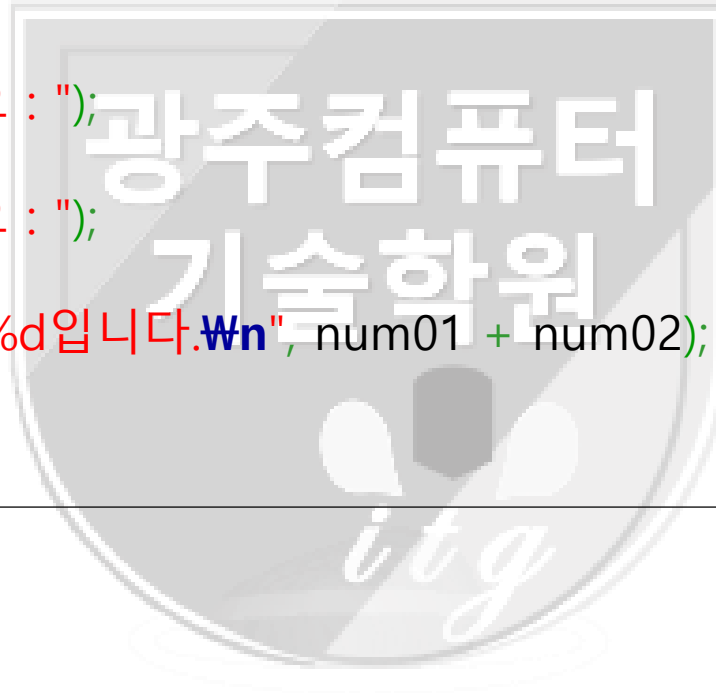
    printf("첫 번째 정수를 입력하세요 : ");
    scanf("%d", &num01);
    printf("두 번째 정수를 입력하세요 : ");
    scanf("%d", &num02);
    printf("입력하신 두 정수의 합은 %d입니다.\n", num01 + num02);
    return 0;
}
```

실행 결과

첫 번째 정수를 입력하세요 : 10
두 번째 정수를 입력하세요 : 20
입력하신 두 정수의 합은 30입니다.

'&'기호는 주소 연산자 입니다.

주소 연산자는 입력받은 데이터를 뒤에 나오는 변수에 저장하라는 의미입니다.



변수타입 – 메모리에 알맞은 크기의 데이터를 저장하기 위한 타입

1. c 언어는 변수를 사용하겠다고 반드시 선언하여 컴파일러에 알려야 합니다.
2. 선언후 나중에 초기화 하는 방법 `int num; num=20;`
3. 선언과 동시에 메모리를 초기화 하는 방법 `double num0= 1.25, num1=3.24;`

상수타입 – `const` 키워드로 선언하며 앞으로 변경하지 않을 숫자이다.
변경되지 않아야 될 리터럴이므로 선언과 동시에 초기화 하며
의미를 부여하기 위해 보통 대문자를 사용하고 연결은 `_` 로 한다.

`const int MAX = 10;` // 리터럴 상수 10 을 심볼릭상수 MAX에 저장하였다.

타입 : (타입의 크기를 넘어가는 데이터 저장시 오버플로우 발생)

정수형 `short(2b)`, `unsigned short(2b)`, `int(4b)`, `unsigned int(4b)`, `long(4b)`, `unsigned long(4b)`

실수형 `float(4b)`, `double(8b)`, `long double(8b)` //내부적으로 정수부와 실수부가 나뉘어 저장

문자형 : `char(1b)`, `unsigned char(2b)`

입력코드

```
char ch = 'a';
```

```
printf("변수 ch에 저장된 값은 %c입니다.\n", ch);
```

```
printf("변수 ch에 저장된 값은 %d입니다.\n", ch);
```

실행 결과

변수 ch에 저장된 값은 a입니다.

변수 ch에 저장된 값은 97입니다.

타입의 변환(casting)

묵시적 변환 : long a = 100l; int b = a; printf("%d",b) :: int 값 100 출력

명시적 변환 : int num01=1; int num02=4; double res1 = (double)num01/num02; :: 0.250000 출력

연산자

산술연산 : + - * / % (우선순위, 동일 우선순위는 전자가 먼저 실행됩니다. () 는 최우선 순위)

대입연산 : = += -= *= /= %=

증감연산 : ++y (선증가), y++ (후증가), y--, --y

```
int x = 10;
```

```
int y = x-- + 5 + --x; // 10+5 = 임시값(15) 연산후 x=9 => 15+8 = y assign 23, x=8
```

```
printf("변수 x의 값은 %d이고, 변수 y의 값은 %d입니다.\n", x, y);
```

출력 :: 변수 x의 값은 8이고, 변수 y의 값은 23입니다.

비교연산 : == != > >= < <= :: 비교조건에 만족하면 true || 1, 불만족시 false || 0

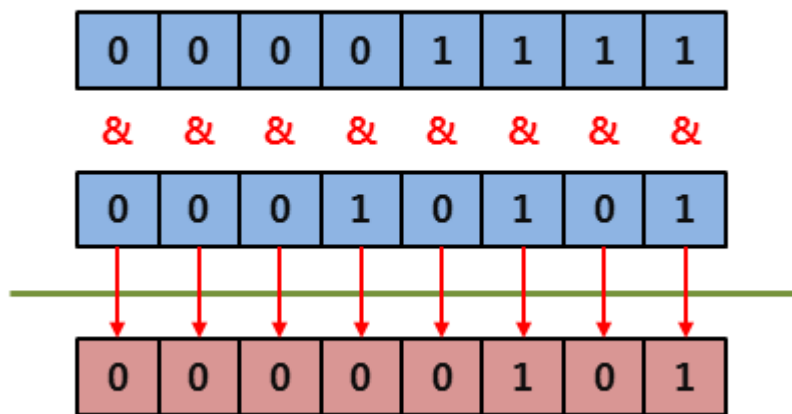
c 언어는 0은 거짓(false)으로 평가되며 0이외의 값은 참(true)으로 평가됩니다.

논리연산 : && || !(not)

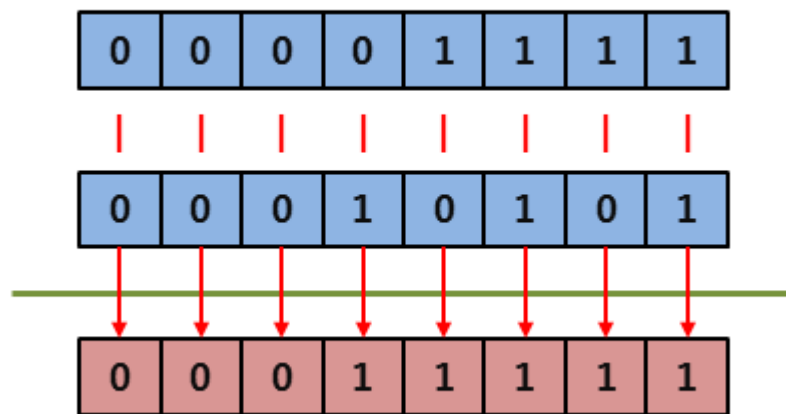
비트연산 : & | ^(xor) ~(not) << >>

시험에 어렵게 출제 될수 있는 비트 연산자

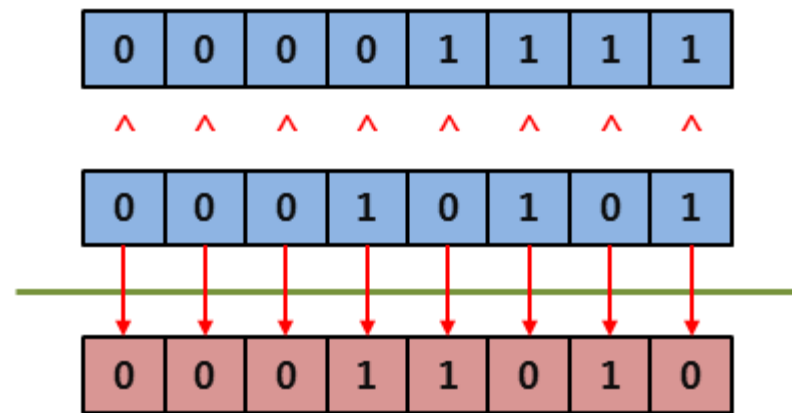
비트 AND 연산



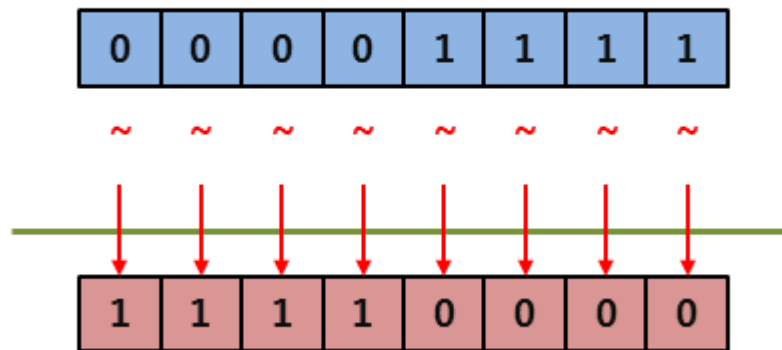
비트 OR 연산



비트 XOR 연산



비트 NOT 연산



↑
부호비트

unsigned 타입이 아닐경우는
음수 표현은 2의 보수 출력

비트연산의 예

```
int x = 7;           // 00000000 00000000 00000000 00000111
//음수 표현 2의 보수
printf("%d", ~x);    // 11111111 11111111 11111111 11111000 : -8
출력 : -8
```

```
int x = 7;           // 00000000 00000000 00000000 00000111
int y = 10;          // 00000000 00000000 00000000 00001010
printf( " %d " , x & y); // 00000000 00000000 00000000 00000010 : 2
출력 : 2
```

참항연산

조건? 참일때 반환값 value : 거짓일때 value

```
int num01 = 15;
int num02 = 8;
int result;
```

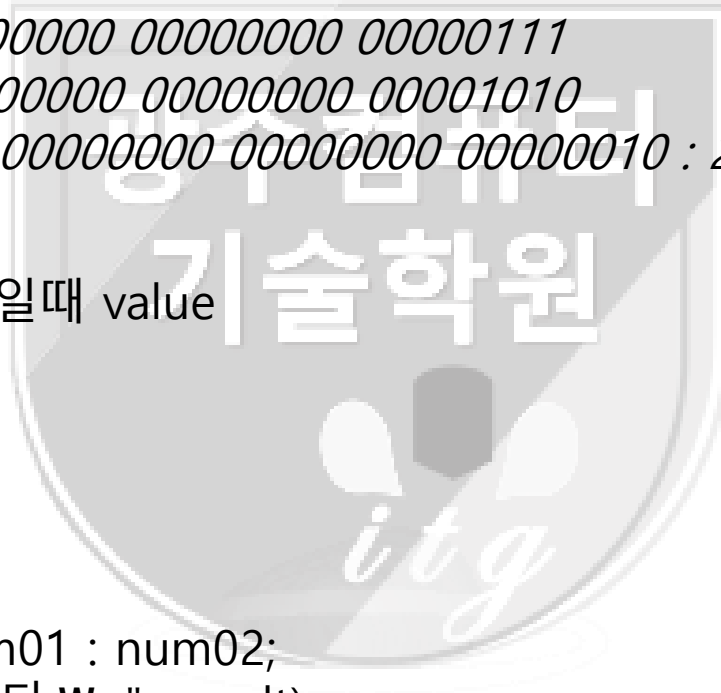
```
result = (num01 > num02) ? num01 : num02;
printf("둘 중에 더 큰수는 %d입니다.\n", result)
```

결과

둘 중에 더 큰수는 15입니다.

C 언어 에서 sizeof 는 count 랑 비슷

sizeof 변수 – 변수에 할당된 메모리 사이즈 byte 값으로 반환



포인터연산자-변수가 가르키는 메모리 위치

1. & 주소연산자
 - 변수의 주소를 반환 합니다.
 2. * 참조연산자
 - 변수의 주소에 저장된 값을 반환합니다.
- 타입* 포인터 = 주소값;

```
int x = 7;           // 변수의 선언
int *ptr = &x;       // 포인터의 선언
int** pptr = &ptr;   // 포인터의 참조
int pstr = &ptr;
```

변수명 : x

할당메모리주소:

0x01

7

x = 7

변수명 : ptr

할당메모리주소:

0x08

0x01

*ptr = &x

변수명 : ptr1

할당메모리주소:

0xff

0x01

*ptr1 = ptr

변수명 : x

할당메모리주소:

0x01

7

x = 7

변수명 : ptr

할당메모리주소:

0x08

0x01

*ptr = &x

변수명 : pptr

할당메모리주소:

0xff

0x08

**pptr = &ptr

메모리 주소를 받을때는 타입은 동일하게
*을 접두시켜 참조값 변수라는 것을 알려준다.

주소를 참조하고 있는 변수는 반드시 참조 변수를 선언하여
간접주소방식이 오류없이 작동된다.



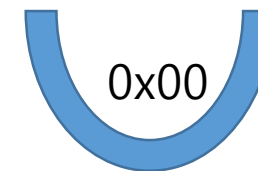
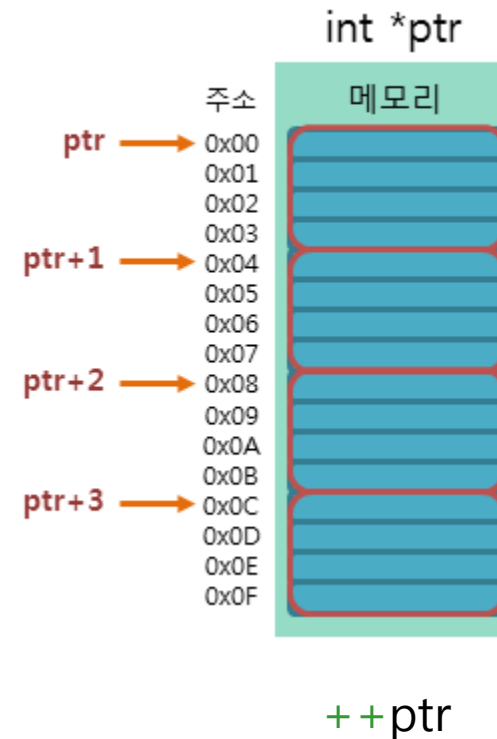
포인터연산자 예제

```
#include <stdio.h>
int main(void)
{
    int num01 = 1234;
    double num02 = 3.14;
    int* ptr_num01 = &num01;
    double* ptr_num02 = &num02;

    printf("포인터의 크기는 %d입니다.\n", sizeof(ptr_num01));
    printf("포인터 ptr_num01이 가리키고 있는 주소값은 %#x입니다.\n", ptr_num01);
    printf("포인터 ptr_num02가 가리키고 있는 주소값은 %#x입니다.\n", ptr_num02);
    printf("포인터 ptr_num01이 가리키고 있는 주소에 저장된 값은 %d입니다.\n",
        *ptr_num01);
    printf("포인터 ptr_num02가 가리키고 있는 주소에 저장된 값은 %f입니다.\n",
        *ptr_num02);
    return 0;
}
```

결과 : 메모리 주소는 각각 달라집니다.
포인터의 크기는 8입니다.
포인터 ptr_num01이 가리키고 있는 주소값은 0x7c255e4입니다.
포인터 ptr_num02가 가리키고 있는 주소값은 0x7c255e8입니다.
포인터 ptr_num01이 가리키고 있는 주소에 저장된 값은 1234입니다.
포인터 ptr_num02가 가리키고 있는 주소에 저장된 값은 3.140000입니다.

포인터의 연산



$$0x00 + 1 = 0x04 \text{ (4byte 건너뛰)}\text{}$$

if 조건문

if(조건1){참일때 수행} else if(조건2){조건1이 거짓이고 조건2가 참일때}else{조건1, 조건2가 모두 거짓일때}
- if 문 뒤의 else if..... else 는 모두 하나의 문장으로 하나의 조건만 수행하고 점프

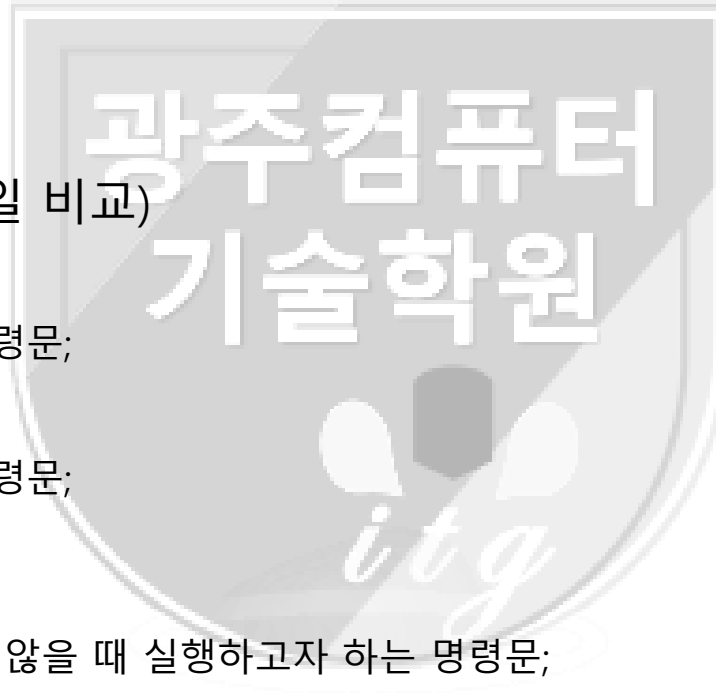
```
int num = 7;  
if (num < 5) { printf("입력하신 수는 5보다 작습니다.\n");}  
else if (num == 5) { printf("입력하신 수는 5입니다.\n");}  
else { printf("입력하신 수는 5보다 큼니다.\n");}
```

실행 결과

입력하신 수는 5보다 큼니다.

switch 조건문 (범위가 아닌 == 동일 비교)

```
switch (조건 값) {  
    case 값1:  
        조건 값이 값1일 때 실행하고자 하는 명령문;  
        break;  
    case 값2:  
        조건 값이 값2일 때 실행하고자 하는 명령문;  
        break;  
    ...  
    default:  
        조건 값이 어떠한 case 절에도 해당하지 않을 때 실행하고자 하는 명령문;  
        break;}
```



switch 예제

```
char ch = 'a';
switch (ch) {
    case 'a':
    case 'A':
        printf("이 학생의 학점은 A입니다.\n");
        break;
    case 'b':
    case 'B':
        printf("이 학생의 학점은 B입니다.\n");
        break;
    case 'c':
    case 'C':
        printf("이 학생의 학점은 C입니다.\n");
        break;
    case 'd':
    case 'D':
        printf("이 학생의 학점은 D입니다.\n");
        break;
    case 'f':
    case 'F':
        printf("이 학생의 학점은 F입니다.\n");
        break;
    default:
        printf("학점을 정확히 입력해 주세요!(A, B, C, D, F)");
        break;
}
```



반복문

while (조건식){

조건식의 결과가 참인 동안 반복적으로 실행하고자 하는 명령문;}

int i = 0; **int** num = 5; 0,1,2,3,4 는 while 안쪽을 수행, 5 가 되면 빠져나옴. 즉 i 값은 최종적으로 5가 됨

while (i < num){

printf("while 문이 %d 번째 반복 수행중입니다.\n", i + 1);

i++; // 이 부분을 삭제하면 무한 루프에 빠지게 됨

}

printf("while 문이 종료된 후 변수 i의 값은 %d입니다.\n", i);

결과

while 문이 1 번째 반복 수행중입니다.

while 문이 2 번째 반복 수행중입니다.

while 문이 3 번째 반복 수행중입니다.

while 문이 4 번째 반복 수행중입니다.

while 문이 5 번째 반복 수행중입니다.

while 문이 종료된 후 변수 i의 값은 5입니다.

do {

조건식의 결과가 참인 동안 반복적으로 실행하고자 하는 명령문;

} **while** (조건식);

while 문과 do ~ while 문의 차이점
조건에 상관없이 무조건 한번의 실행이
필요할때 do ~ while 문을 사용한다.



for 문

for (초기식; 조건식; 증감식) {
 조건식의 결과가 참인 동안 반복적으로 실행하고자 하는 명령문;}

```
int i;  
int num = 7;
```

```
for (i = 0; i < num; i++){  
    printf("for 문이 %d 번째 반복 수행중입니다.\n", i + 1);}  
printf("for 문이 종료된 후 변수 i의 값은 %d입니다.\n", i);
```

출력결과

```
for 문이 1 번째 반복 수행중입니다.  
for 문이 2 번째 반복 수행중입니다.  
for 문이 3 번째 반복 수행중입니다.  
for 문이 4 번째 반복 수행중입니다.  
for 문이 5 번째 반복 수행중입니다.  
for 문이 6 번째 반복 수행중입니다.  
for 문이 7 번째 반복 수행중입니다.  
while 문이 종료된 후 변수 i의 값은 7입니다.
```

while 문과 for 문의 차이점
while문은 증감식과 초기식이 없어
바깥쪽에 초기식을 넣어주고
루프 안쪽에 증감식을 넣어 조건에서
탈출할 수 있다.

루프문의 제어

continue; 문장을 만나는 시점에서 다음 구문을
실행하지 않고 for 문은 증감식 수행후
계속 구문을 진행한다.

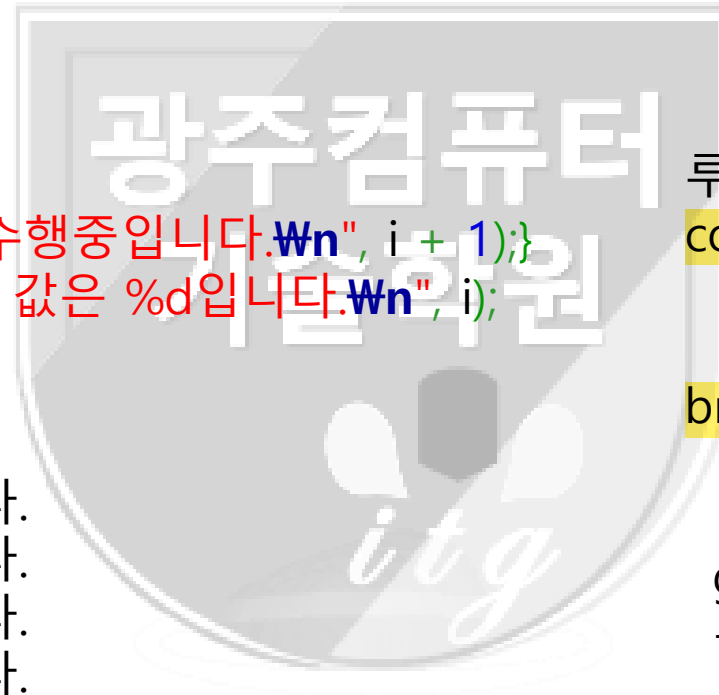
break; 문장을 만나는 시점에서 가장 가까운 루프
문장을 탈출한다.

goto 문 지정된 레이블로 점프한다.
- 점프로 문장이 복잡해져서 현업에서도
잘 사용하지 않는다.

```
goto ONE;
```

```
...  
ONE:
```

.....



재귀호출 - 반복문을 대체 할 수 있다.

```
int rSum(int n){  
    if(n==1){  
        return 1;  
    }  
    return n+rSum(n-1);  
}  
printf("%d",rSum(5));
```

결과
15



배열 (array) – 단일 이름을 갖는 변수 집합체 – 배열의 이름은 포인터 상수입니다.

1차원 배열 - 단일 이름으로 메모리 주소에 매핑 된다. 반드시 변수와 동일하게 타입을 지정해야한다.

타입 변수명 : int arr

```
index : 0    [1]    [2]    [3]    ...
```

배열의 크기인 1차원
크기를 반드시 지정
해야한다.

2차원 배열 - 단일 이름으로 메모리 주소에 매핑 된다. 반드시 변수와 동일하게 타입을 지정해야한다.

타입 변수명 : int arr_2

```
index : 00  [1][0]  [2][0]  ...
        01  [1][1]  [2][1]
```

배열의 크기인 2차원
크기를 반드시 지정
해야한다.

* 인덱스 주소는 0 부터 시작하기에 인덱스 번호는 배열 총길이보다 1이 적다

배열의 선언 :

배열의 길이 = sizeof(배열 이름) / sizeof(배열 이름[0])

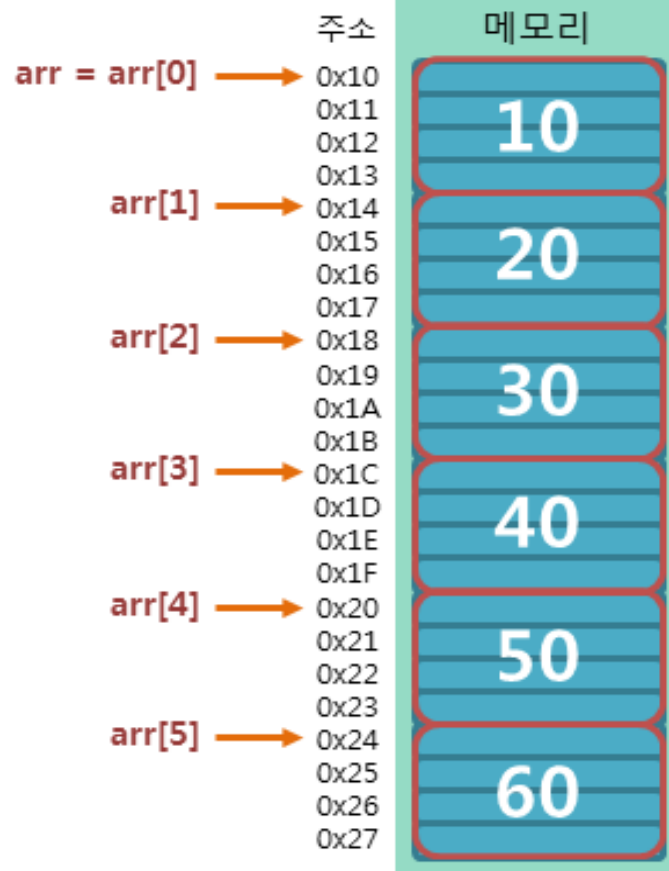
타입 이름[길이]; //선언만

타입 배열 이름[길이] = {요소1,요소2,...} //선언과 동시에 초기화

타입 배열 이름[] = {요소1,요소2,...} //묵시적 자동길이지정

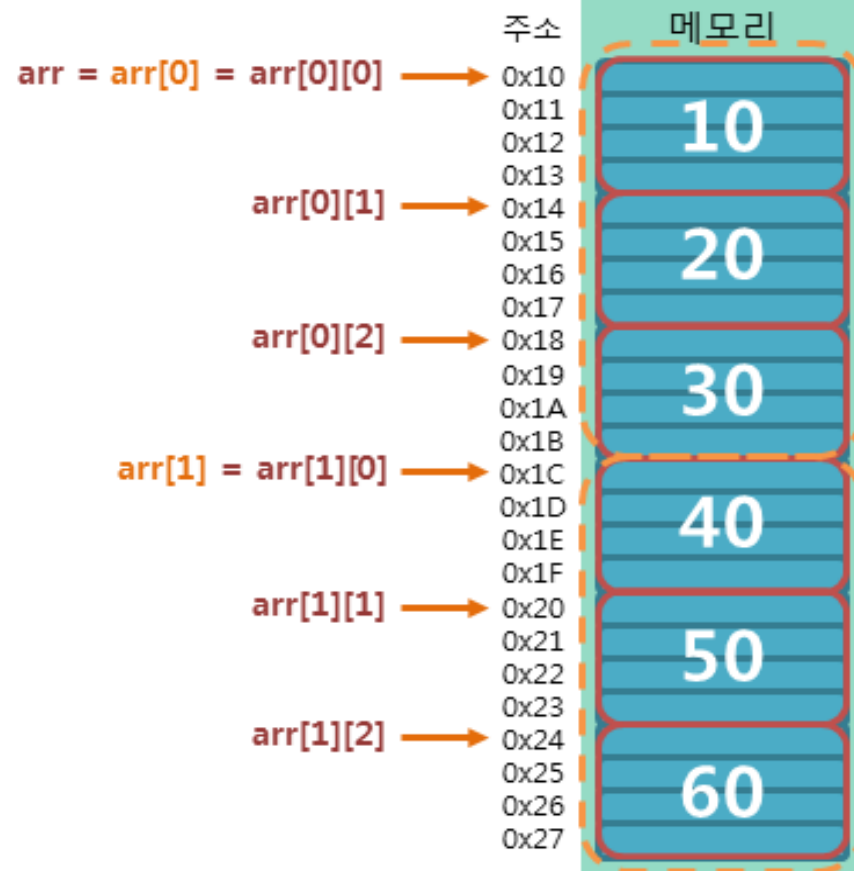
int arr1[6]

int arr1[6]



int arr2[2][3]

int arr2[2][3]



int arr2[열][행]

```
int i;
int sum = 0;
int grade[3] = {85, 65, 90}; // grade[0], grade[1], grade[2]만 선언 및 초기화
grade[3] = 100;             // grade[3]를 선언하지 않고 초기화 진행

for (i = 0; i < 4; i++)      // grade[3]도 수식에 포함
{
    sum += grade[i];
}
printf("국영수 과목 총 점수 합계는 %d이고, 평균 점수는 %f입니다.\n", sum, (double)sum/3);
```

실행 결과

국영수 과목 총 점수 합계는 340이고, 평균 점수는 113.333333입니다.

```
int grade[] = {85, 65, 90}; // 배열의 길이를 명시하지 않음
int arr_len = sizeof(grade) / sizeof(grade[0]); // 배열의 길이를 구하는 공식
printf("배열 arrGrade의 길이는 %d입니다.\n", arr_len);
```

실행 결과

배열 grade의 길이는 3입니다.

2차원배열

1차원형태 초기화(가독성 좋지 않음)

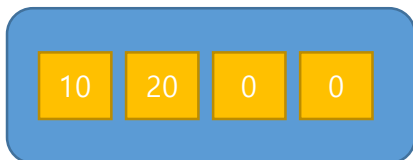
타입 배열이름[열의길이][행의길이] = {배열요소[0][0], 배열요소[0][1], ..., 배열요소[1][0], 배열요소[1][1], ..., 배열요소[2][0], 배열요소[2][1], ...};

2차원형태 초기화(가독성 좋음)

타입 배열이름[열의길이][행의길이] = {
{배열요소[0][0], 배열요소[0][1], ...},
{배열요소[1][0], 배열요소[1][1], ...},
{배열요소[2][0], 배열요소[2][1], ...},
...};

```
int arr[][4] = {  
    {10, 20},  
    {30, 40, 50, 60},  
    {0, 0, 70, 80}  
};
```

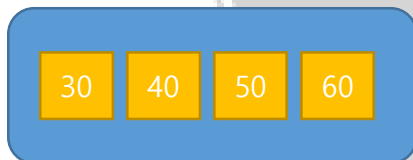
선언과 동시 초기화시 길이에 맞지
않는 부분은 0으로 자동초기화



[0 인덱스 열]

[0][0] == 10

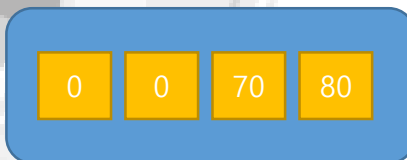
[0][1] == 20



[1 인덱스 열]

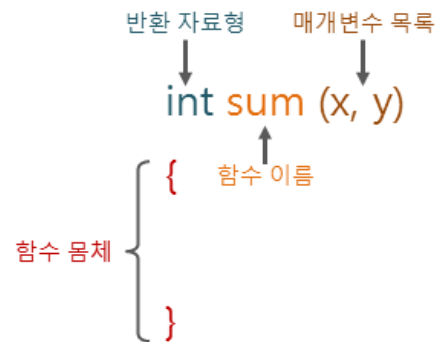
[1][0] == 30

[1][1] == 40



[2 인덱스 열]

함수



함수매개변수(파라미터)전달

1. 값전달
2. 참조형 전달

함수의 원형 선언

현재는 다중패스 컴파일 방식으로 오류가 없습니다.

원래 함수는 main 함수 전에 존재해야 함수를 사용할 수 있습니다.

함수를 main 뒤에 구현할 경우는 함수 원형(prototype)을 먼저 선언해줍니다.

반환타입 함수이름(매개변수타입);

{ 구현 부 } 후에 구현 하여도 됩니다.

```
int bigNum (int, int);
```

```
void main(){... bigNum();}
```

```
int bigNum(int n, int m){...}
```



```
include <stdio.h>
void local(int*);
int main(void){
    int var = 10;
    printf("변수 var의 초기값은 %d입니다.\Wn", var);
    local(var);
    printf("local() 함수 호출 후 변수 var의 값은 %d입니다.\Wn", var);
    return 0;
}
```

```
void local(int num)
{
    num += 10;
}
```

변수 var의 초기값은 10입니다.
local() 함수 호출 후 변수 var의 값은 10입니다.

```
include <stdio.h>
void local(int*);
int main(void){
    int var = 10;
    printf("변수 var의 초기값은 %d입니다.\Wn", var);
    local(&var);
    printf("local() 함수 호출 후 변수 var의 값은 %d입니다.\Wn", var);
    return 0;
}
```

```
void local(int* num)
{
    *num += 10;
}
```

변수 var의 초기값은 10입니다.
local() 함수 호출 후 변수 var의 값은 20입니다.

Variable Scope

지역변수 : 함수내 활동 변수

전역변수 : 함수 밖의 변수 :: 초기값 없을 때 0으로 자동초기화

- 전역변수와 지역변수의 이름이 같을때는 지역변수로 활동
- 함수 종료후 다시 전역변수 복구

static 변수 : 정적변수이며 지역범위내 선언하더라도 전역처럼 값을 유지하나
해당 지역에서만 사용가능하다(지역+전역)/프로그램이 종료될때 소멸

이중포인터

포인터에 포인터를 담는 변수

//다양한 포인터

void otherPoint(){

int b = 10;

int* ptr = &b;

int** pptr = &ptr;

printf("ptr 참조값: %d ", *ptr);

printf(" pptr 참조값: %d ", **pptr);

printf(" pptr 값: %x ", *pptr);

}

ptr 참조값: 10 pptr 참조값: 10

pptr 값: 61fec8

void 포인터(void pointer)

-타입 명시가 없어 어떠한 주소라도 담을수 있음.

-캐스팅후 접근 사용할 수 있다.

//void pointer

void otherPoint1(){

void *ptr;

int num = 1;

ptr = #

printf("ptr 출력:%d\n",*(int*)ptr);

float numf = 0.45;

ptr = &numf;

printf("ptr 출력:%f\n",*(float*)ptr);

int arr[3]={1,2,3};

ptr = arr;

printf("ptr arr 포인터 연산 출력:%d",*(((int*)ptr)+1));

printf("void 포인터 어레이접근: %d\n",((int*)ptr)[1]);

출력

ptr 출력:1

ptr 출력:0.450000

ptr arr 포인터 연산 출력:2

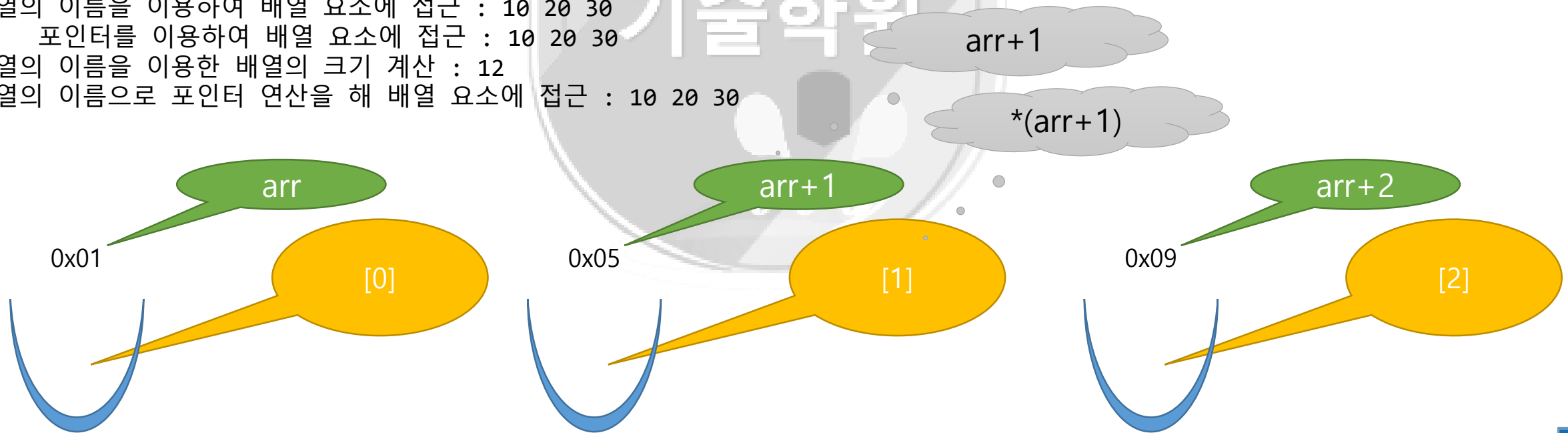
void 포인터 어레이접근: 2

포인터와 배열의 관계

배열의 이름은 포인터 상입니다.

```
void arrayPoint(){  
int arr[3] = {10, 20, 30}; // 배열 선언  
int *ptr_arr = arr; // 포인터에 배열의 이름을 대입함  
printf("배열의 이름을 이용하여 배열 요소에 접근 : %d %d %d\n", arr[0], arr[1], arr[2]);  
printf("포인터를 이용하여 배열 요소에 접근 : %d %d %d\n", ptr_arr[0], ptr_arr[1], ptr_arr[2]);  
printf("배열의 이름을 이용한 배열의 크기 계산 : %d\n", sizeof(arr));  
printf("배열의 이름으로 포인터 연산을 해 배열 요소에 접근 : %d %d %d\n", *(arr+0), *(arr+1), *(arr+2));  
}
```

배열의 이름을 이용하여 배열 요소에 접근 : 10 20 30
포인터를 이용하여 배열 요소에 접근 : 10 20 30
배열의 이름을 이용한 배열의 크기 계산 : 12
배열의 이름으로 포인터 연산을 해 배열 요소에 접근 : 10 20 30



포인터배열

배열요소가 포인터 / 포인터를 담는 배열이므로 일반배열과 다르게 타입* 선언

타입* 이름[크기]

```
void pointArr(){
    int i, arr_len;
    int num01 = 10, num02 = 20, num03 = 30;
    int* arr[3] = {&num01, &num02, &num03}; // int형 포인터 배열 선언
    arr_len = sizeof(arr)/sizeof(arr[0]);
    for (i = 0; i < arr_len; i++)
    {
        printf("%d\n", *arr[i]);
    }
}
```

출력
10
20
30

일반변수의 배열 형식 접근

```
int x = 35;
int y = 40;
int* xx = &x;
int* yy = &y;
printf("배열접근 %d ",(xx-1)[0]);
printf("배열접근 %d ",xx[0]);
printf("배열접근 %d ",yy[1]);
```



배열포인터

2차원 배열을 가르키는 포인터

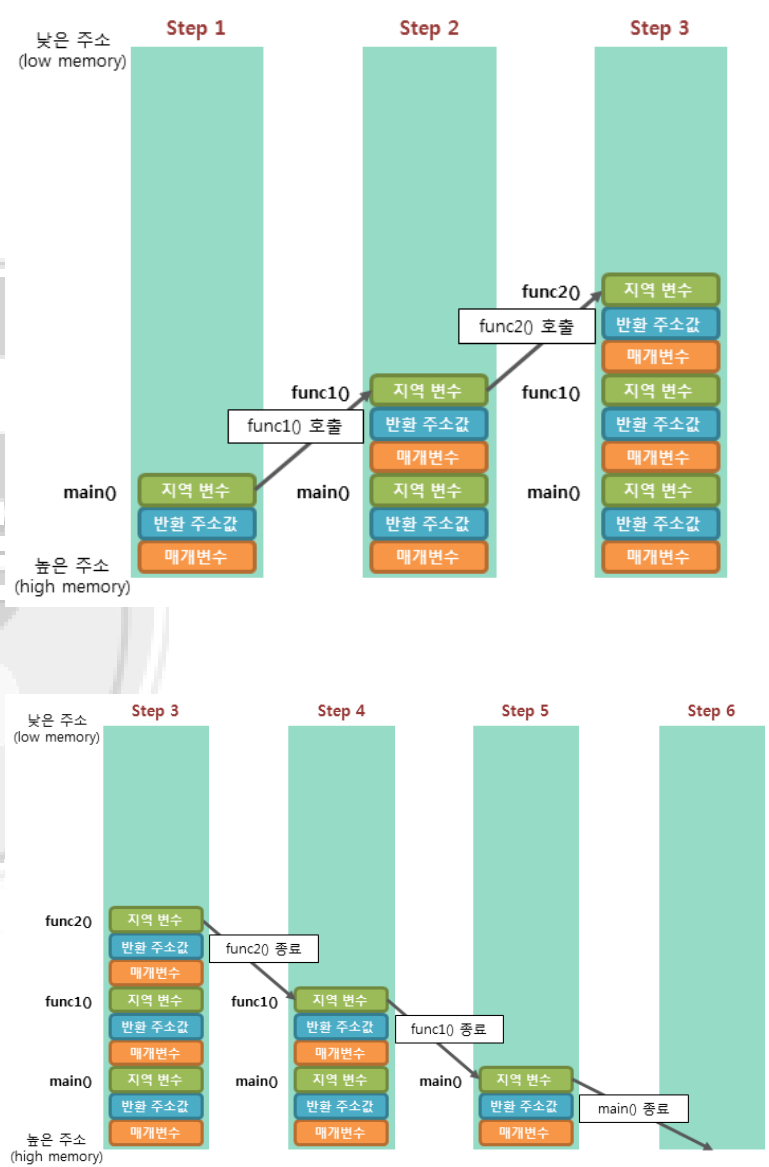
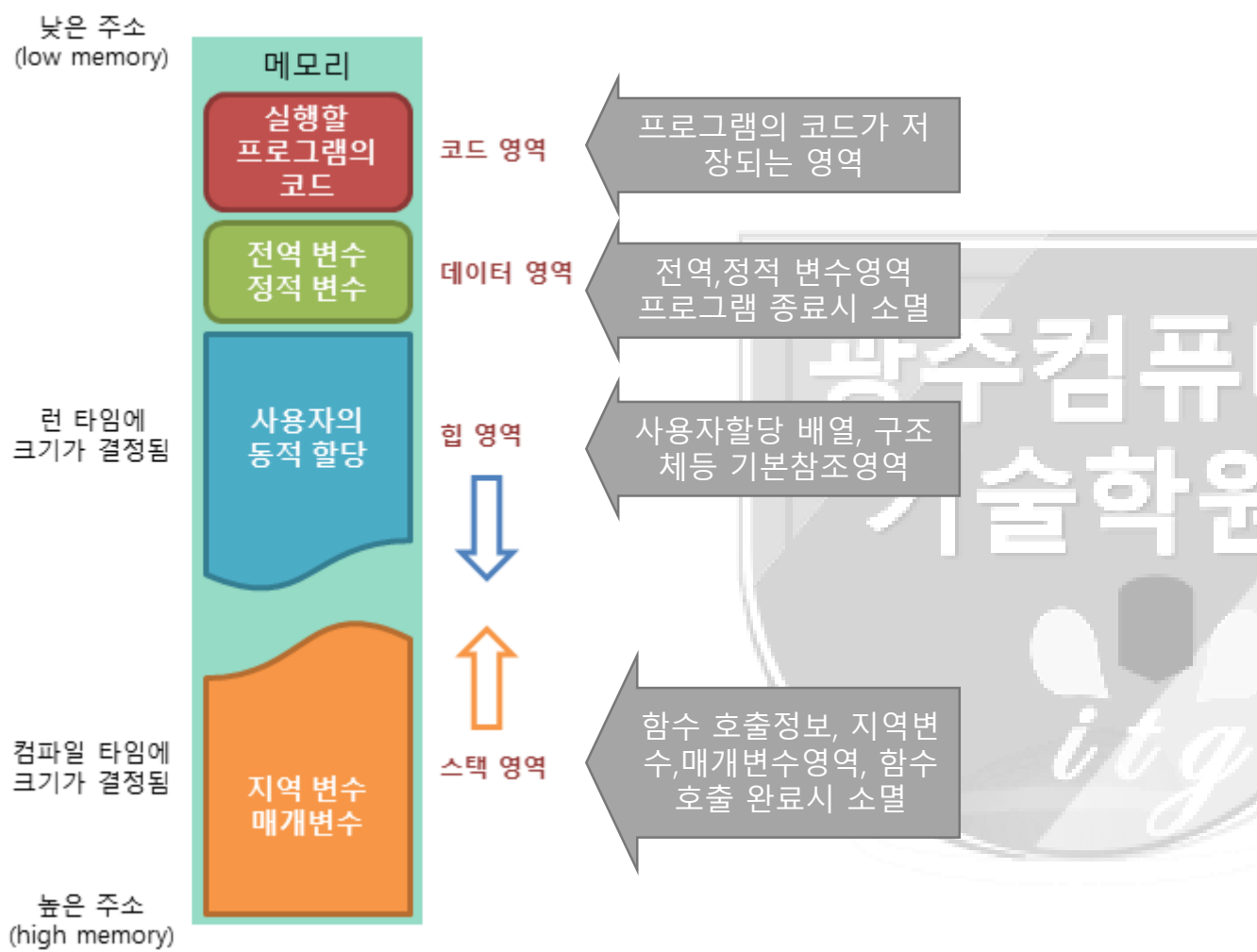
```
int arr[2][3] =           // 배열의 선언
{
    {10, 20, 30},
    {40, 50, 60}
};
```

```
int (*pArr)[3] = arr;     // 배열 포인터의 선언
```

```
printf("%d\n", arr[1][1]); // 배열 이름으로 참조
printf("%d\n", pArr[1][1]); // 배열 포인터로 참조
```

출력
50
50

배열접근 40 배열접근 35 배열접근 35



malloc() 함수

malloc() 함수는 프로그램이 실행 중일 때 사용자가 직접 힙 영역에 메모리를 할당할 수 있게 해줍니다.

malloc() 함수의 원형은 다음과 같습니다.

원형

```
#include <stdlib.h>
void *malloc(size_t size); // 주소값 반환, 없을시 null pointer
```

free() 함수

free() 함수는 힙 영역에 할당받은 메모리 공간을 다시 운영체제로 반환해 주는 함수입니다.

원형

```
#include <stdlib.h>
void free(void *ptr);
```

free() 함수를 사용하여 다 사용한 메모리를 해제해 주지 않으면, 메모리가 부족(메모리누수) 발생

calloc() 함수

malloc() 함수와 다른 점은 할당하고자 하는 메모리의 크기를 두 개의 인수로 나누어 전달

원형

```
#include <stdlib.h>
void *calloc(size_t nmemb, size_t size);
```

예제

```
int i; int arr_len = 3; int* ptr_arr;
ptr_arr = (int*) malloc(arr_len * sizeof(int)); // 메모리의 동적 할당
if (ptr_arr == NULL) // 메모리의 동적 할당이 실패할 경우
{
    printf("메모리의 동적 할당에 실패했습니다.\n");
    exit(1);
}

printf("동적으로 할당받은 메모리의 초기값은 다음과 같습니다.\n");
for (i = 0; i < arr_len; i++)
{
    printf("%d ", ptr_arr[i]);
}
free(ptr_arr); // 동적으로 할당된 메모리의 반환
```

실행 결과

동적으로 할당받은 메모리의 초기값은 다음과 같습니다.
0 0 0

realloc() 함수

realloc() 함수는 이미 할당된 메모리의 크기를 바꾸어 재할당

원형

```
#include <stdlib.h>
void *realloc(void *ptr, size_t size);
```

문자열(string)

큰따옴표 " " 를 사용해 표현되는 문자열 상수,
메모리에 저장된 연속된 character의 배열

```
char str01[] = "This is a string."; // 크기를 지정하지 않은 문자열 변수 선언
char str02[7] = "string";           // 크기를 지정한 문자열 변수 선언
```

널(NULL) 문자

문자열의 끝을 프로그램 알려주는 역할이고 ' \0 ' 으로 표시하고 아스키코드값은 0

예제

```
int str_len = 0;
char str[] = "string";
while (str[str_len] != '\0') // 널 문자가 나올 때까지 길이를 증가함
{
    str_len++;
}
printf("이 문자열의 길이는 %d입니다.\n", str_len);
```

실행 결과

이 문자열의 길이는 6입니다.

fgets() 함수

fgets() 키보드 또는 파일에서 문자열을 입력

puts() 함수

puts() 표준 출력 모니터에 문자열,자동줄바꿈,서식지정 안됨

fputs() 함수

모니터 또는 파일 문자를 출력/저장

문자열 처리 함수

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

strlen() 함수

문자열의 길이 반환, 마지막 문자인 널 문자는 제외

원형

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

```
size_t strlen(const char *s);
```

예제

```
char str[] = "C언어";  
printf("이 문자열의 길이는 %d입니다.\n", strlen(str));
```

실행 결과

이 문자열의 길이는 7입니다.(utf-8 한글 3byte)

strcat() 함수와 strncat() 함수

문자열에 다른 문자열을 연결

strcat(source,target) - 문자열이 저장된 배열공간을 넘어 오버플로우 발생가능

strncat(source,target,targetN) - 널문자 또는 target 개의 최대길이에 도달하면 중지, 오버플로우방지

strcpy() 함수와 strncpy() 함수

문자열을 복사

첫 번째 인수 배열에 두 번째 인수 문자열을 복사

strcpy 오버플로우 가능성으로 strncpy(배열,복사문자,복사문자최대크기) 사용

자바의 equal 이랑 같아요

strcmp() 함수와 strncmp() 함수

문자열의 내용을 비교하는 함수입니다.

양수 - 첫번째 더 큰경우, 0 - 내용일치, 음수 - 두번째가 더 큰경우

atoi() - int, atol() - long, atoll() - longlong, atof() - double 함수

전달된 문자열을 해당 타입의 숫자로 변환

toupper() 함수와 tolower() 함수

영문자를 모두 대문자나 소문자로 변환

구조체

구조체(structure type) - 사용자 정의 타입(자료구조의 표현체)
다양한 타입의 변수 집합을 하나의 타입으로 나타낸 것
변수를 구조체의 멤버(member) 또는 멤버 변수(member variable)라고함

문법

```
struct 구조체이름 구조체변수이름;  
struct 구조체이름{  
    멤버변수타입 멤버변수이름;  
    멤버변수타입 멤버변수이름;  
    ...};
```

구조체는 복합타입 입니다.
각기 크기가틀려 배열보다는 변수에 가깝습니다.

```
struct book  
{  
    char title[30];  
    char author[30];  
    int price;  
} my_book;
```

구조체 타입 book 입니다.

구조체 타입은 char 배열, int 타입을 갖습니다.

구조체 타입을 갖은 변수이름입니다.

```
struct book  
{  
    char title[30];  
    char author[30];  
    int price;  
}  
struct book my_book
```

변수는 나중에 선언해도 됩니다.

```
typedef struct [book]  
{  
    char title[30];  
    char author[30];  
    int price;  
} BBB;  
[struct] BBB my_book;
```

typedef 는 타입을 재정의 할수 있습니다.
위는 타입을 BBB로 다시 정의하였습니다.
재정의시 구조체 타입이름은 book는 생략해도 됩니다.
typedef 로 재정의된 구조체는 변수선언시
struct 를 생략할 수 있습니다.

* 하나의 집합체로된 변수로 생각하면 됩니다.
변수 포인터 방식을 그대로 따릅니다.
말 그대로 하나의 타입을 갖는 덩어리



구조체 초기화

```
my_book = {.title="html", .author="홍", .price=20000}
```

```
my_book = {"html", "홍", 20000}
```

멤버변수 순서에 상관없이 이름으로 초기화

배열처럼 초기화, 순서대로 초기화되며 나머지는 0으로 초기화

구조체 접근

접근은 . 으로 접근후 데이터를 인출

주소방식으로 접근시 -> 로 접근후 데이터 인출

예제

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

```
struct book
{
    char title[30];
    char author[30];
    int price;
};
```

실행 결과

첫 번째 책의 제목은 HTML과 CSS이고, 저자는 홍길동이며, 가격은 28000원입니다.
두 번째 책의 제목은 Java language이고, 저자는 이며, 가격은 30000원입니다.

문자열 배열은 초기화가 안되어 있으므로 변수 선언
시 초기화 해야합니다.(문자열상수)

```
int main(void)
{
    struct book my_book = {"HTML과 CSS", "홍길동", 28000};
    struct book java_book = {.title = "Java language", .price = 30000};

    printf("첫 번째 책의 제목은 %s이고, 저자는 %s이며, 가격은 %d원입니다.\n",
        my_book.title, my_book.author, my_book.price);
    printf("두 번째 책의 제목은 %s이고, 저자는 %s이며, 가격은 %d원입니다.\n",
        java_book.title, java_book.author, java_book.price);
    return 0;
}
```

선언후 나중에 초기화 방법

```
struct book{
    char* title;
    char* author;
    int price;
};
struct book my_book;
my_book.title="html";
my_book.author="홍길동";
my_book.price=30000;
```

typedef 타입재정의시

예제

```
#include <stdio.h>
typedef struct{
    char title[30];
    char author[30];
    int price;
} TEXTBOOK;
```

```
int main(void){
    TEXTBOOK my_book = {"HTML과 CSS", "홍길동", 28000};
    TEXTBOOK java_book = {.title = "Java language", .price = 30000};

    printf("첫 번째 책의 제목은 %s이고, 저자는 %s이며, 가격은 %d원입니다.\n",
        my_book.title, my_book.author, my_book.price);
    printf("두 번째 책의 제목은 %s이고, 저자는 %s이며, 가격은 %d원입니다.\n",
        java_book.title, java_book.author, java_book.price);
    return 0;
}
```

실행 결과

첫 번째 책의 제목은 HTML과 CSS이고, 저자는 홍길동이며, 가격은 28000원입니다.
두 번째 책의 제목은 Java language이고, 저자는 이며, 가격은 30000원입니다.

구조체를 배열에

```
struct book text_book[3] =
{
    {"국어", "홍길동", 15000},
    {"영어", "이순신", 18000},
    {"수학", "강감찬", 10000}
};

puts("각 교과서의 이름은 다음과 같습니다.");
printf("%s, %s, %s\n", text_book[0].title, text_book[1].title, text_book[2].title);
```

실행 결과

각 교과서의 이름은 다음과 같습니다.
국어, 영어, 수학



구조체 포인터

void structImple(){//구조체는 배열보다 변수 타입에 가깝다.

//char test[30]; //문자열상수는 선언후 초기화가 되지 않습니다.

//test="반갑습니다";//컴파일 오류

char* test;

test = "안녕하세요";//포인터를 잡고 문자열을 넣어줍니다.

typedef struct {

char* title;

char* author;

int price;

} BOOK;

BOOK my_book;

my_book.title="html";

my_book.author="홍길동";

my_book.price=30000;//또는

BOOK you_book = {"c언어","김씨"};

you_book.price=40000;

BOOK* tbook = &you_book;

printf("구조체포인터 접근 -> : %s\n",tbook->title);

printf("구조체포인터 접근 * : %s",(*tbook).author);

}

타입재선언으로 구조체 타입 재정의

char* 참조형으로 my_book 선언후 나중에 초기화 가능

char title[] 형태라면 you_book 선언과 동시에 문자열초기화

구조체 포인터 선언(구조체는 복합변수타입으로 & 주소로

구조체 포인터로 구조체 멤버접근

구조체포인터 접근 -> : c언어

구조체포인터 접근 * : 김씨

12. 다음은 C언어 소스 코드이다. 출력 결과를 쓰시오.

```
#include <stdio.h>
void main(){
    int i,j; int temp;
    int a[5] = {75,95,85,100,50};
    for(i=0; i<4; i++){
        for(j=0; j<4-i; j++){
            if(a[j] > a[j+1]){
                temp=a[j];
                a[j] = a[j+1];
                a[j+1] = temp;
            }
        }
    }
    for(i=0; i<5; i++){
        printf("%d", a[i]);
    }
}
```

답기

50, 75, 85, 95, 100

2. 다음은 C언어 소스 코드이다. 출력 결과를 쓰시오.

```
#include <stdio.h>
void main(){
    int i=0, c=0;
    while (i<10){ i++; c*=i; }
    printf("%d",c);
}
```

답기
0

13. 다음은 C언어 소스 코드이다. 출력 결과를 쓰시오.

```
#include <studio.h>
int r1(){
    return 4;
}
int r10(){
    return (30+r1());
}
int r100(){
    return (200+r10());
}
int main(){
    printf("%d\n", r100());
    return 0;
}
```

18. 다음은 C언어 소스 코드이다. 출력값을 쓰시오.

```
#include <stdio.h>
void main(){
    char *p = "KOREA" ;
    printf("%s\\n" , p);
    printf("%s\\n" , p+3);
    printf("%c\\n" , *p);
    printf("%c\\n" , *(p+3));
    printf("%c\\n" , *p+2);
}
```



배열내부에 접근하였다.

닫기
KOREA
EA
K
E
M

15. 다음은 C언어 프로그램이다. 실행 결과를 쓰시오.

```
#include <stdio.h>
struct good {
    char name[10];
    int age;
};
void main(){
    struct good s[] = {"Kim",28},{"Lee",38},{"Seo",50},{"Park",35};;
    struct good *p;
    p = s;
    p++;
    printf("%s\n", p-> name);
    printf("%s\n", p-> age);
}
```

닫기
Lee 38



16. 다음은 C언어에 관한 소스코드이다. 실행 결과값을 작성하시오.

```
int main(){
    int res;
    res = mp(2,10);
    printf("%d",res);
    return 0;
}

int mp(int base, int exp) {
    int res = 1;
    for(int i=0; i < exp; i++){
        res = res * base;
    }

    return res;
}
```



달기
1024

18. 다음은 C언어 문제이다. 출력값을 작성하시오.

```
int main(){
    int ary[3];
    int s = 0;
    *(ary+0)=1;
    ary[1] = *(ary+0)+2;
    ary[2] = *ary+3;
    for(int i=0; i<3; i++){
        s=s+ary[i]
    }
    print("%d",s);
}
```



닫기
8

12. 다음 C언어에 대한 알맞는 출력값을 쓰시오.

```
#include <stdio.h>

int main(){
    int *arr[3];
    int a = 12, b = 24, c = 36;
    arr[0] = &a;
    arr[1] = &b;
    arr[2] = &c;
    printf("%d\n", *arr[1] + **arr + 1);
}
```


17. 다음 C언어에 대한 알맞는 출력값을 쓰시오.

```
#include <stdio.h>
struct jsu {
    char name[12];
    int os, db, hab, hhab;
};
int main(){
    struct jsu st[3] = {"데이터1", 95, 88},
                      {"데이터2", 84, 91},
                      {"데이터3", 86, 75}};

    struct jsu* p;
    p = &st[0];
    (p + 1)->hab = (p + 1)->os + (p + 2)->db;
    (p + 1)->hhab = (p+1)->hab + p->os + p->db;
    printf("%d\n", (p+1)->hab + (p+1)->hhab);
}
```

8. 다음 소스코드에 대한 출력값을 작성하시오.

답기
2

```
struct A{
    int n,
    int g
}
int main(){
    A a = new A[2]
    for(i=0; i <2; i++) {
        a[i].n = i,
        a[i].g=i+1
    }
    System.out.printf(a[0].n + a[1].g);
}
```

[CS](#)

15. 다음 C언어에서 출력에 대한 알맞은 답을 작성하시오..

답기
10

```
#include <stdio.h>
int len(char*p);
int main(){
    char*p1 = "2022";
    char*p2 = "202207";
    int a = p1;
    int b = p2;
    printf("%d", len(a) + len(b));
}
int len(char*p){
    int r = 0;
    while(*p != '\0'){
        p++;
        r++;
    }
    return r;
}
```

16. 다음 C언어 코드에서 알맞는 출력값을 작성하시오.

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    int a[4] = {0, 2, 4, 8};
    int b[3] = {};
    int i = 1;
    int sum = 0;
    int *p1;
    for (i; i < 4; i++) {
        p1 = a + i;
        b[i-1] = p1 - a[i-1];
        sum = sum + b[i-1] + a[i];
    }
    printf("%d", sum);
    return 0;
}
```

14. 다음 소스코드에서 입력값이 5가 들어왔을때 출력되는 값을 작성하시오

답기
120

```
#include <stdio.h>

int func(int a) {
    if (a <= 1) return 1;
    return a * func(a - 1);
}

int main() {
    int a;
    scanf("%d", &a);
    printf("%d", func(a));
}
```

[CS](#)

15. 다음 중, 괄호 () 안에 들어갈 연산자를 써서 정수를 역순으로 출력하는 알맞는 답을 작성하시오.

```
#include <stdio.h>
int main() {
    int number = 1234;
    int div = 10;
    int result = 0;
    while (number ( 1 ) 0) {
        result = result * div;
        result = result + number ( 2 ) div;
        number = number ( 3 ) div;
    }
    printf("%d", result);
    return 0;
}
```

결과 : 4321

답기

1. >

2. %

3. /

- result 는 역으로 된 숫자이고 number은 뒤부터 하나씩 추출한다.
- 뒤부터 추출할때는 나머지를 구하고 계속 10으로 나누어 남은 숫자를 저장한다.

19. 다음 소스코드가 실행할 때의 출력값을 작성하시오.

답기
29

```
#include <stdio.h>

int isPrime(int number) {
    int i;
    for (i=2; i<number; i++) {
        if (number % i == 0) return 0;
    }
    return 1;
}

int main(void) {
    int number = 13195, max_div=0, i;
    for (i=2; i<number; i++)
        if (isPrime(i) == 1 && number % i == 0) max_div = i;
    printf("%d", max_div);
    return 0;
}
```

2~13195 중 2와 자신이외의 어떤수에도 나누어 떨어지지 않는 수 중 - 소수

소수로 소인수분해

$13195/5=2639 \%0$

$2639/7 = 377 \%0$

$377/13 = 29 \%29$

소인수 : 5,7,13,29

8. 다음 소스코드에 대한 출력값을 작성하시오.

닫기
2

```
struct A{
    int n,
    int g
}

int main(){
    struct A a[2];
    for(int i=0; i <2; i++) {
        a[i].n = i;
        a[i].g=i+1;
    }
    printf("%d",a[0].n + a[1].g);
}
```



15. 다음 C언어에서 출력에 대한 알맞은 답을 작성하시오..

```
#include <stdio.h>
int len(char*p);
int main(){
    char*p1 = "2022";
    char*p2 = "202207";

    int a = len(p1);
    int b = len(p2);
    printf("%d", a + b);
}
int len(char*p){
    int r = 0;
    while(*p != '\0'){
        p++;
        r++;
    }
    return r;
}
```

닫기
10



16. 다음 C언어 코드에서 알맞는 출력값을 작성하시오

답기
22

```
#include <stdio.h>
int main(int argc, char *argv[]) {
    int a[4] = {0, 2, 4, 8};
    int b[3];
    int sum = 0;
    int *p1;
    for (int i = 1; i < 4; i++) {
        p1 = a + i;
        b[i-1] = *p1 - a[i-1];
        sum = sum + b[i-1] + a[i];
    }
    printf("%d", sum);
    return 0;
}
```

