C 언어 이론 및 실습 2

Human-Robot Interaction Laboratory

KYUNG HEE UNIVERSITY

제어문

- C언어에서 제어문의 종류
  - 루프문 : 반복 처리를 표현하는 제어문
    while, do ... while, for
  - 선택문 : 조건식에 따라 선택하여 처리하는 제어문
    if, if ... else, switch
  - 보조 제어문 : break, continue, goto, return

- 제어문 사용 시 유의사항 :
  - 구조적인 프로그래밍을 한다.
  - 실행조건은 조건식으로 한다.
  - 조건식은 &&나 || 로 연결 가능
  - 문이 여러 문장으로 된 경우는 { ... } 사용

제어문

>> 점프문 (goto label 문)

- 프로그래밍에 있어서 반복되는 문장이나 선택적으로 실행할 수 있는 두 개 이상의 문장 블록이 포함 되어 있어서 어떤 언어든 프로그램이 다른 시점에서 진행할 수 있게 하는 명령어문

- 대표적인 예 : goto label

```
int i=1, s=0;                          ①
start :                                ②
if (i>100)                             ③
   goto end ;                          ④
s += i++ ;                             ⑤
goto start ;                           ⑥
end :                                  ⑦
printf("Total : %d ₩n", s);            ⑧
```

[출력결과]

Total : 5050

① 변수 초기화
② start 라벨 : goto start를 만나면 점프하는 곳
③ i가 101이상이면 ④번 실행 그렇지 않으면 ⑤번 실행
④ if 문이 참이면 실행 end 라벨 (⑦번)로 이동
⑤ s=s+i, i=i+1
⑥ start 라벨 (②번)로 이동
⑦ end 라벨 : goto end를 만나면 점프하는 곳
⑧ 결과물 출력

- 위 예제의 문제점 : 아무 곳이나 점프가 가능하여 서로 뒤엉켜 있다

- C 언어에서는 goto 문을 무시해도 구조적인 프로그램이 가능

제어문

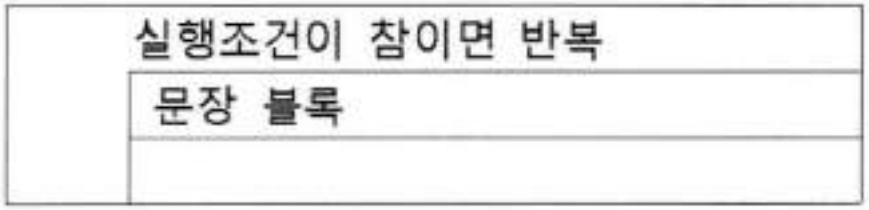
>> 루프문 - while

- 형식

```
while (식)
  실행문
```

- 설명
  - 식의 값이 참이면 식의 값이 거짓이 될 때까지 문을 반복 처리
  - 식의 값이 항상 참 (0이외의 수)이면 문은 무한 반복
  - 식의 값이 거짓 (0)이면 문은 한번도 반복되지 않음
  - 식은 문을 실행시킬 수 있는 실행 조건 (종료 조건이 아님)
  - 실행문은 어떠한 실행문도 가능 (빈 문장도 가능)
  - 실행문은 한 개 이상의 C 언어문장으로 가능
  - 실행문이 여러 개의 문장인 경우 { ... }으로 표현

- 구조도



제어문

>> 루프문 - while

- **예시**

예) 빈 문장

```
int x=4 ;
while (++x < 7)
      ;
printf ("%d₩n", x);
```

[출력결과]

7

예) 문이 1개의 문장으로만 구성된 경우

```
int x=4 ;
while (x < 7)
    ++x ;
printf ("%d₩n", x);
```

[출력결과]

7

예) 문이 여러 개의 문장으로만 구성된 경우

```
int x=4 ;
while (x < 7)
{
    ++x ;
    printf (" %d : ", x);
}
```

[출력결과]

5 : 6 : 7 :

예) 식의 값이 항상 참이므로 무한 루프가 되는 경우

```
while (1)
   printf ("Otto₩n", x);
```

[출력결과]

Otto
Otto
Otto
...

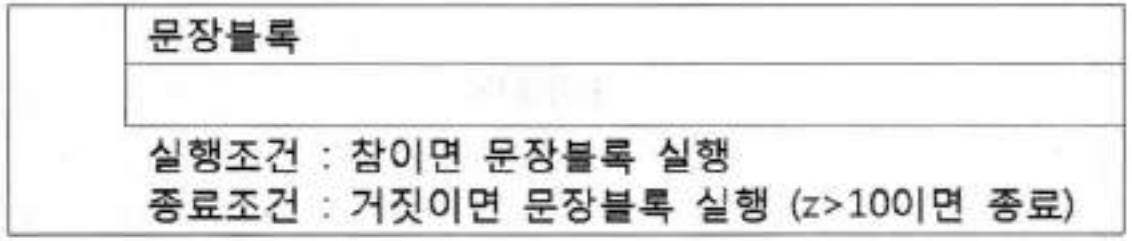
제어문

>> 루프문 - do ... while

- 형식

```
do
   실행문
while (식)
```

- 설명
  - 문과 식은 while식과 동일
  - do ... while 는 문이 적어도 1번 실행된 후 조건 검사
  - while 루프문은 조건식이 거짓인 경우 문을 실행하지 않음
- 구조도



종료조건인 경우 구조도 조건과 실제 조건식은 반대여야 한다.

```
do
{
    문장블럭
} while (z<=10);
```

제어문

>> 루프문 - do ... while

- **예시**

예)

```
int x=4 ;
do
{
    ++x ;
    printf (" %d : ", x);
} while (x < 7);
```

[출력결과]

5 : 6 : 7 :

제어문

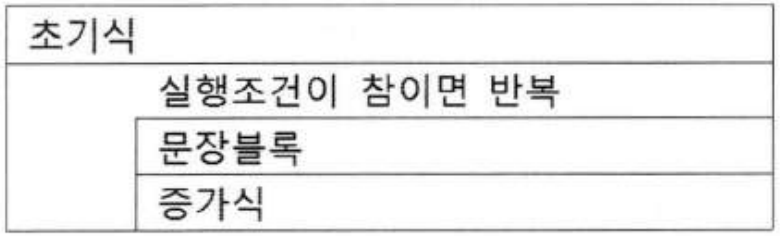
>> 루프문 - for

- 형식

```
for ([초기식] ; 식 ; [증가식])
    실행문
```

- 설명
  - 실행문의 반복 처리 전에 먼저 초기식을 실행
  - 식의 값이 참이면 실행문을 반복 실행
  - 실행문을 실행한 다음 증가식을 실행
  - 식이 거짓이 될 때 까지 실행문과 증가식을 반복실행
  - [ ] 내의 식은 생략가능

- 구조도



제어문

>> 루프문 - for

- **예시**

예)

```
int i, sum=0 ;
for (i=1; i<=100; i++)
{
    sum=sum+i ;
}
printf ("sum = %d : ", sum);
```

[출력결과]

sum = 5050

예) 증가식과 초기식의 생략

```
int x=4 ;
for ( ; ++x < 7 ; )
    ;
printf ("%d₩n", x);
```

[출력결과]

7

예) 실행 조건식이 없는 경우 (무한루프)

```
for ( ; ; )
    printf ("Otto₩n", x);
```

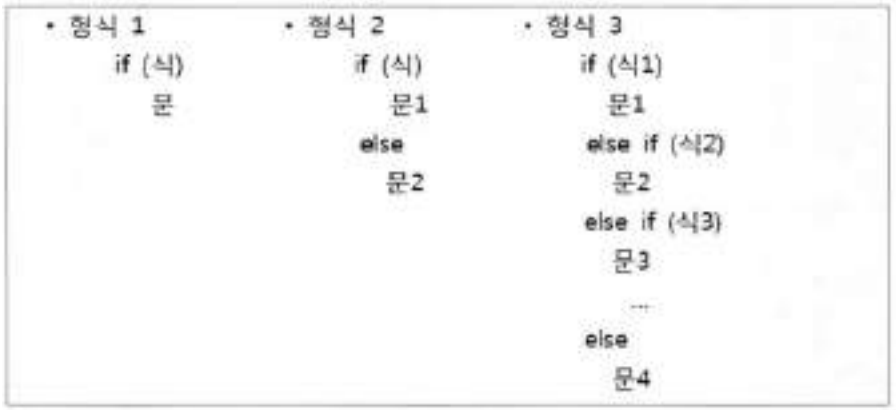
[출력결과]

Otto
Otto
...

제어문

>> 선택문 - if

- 형식



- 설명
  - 형식 1에서 식이 참이면 문을 실행
  - 형식 1에서 식이 거짓이면 문을 실행하지 않고 다음 문장을 실행
  - 형식 2에서 식이 참이면 문1을 거짓이면 문2를 실행
  - 형식 3에서 식이 거짓이면 다음 식의 조건을 판별
  - 형식 3에서 식이 참이면 그 식에 해당되는 문을 실행 후 나옴
  - 형식 3에서 식이 모두 거짓이면 문4를 실행
  - 실행문의 문장이 여러 개가 있는 경우 { ... }으로 표현
  - 식은 조건식이 와야함

제어문

>> 선택문 - if

- 구조도

- 예시

예)

```
char c;
printf("Yes of No? (enter 'y' or 'n') ");
scanf ("%c", &c);
if (c=='y' || c=='Y') printf ("₩n Yes ₩n");
else if (c=='n' || c=='N') printf ("₩n No ₩n");
else printf("₩nWrong character!!₩n");
```

[입력] Y	[출력결과] Yes
[입력] n	[출력결과] No
[입력] a	[출력결과] Wrong character!!

예) 문장이 여러 개 일 때 { ... } 이 없는 경우

```
if(x>0)
    y++;
    z--;        ①
else
    z++;
```

→

```
if(x>0)
{
    y++;
    z--;
}
else
    z++;
```

① 컴파일 오류 메시지 발생

예) 식이 조건식이 아닌 경우

```
int x=4 ;
if (x=3)                ①
    x=1;
else
    x=17;
printf ("%d₩n", x);
```

[출력결과]

1

① x=3은 참으로 인식 : 항상 x=1 만 실행

제어문

>> 선택문 - switch

- 형식

```
switch (식)
{
    case 상수식1 :
        문1
        break;
    case 상수식2 :
        문2
        break;
    ...
    defalut :
        문
}
```

- 설명
  - 식에 해당하는 타입은 정수형이며 값을 할당할 수 있는 식
  - 상수식1, 2는 정수형 상수
  - 식과 상수식을 비교하여 일치하면 그에 해당하는 문을 수행
  - Break가 있으면 switch 문을 빠져 나옴
  - Break가 없으면 해당 case 문에서부터 switch문 끝까지 모든 문을 실행
  - 상수식과 맞는 것이 없으면 default문 실행

제어문

>> 선택문 - switch

- 구조도

switch (식)			
상수식 1	상수식 2	...	default
문장블럭	문장블럭	문장블럭	문장블럭

- 예시

예)

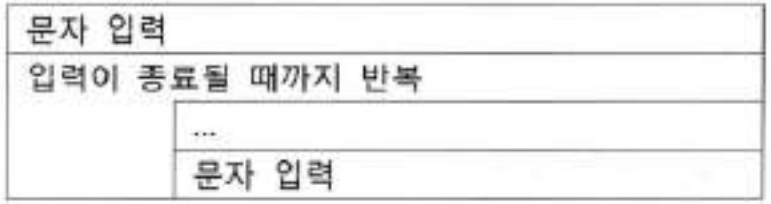
```
char c;
printf("Yes of No? (enter 'y' or 'n') ");
scanf ("%c", &c);
switch (c)
{
      case 'Y' :
      case 'y' : printf ("₩n Yes ₩n"); break;
      case 'N' :
      case 'n' : printf ("₩n No ₩n"); break;
      default : printf ("₩n Wrong character!! ₩n");
}
```

[입력] Y	[출력결과] Yes
[입력] n	[출력결과] No
[입력] a	[출력결과] Wrong character!!

구조도 변환의 문제점

- 루프 실행 후 조건 검사
  - 구조도의 종료 조건은 논리부정을 이용하여 do-while 구문의 실행조건으로의 변경이 필요 (종료조건 = !(실행조건))
    → 5p do ... while 문의 구조도 참조
  - 논리 연산자의 부정을 이용하여 간소화 (드모르간 법칙)

- 루프 구조를 이용한 문자 단위의 데이터 읽기
  - 읽기 루프 구조도



  - C언어 프로그래밍

```
int c;
c = getchar();
while (c!=EOF)
{
    ...
    c = getchar();
}
```

```
int c;
while( (c = getchar()) !=EOF)
{
    ...
}
```

구조도 변환의 문제점

- break문
  - switch문 : case문 실행 후 switch 문 빠져 나옴 (구조도 가능)
  - 루프문 : 현재 루프문을 빠져 나옴 (구조도 불가)
    → 구조화된 code로 변경 필요

예)

```
while (x<100)
{
        ...
        if (z==5)
                break;
        z++;
        ...
}
```

→

```
while (z!=5 && x<100)
{
        ...
        z++;
        ...
}
```

구조도 변환의 문제점

- continue문
  - 루프문 : 루프 실행을 중단 후 다음 반복 시작(구조도 불가)
    → 구조화된 code로 변경 필요

예) 1부터 100까지 홀수만 더하는 프로그램

```
int i, s=0;
for( i=1; i<=100 ; i++)
{
        if (i%2==0)
                continue;
        s=s+i;
}
printf("Sum of Odd : %d",s);
```

[실행결과]
Sum of Odd : 2500

→

```
int i, s=0;
for( i=1; i<=100 ; i++)
{
        if (i%2)
                s=s+i;
}
printf("Sum of Odd : %d",s);
```

[실행결과]
Sum of Odd : 2500

벡터와 배열

>> 벡터(1차원 배열)

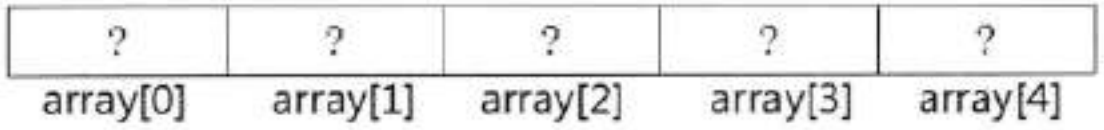
- 정의
  - 같은 데이터 타입을 가진 여러 요소로 구성된 데이터 집합체

- 특징
  - 순서가 있음
  - 모든 요소는 메모리를 차지하고 있음
  - 각 요소는 정수 인덱스를 통해 지정될 수 있으며, 지정된 후에는 보통의 변수처럼 사용가능

- 배열 선언

type name[size];	type : 배열 각 요소의 타입
	name : 배열로 선언되는 변수 이름
	size : 배열의 요소 개수

  - 배열 선언시 초기에는 각 요소의 값 미정(단순 변수와 동일)
  - 예) int array[5];



KYUNG HEE UNIVERSITY

벡터와 배열

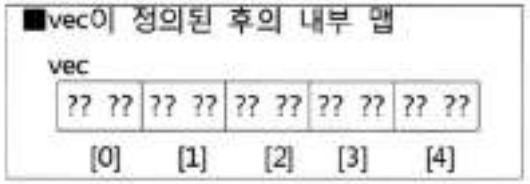
>> 벡터(1차원 배열)

- **예제 및 설명**

```
int z = 4; //①
int vec[5]; //②
vec[0] = 13; //③
printf("%d₩n", vec[0]); //④
vec[z] = 3; //⑤
vec[z / 2] = vec[z] + 1; //⑥
```

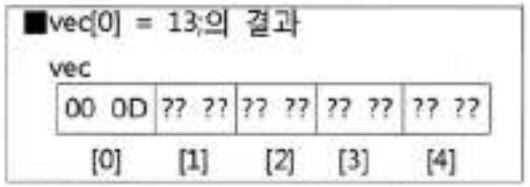
① 정수형 변수 선언 및 초기화

② 5개의 정수 요소로 된 벡터 선언



- 연속적인 인덱스가 0부터 시작함에 주의

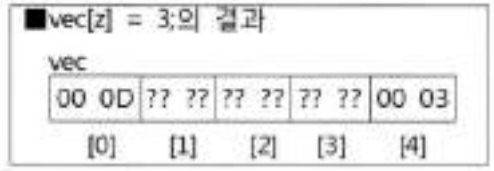
③ 첫 번째 요소에 13의 값을 대입



- 벡터의 내용은 16진수 표기법으로 표현

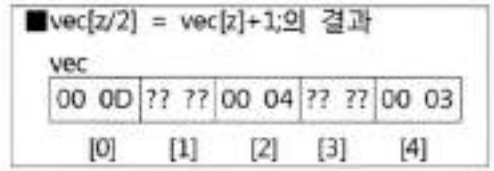
④ 출력

⑤ 다섯 번째 요소에 3의 값을 대입



- 벡터 요소를 지정할 때 인덱스로 변수 사용 가능
- 벡터를 정의할 때는 요소의 수로 변수 사용 불가능

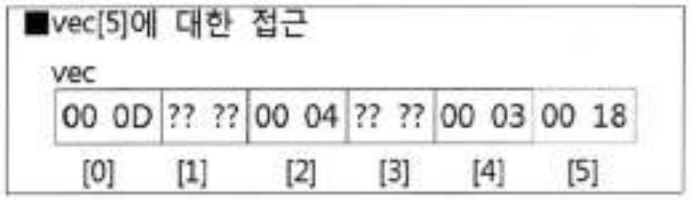
⑥ 마지막 요소의 값에 1을 더한 값을 세 번째 요소에 대입



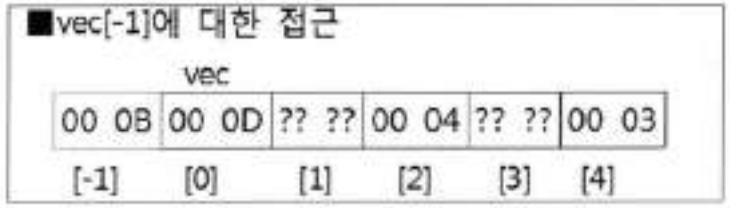
- 일반 변수가 나타날 수 있는 모든 곳에 벡터 요소 사용 가능

- **고려 사항**

① vec[5] = 24;



② vec[-1] = 11;



- 벡터에 할당된 메모리 영역에 속하지 않는 메모리 장소 접근
  → 에러 발생

벡터와 배열

>> 벡터(1차원 배열)

- 요약
  - vname이라는 이름을 가지고 type이라는 데이터 타입의 요소를 num개 가진 벡터
    → type vname[num];
    * vname은 변수 이름에 관한 C의 규칙에 따라 생성
    * type은 각 요소의 타입이며 도든 자체 정의 데이터 타입이 허용
    * num은 요소의 수로써 양의 정수 상수 표현식만 가능

불가능	가능
int num = 5; int vec[num];	int vec[2 * 3 - 1];

  - 이렇게 정의된 벡터의 한 요소는 vname[index]와 같은 형태로 프로그램의 어디에서나 사용가능

벡터와 배열

>> 벡터(1차원 배열)

- 예시 (한 줄의 텍스트를 입력하여 역순으로 출력)

```
char line[80+1]; //①
int i, c; //②
for(i=0; i<80 && (c=getchar())!='₩n' && c!=EOF; ++i) //③
    line[i]=c;
line[i]='₩0';
for(--i; i>=0; --i) //④
    putchar(line[i]);
putchar('₩n');
```

① 배열 선언
② 변수 i는 각 벡터 요소를 지정하기 위한 인덱스로 사용, 변수 c는 읽어들인 문자를 저장
③ for문 : 벡터 line에 최대 80개의 문자를 저장
④ for문 : 역순으로 읽어들인 문자를 출력

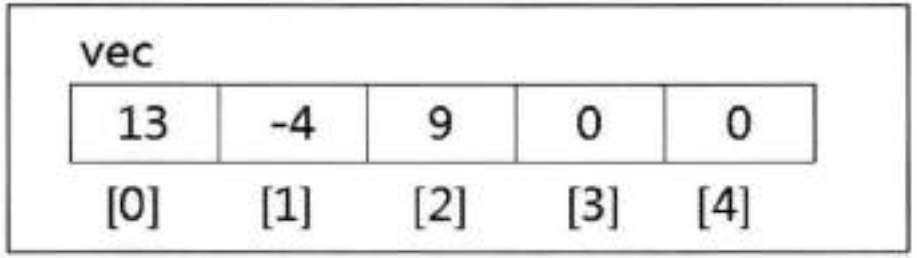
※ 의문 사항
- 입력된 줄 바꿈 문자는 어떻게 되었는가?
☞ 줄 바꿈 문자는 읽은 후 c에는 저장되지만 벡터의 요소에는 저장되지 않으며, 텍스트만을 line에 저장하려 할 때 유용
- 최대 문자수가 왜 벡터의 크기인 81이 아니고 80인가?
☞ 문자열 끝 표시자인 '₩0'을 마지막에 입력된 텍스트 문자 바로 다음 요소에 대입하여, 문자열의 끝을 수동으로 표시

- 결론
  - C에는 문자열 변수에 대한 데이터 타입이 없다
  - 문자열의 끝은 마지막 문자 바로 다음에 ASCII 0 을 저장하여 표시
  - char 벡터의 크기는 저장될 문자의 최대수에 의존하며, 문자열 끝 표시자를 저장할 요소 추가

벡터와 배열

벡터의 초기화

- int vec[5] = {13, -4, 9}
  - 처음 세 개의 요소에 차례대로 초기화가 이루어지고, 마지막 두 개의 요소는 자동적으로 0으로 초기화됨



  - 벡터가 가지고 있는 요소보다 더 많은 요소를 초기화 할 수 없음

- int vec[] = {13, -4, 9, 108, 25}
  - 벡터의 모든 요소를 초기화시킨다면 요소의 개수를 명시하지 않아도 됨
  - int vec[5] = {13, -4, 9, 108, 25} 와 동일
- int vec[5];
  - vec[0] = 13
  - vec[1] = -4
  - 벡터가 정의될 때 초기화되지 않았으면 벡터의 각 요소에 대해 별도의 대입문을 작성
- char line[] = {'O', 't', 't', 'o', '₩0'};
  - char line[] = "Otto"; 와 동일

벡터와 배열

>> 벡터 인수 전달

```
#include <stdio.h>
void set(int vec[], int n) //①
{
	int i;
	for(i=0; i<n; ++i)
		vec[i] = i+1;
}
int main(void)
{
	int i;
	int numbers[5];
	set(numbers, 5); //②
	for(i=0; i<5; ++i)
		printf("%d ", numbers[i]);
	putchar('₩n');
	return 0;
}
```

① 함수 정의 시, 빈 대괄호 []를 가진 벡터를 매개변수로 정의

② 함수 호출 시, 벡터 전체를 인수로 전달하려면 대괄호 []를 제외한 벡터의 이름을 이용

- 벡터 numbers는 함수 set()이 호출될 때 vec에 인수로 전달
- 두 번째 인수인 n을 통해 요소의 수도 전달(벡터의 실제 크기에 대한 정보 전달)

벡터와 배열

벡터 인수 전달

- 문자열을 읽어들이는 예

```
char line[80+1];
gets(line); //①
...
fgets(line, 81, stdin); //②
```

① 줄 바꿈 문자는 저장되지 않고 '₩0'으로 대치
② 첫 번째 인수(line) : 읽어들인 문자열이 저장될 벡터
두 번째 인수(81) : 벡터의 크기
세 번째 인수(stdin) : 표준 입력
- fgets() : 파일로부터 줄 단위로 읽어들일 수 있음
- 줄 바꿈 문자를 벡터에 저장하고 마지막에 '₩0'을 추가

- gets()와 fgets()
  - gets()는 호출하기가 간단하지만, 인수로 받은 벡터의 크기를 알 수 없음
  - fgets()는 줄 바꿈 문자를 벡터에 저장하는 것이 부자연스러움

벡터와 배열

>> 벡터 인수 전달

- getline()함수

```
#include <stdio.h>
int getline(char s[], int lim) { //①
    int i, c;
    for(i=0; i<lim-1 && (c=getchar())!=EOF && c!='₩n'; ++i)
        s[i] = c; //②
    s[i] = '₩0';
    fflush(stdin); //③
    if(i==0 && c==EOF)
        i = EOF;
    return i;
}
```

① 함수의 리턴 타입은 int
- 입력된 문자의 개수 정보를 넘겨줌

② 매개변수로 전달된 벡터에 읽어들인 문자열을 저장

③ 키보드 버퍼(buffer)를 비움
- 사용자가 벡터의 크기보다 더 많은 문자를 입력하면 넘치는 문자는 저장하지 않고 버림

※ 고려사항

- 표준 함수와 달리 getline()의 프로그램 코드는 getline()을 호출하는 모든 프로그램에 링크 되어야 함
- 링크 에디터가 표준 라이브러리뿐만 아니라 개인 라이브러리도 검사하도록 옵션을 수정해야 함

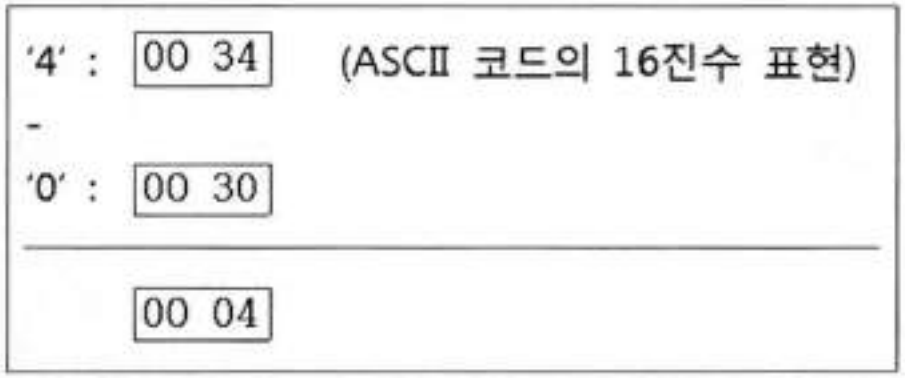
벡터와 배열

>> 벡터 인수 전달

- 숫자로 된 문자열을 int 수로 변환하는 함수
  - int asctoint(char s[])
  - 표준 함수 atoi()
    → stdlib.h에 프로토타입(prototype)이 있으므로 이 헤더 파일을 포함하여 사용

※ 표준 함수 atof()
  - 연속된 숫자 문자를 double형 숫자로 변환

- 숫자 문자를 값으로 변환
  - ACSII 코드 표 참조



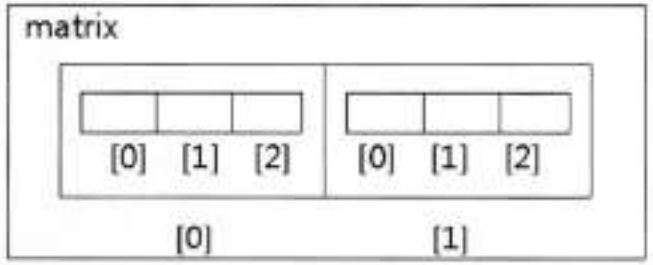
- 12장의 표준 함수를 참조

벡터와 배열

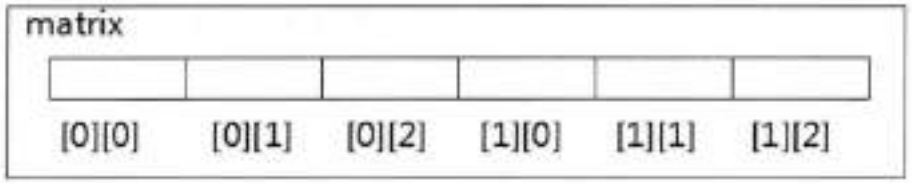
>> 다차원 배열

- 다차원 배열
  - 두 개, 세 개, 네 개 등 여러 개의 인덱스로 접근할 수 있는 동일한 데이터 타입의 요소를 모아놓은 데이터 집합체
  - 벡터 : 1차원 배열
  - 6개의 int요소를 가진 행렬
    → int matrix[2][3]
  - matrix의 메모리 맵

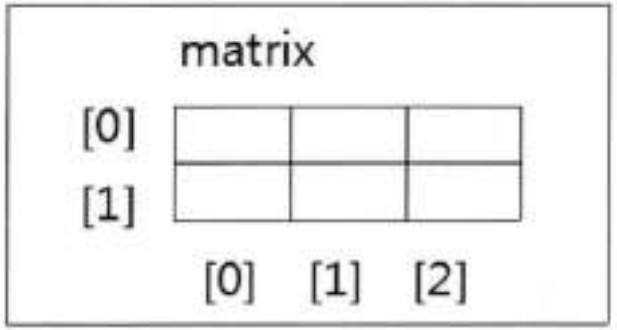
①



②



③



- 행렬의 각 요소는 두개의 인덱스를 가지고 지정되어야 함

벡터와 배열

>> 다차원 배열

- **프로그램 예**

- matrix의 메모리 맵을 초기화

```
int matrix[2][3];
int i, k;
int z = 0;
for(i=0; i<2; ++i)
        for(k=0; k<3; ++k)
                matrix[i][k] = ++z;
```

출력 예

```
for(k=0; k<3; ++k)
        for(i=0; i<2; ++i)
                printf("%d:", matrix[i][k]);

출력 ⇒ 1:4:2:5:3:6:
```

벡터와 배열

>> 다차원 배열

- N - 차원 배열에 대한 일반적인 규칙
  - 배열이 함수로 전달된다면 함수 헤더에 정의되는 해당 배열의 처음 대괄호 []는 항상 비어있어야 함
  - 두 번째 []부터는 전송되는 배열과 같은 수를 가져야 함
  - 첫 번째 []에 채워질 차수는 추가 인수로서 함수에게 전달 되어야 함

```
void set(int m[][3], int n) {
        int i, k, z = 0;
        for(i=0; i<n; ++i)
                for(k=0; k<3; ++k)
                        m[i][k] = ++z;
}
```

- 행렬의 초기화
  - int matrix[2][3] = {{13, -4, 195}, {25, 37, 2}};
    → 행렬이 바로 초기화 된다면 첫 번째 대괄호[]는 벡터의 경우와 같이 정의할 때는 빈 경우로 둘 수 있다

함수

>> 정의, 선언, 호출

- **함수 정의**

```
[static] type fname (plist)
{
    . . .
    . . . /* Instructions from the functions structure */
    . . .
}
```

1. type : 리턴이 없는 경우 void
   리턴이 있는 경우 리턴값의 데이터 타입
2. fname : 함수이름
3. plist : 매개변수가 없는 경우 void 또는 빈 괄호 ( )
   매개변수가 있는 경우 [register] type variable 형식으로 정의하며 매개변수들은 ','로 분리
   * static으로 정의되지 않는 한 함수는 아무 모듈에서나 extern으로 선언하고 호출할 수 있다. static으로 정의된 함수는 모듈 내의 함수에서만 호출이 가능하다.

함수

>> 정의, 선언, 호출

- **함수 또는 프로토 타입의 선언**

[extern | static] type fname (plist);

* 키워드 extern은 생략하는 것이 보통으로 다음 두 함수의 프로토타입은 같다.
  예1: extern int getline (char * s, int lim);
  예2: int getline (char * s, int lim);
* 변수 이름은 생략이 가능하다. 다음 두 예는 같다.
  예1: int getline (char * s, int lim);
  예2: int getline (char *, int);

- **함수의 호출**

fname (arglist)

1. arglist : 함수의 정의에 명시된 형식 매개변수들에 전달될 실제 인수. 콤마로 분리하여 그 수와 타입이 형식 매개변수와 일치해야 한다.
2. 리턴값이 없는 void함수일 경우 호출 표현식은 아무값도 가지지 않는다.

- 인수의 자동 타입변환
  인수와 매개변수가 반드시 동일한 타입일 필요는 없음.
  * 함수의 프로토타입이 없는 경우 자동 타입변환 원칙
    1. char, unsigned char, short, unsigned short 타입의 인수는 int 타입으로 변환됨.
    2. float 타입은 double타입으로 타입변환

함수

>> 정의, 선언, 호출

- 예시

```
# include <stdio.h>
void func1 (double z)
{
    printf ("%lf\n", z);
}
void func2 (char * p)
{
    printf ("%c\n", *p);
}
int main (void)
{
    long z = 0x41424344L;
    func1 (3.5);
    func1 (3.5f);
    func1 (12);
    func1 (-18L);
    func1 ((short) 7);
    func1 ('A');
/*  func1 ("A");  Error!
                  char pointer: no conversion into double! */
    func2 ("ABC");
/*  func2 (1);  Error!
                int: no conversion into char *! */
    func2 (&z);   /* Warning: "Suspicious pointer
                  conversion"! */
    return 0;
}
/* ------ program run: ---------- *
3.500000
3.500000
12.000000
-18.000000
7.000000
65.000000
A
D
* ------------------------------- */
```

함수

>> 정의, 선언, 호출

- 함수의 중첩

```
예: int incr (int x)
    {
        return ++x;
    }
    int quad (int z)
    {
        return z * z;
    }
```

  - 여기서 표현식 quad(incr(3))의 값은 16이다. 먼저 incr함수가 호출되고 그 다음 그 결과(리턴값)가 quad함수의 인수가 되어 quad함수가 호출된다.