

# 컴퓨터가 표현하는 데이터 1

## 숫자

- 소수점 이하 자릿수를 다루지 않는 숫자 → 정수형 (int)
- 소수점 이하 자릿수를 다루는 숫자 → 실수형 (단정도 실수형(float), 배정도 실수형(double))

## 문자

- 하나의 문자
- 문자가 여러 개 모여있는 형태 → 문자열

# 기본 자료형

## 자료형 종류와 표현 범위

자료형	자료형 이름	표현범위	예
정수형	int	4바이트(-2147483648~2147483647)	-984, 56
	short int	2바이트(-32768~32767)	
	long int	4바이트	
	unsigned int	4바이트	34, 987
	unsigned short int	2바이트	
실수형	float	4바이트	-76.345, 674.3 소수점이 있는 상수는 배정도형 실수로 취급
배정도형 실수	double	8바이트	
	long double	8바이트	
문자형	char	1바이트(-128~127)	'A', '4', '*'
	unsigned char	1바이트(0~255)	

# 단축 연산자

## 단축 연산자

연산	연산자	설명
덧셈 누적	$+=$	$a=10; a+=3;$ 변수 a는 13이 저장됨
뺄셈 누적	$-=$	$a=10; a-=3;$ 변수 a는 7이 저장됨
곱셈 누적	$*=$	$a=10; a*=3;$ 변수 a는 30이 저장됨
나누기 누적	$/=$	$a=10; a/=3;$ 변수 a는 3이 저장됨
나머지 누적	$\%=$	$a=10; a\%=3;$ 변수 a는 1이 저장됨
1누적	$++$	$a=10; a++;$ 변수 a는 11이 저장됨
1감소	$--$	$a=10; a--;$ 변수 a는 9이 저장됨

# 연산자 우선순위 1

- + 연산 방향 : 왼쪽 → 오른쪽
- + 우선순위가 높은 연산자가 먼저 실행

우선순위	연산자
높음	후치 연산자(변수++, 변수--), static_cast<type>() +, -, ~ (단항), 전치 연산자(++변수, --변수) ! *, /, % +, - <, <=, >, >=, ==, != &, ^,   (비트 연산자) && (AND)    (OR)
낮음	=, +=, -=, *=, /=, %= (대입 연산자)

# 연산자 우선순위 2

연산 결과는?

$$3 * 4 - 78 < 12 - 8 \% 5$$

연산 순서

- ①  $3 * 4 \rightarrow 12$
- ②  $12 - 78 \rightarrow -66$
- ③  $8 \% 5 \rightarrow 3$
- ④  $12 - 3 \rightarrow 9$
- ⑤  $-66 < 9 \rightarrow$  참 (1)

# 조건문 if

## 조건이 참일 때 주어진 내용 실행

```
if (조건)
{
    조건이 참일 때 수행할 문장;
}
```

## 문제 : 정수형 변수의 값이 100 보다 크면 "100 보다 크다."라고 화면에 출력한다.

- ① 정수형 변수 a를 선언하고 123으로 초기화한다.
- ② "변수 a의 값이 100보다 크면 "100보다 크다."라고 출력한다.

```
int a;  
  
cout << "정수 입력 : " ;  
cin >> a;  
  
if (a>100)  
{  
    cout << "a의 값 : " << a << endl;  
    cout << a << "는 100보다 크다" << endl;  
}  
  
cout << "즐거운 C++ 프로그램을 종료합니다.!!!" << endl;
```

# if~else

- 조건이 참일 때, 거짓일 때 각 내용 실행

```
if (조건)
{
    조건이 참일 때 수행할 문장;
}
else
{
    조건이 거짓일 때 수행할 문장;
}
```

문제) 정수를 입력해서 입력한 정수가 100보다 크면 "100 보다 크다"를 출력하고 그렇지 않으면 "100보다 작다"를 출력하자.

조건이 참일 때 실행할 내용

조건이 거짓일 때 실행할 내용

# if~else 실습

## 소스 3-4

```
int a;  
  
cout << "정수 입력 :" ;  
cin >> a;  
  
if (a>100)  
    cout <<"100보다 크다" << endl;  
else  
    cout << "100보다 작다" << endl;
```

정수 입력 : 200  
100보다 크다

정수 입력 : 90  
100보다 작다

정수 입력 : 100  
100보다 작다

# if ~else if ~ ... else

주어진 조건이 여러 단계로 이어질 때

if (조건1)

조건1이 참일 때 실행;

else if (조건2)

조건2가 참일 때 실행;

else if (조건3)

조건3이 참일 때 실행;

.....

else

위의 조건이 모두 거짓일 때 실행;

# break

## 주어진 범위를 벗어나게 함

```
switch(num)
{
    case 1:
        cout << "1의 값이 입력되었습니다." << endl;
        cout << "*****" << endl;
break;
    case 2:
        cout << "2의 값이 입력되었습니다." << endl;
        cout << "*****" << endl;
break;
    default:
        cout << "1 또는 2가 아닌 값이 입력되었습니다." << endl;
        cout << "*****" << endl;
}
```

숫자 입력 : 1  
1의 값이 입력되었습니다.  
\*\*\*\*\*

숫자 입력 : 2  
2의 값이 입력되었습니다.  
\*\*\*\*\*

# 반복문 - for

## for 문

for (초기화 ; 조건문 ; 증감문 )  
반복할 내용;

### - 반복문 for 실행 순서

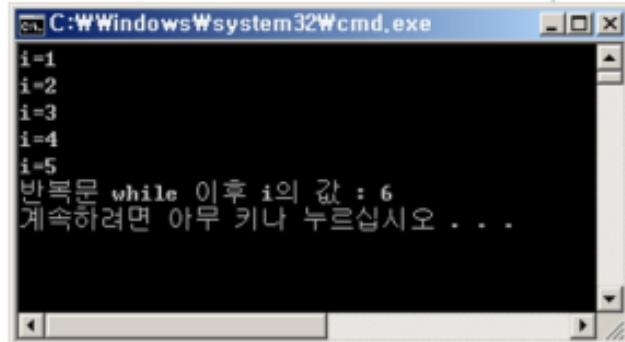
- ① 초기화를 실행한다.
- ② 조건문을 실행한다. 조건이 참이면 ③을 실행하고, 거짓이면 반복문을 종료한다. (반복문 밖으로 제어가 이동됨)
- ③ 반복할 내용을 실행한다.
- ④ 증감문을 실행한 후 ②를 실행한다.

# 반복문 - while

## while 문

while (조건문)  
반복할 내용;

```
int i;  
  
i=1;  
while (i<=5)  
{  
    cout << "i=" << i << endl;  
    i++;  
}  
  
cout << "반복문while 이후 i의 값: " << i << endl;
```



# 반복문 - do~while

- + do~while : 적어도 한번은 실행하고 주어진 조건이 참이면 다시 반복함

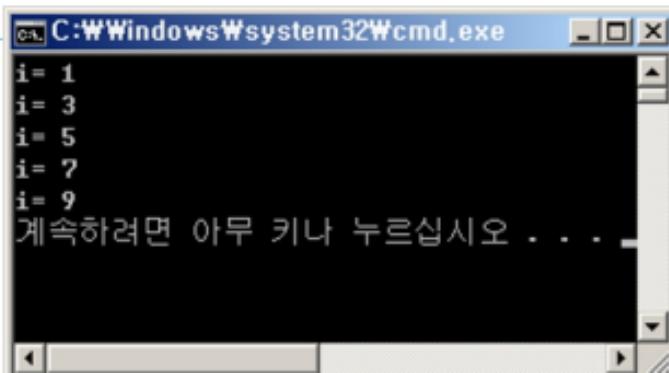
```
do {  
    반복할 내용;  
} while (조건문);
```

```
int score;  
  
do {  
    cout << "점수입력: " ;  
    cin >> score;  
} while (score>100 || score<0);  
  
cout << "당신이 입력한 점수는" << score << "입니다. " << endl;
```

# continue

- 반복 범위 처음으로 실행 제어 이동

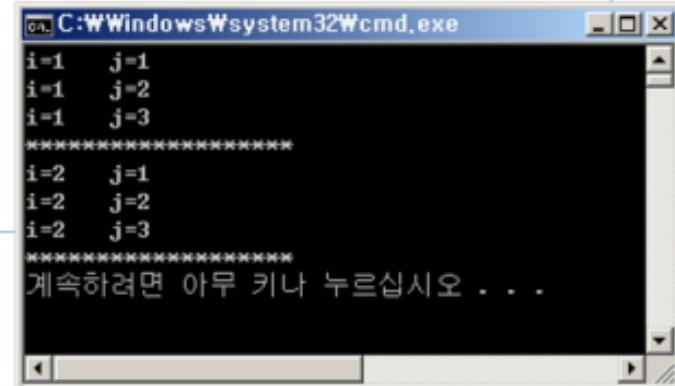
```
int i;  
  
for (i=1; i<10; i++)  
{  
    if (i%2 == 0)  
        continue;  
    cout << "i= " << i << endl;  
}
```



# 반복문의 중복

- 반복문 내에 또 다른 반복문을 포함

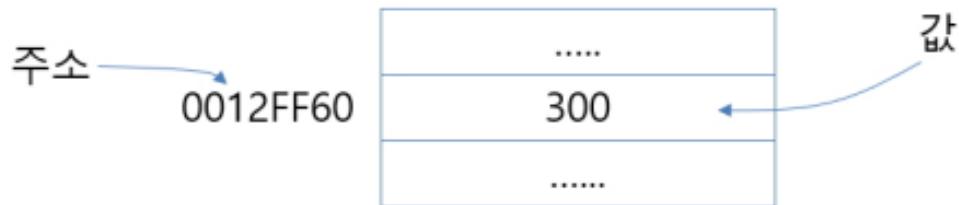
```
int i, j;  
  
for (i=1; i<=2; i++)  
{  
    for (j=1; j<=3; j++)  
    {  
        cout << "i=" << i << " " ;  
        cout << "j=" << j << endl;  
    }  
  
    cout << "*****" << endl;  
}
```



```
i=1    j=1  
i=1    j=2  
i=1    j=3  
*****  
i=2    j=1  
i=2    j=2  
i=2    j=3  
*****  
계속하려면 아무 키나 누르십시오 . . .
```

# 포인터

- 컴퓨터는 처리하는 모든 데이터를 주기억장치에 저장한다.
- 포인터 : 주기억장치의 주소



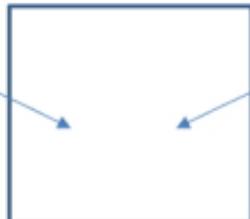
- 포인터 변수 : 주소를 저장
- 일반 변수 : 값을 저장

# 포인터 변수 1

## 변수

- 저장장소
- 변수는 사용하기 전에 미리 선언해야 한다.
- 변수는 사용하기 전에 초기화 되어야 한다.
  - 초기화하지 않은 경우 쓰레기 값이 저장되어 있다.

값을 저장하려면  
일반 변수로 선언  
되어야 함!!



주소를 저장하려  
면 포인터 변수로  
선언되어야 함!!

```
#include <iostream>
using namespace std;

int main()
{
    int sum=0, i=0;

    for (i=1; i<=10; i++)
        sum=sum+i;

    cout << "1~10까지의 합 :" << sum << endl;
    return 0;
}
```

# 포인터 변수 2

- 변수 값 참조 – 일반변수 이름 사용
- 변수의 주소 확인(주소지정 연산자 & 사용)

소스 4-2 (ch04\_01\_1.cpp)

```
int a=100;  
  
cout << "a에 저장된 값 :" << a << endl;  
cout << "a의 주소 :" << &a << endl;  
  
return 0;
```

실행 결과

a에 저장된 값 : 100  
a의 주소 : 0012FF60

# 포인터 변수 3

## 포인터 변수

- 변수의 주소를 저장
- 포인터 변수에 저장할 주소에 저장될 자료형과  
포인터 변수의 자료형은 일치해야 함  
(정수형 포인터 변수는 정수형 변수의 주소, 문자  
형 포인터 변수는 문자형 변수의 주소를 저장함)

## 일반변수와 마찬가지로 사용 전 선언해야 함

자료형      일반변수이름;

자료형      \*포인터변수이름;

# 일반변수와 포인터 변수

- + 일반변수 : 값을 저장해야 할 경우 필요
- + 포인터변수 : 변수의 주소를 저장해야 할 경우

	일반변수	포인터 변수	비고
선언	<u>자료형</u> 변수이름;	<u>자료형</u> * <u>포변수이름</u> ;	
값 할당	변수이름=값;	<u>포변수이름</u> =주소; * <u>포변수이름</u> =값;	& : 주소지정 연산자 * : 간접 연산자
선언과 동시에 초기화	<u>자료형</u> 변수이름=값;	<u>자료형</u> * <u>포변수이름</u> =주소;	

'포인터변수이름'을 '포변수이름'으로 표기함

# 배열 1

## 변수

- 하나의 기억 공간
- 예) 나이, 점수, 성별 등의 데이터를 저장

## 배열

### **연속적인 기억 공간**

- 예) 30명의 총점을 저장, 세 과목의 성적을 저장
- 선언할 때 연속적으로 필요한 기억 공간의 개수를 표시

# 배열 2

- 배열 선언 : 자료형, 배열이름, 크기

자료형 배열이름[크기];

- 배열 첨자는 0부터 시작

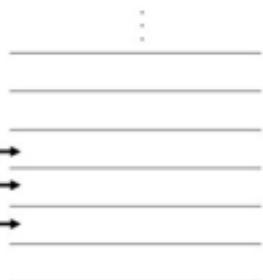
- int score[3]; 의 경우  
 score[**0**], score[**1**], score[**2**]

- char name[30];의 경우  
 name[**0**], name[**1**], ..., name[**28**], name[**29**]

int score[3];



컴퓨터내의 메모리



# 배열 3

## 배열이름 = 주소

```
int score[3];
```

```
&score[0] == score  
&score[1] == score+1  
&score[2] == score+2
```

```
char name[5];
```

```
&name[0] == name  
&name[1] == name+1  
&name[2] == name+2  
&name[3] == name+3  
&name[4] == name+4
```

정수형 배열의 경우 각 기억장소가  
4바이트 단위로 증가함

문자형 배열의 경우 각 기억장소가  
1바이트 단위로 증가함

# 배열 초기화

## 선언과 동시에 초기화

자료형 배열이름[크기]={초기값1, 초기값2, ....};

자료형 배열이름[크기]={초기값, };

```
int s[3]={10,20, 30};
```

```
int a[5]={0,};
```

# 문자열

+ 문자열 : 문자의 모음

+ 문자 배열

```
char string[30] = "computer";
```

```
char string[30] = { 'c', 'o', 'm', 'p', 'u', 't', 'e', 'r', 'W', 'O' };
```

- 문자열 상수는 쌍따옴표 ""로 표기

- 문자열 마지막을 알리는 널(NULL)문자가 자동으로 입력됨

# 문자열 함수 1

- + 자주 사용하는 문자열 처리 함수를 라이브러리에서 제공함
- + 문자열 길이 구하기
  - strlen(const char \*\_Str)
- + 문자열 복사하기
  - strcpy(char \*\_Dest, const char \*\_Str)  
strcpy\_s(char \*\_Dest, rsize\_t SizelnBytes, const char \*\_Str)

# 문자열 함수 2

## ▶ 문자열 결합하기

- strcat(char \*\_Dest, const char \*\_Str)
- strcat\_s(char \*\_Dest, rsize\_t Sizeln Bytes, const char \*\_Str)

## ▶ 문자열 비교하기

- strcmp(const char \*str1, const char \*str2)

# 포인터 : 메모리 주소

## 포인터의 크기

- 주기억장치에서의 자료 처리 기본 단위
- 운영체제에 의해 결정, 윈도우 XP(32비트 운영체제)

```
int *pi;  
char *pc;  
float *pf;  
double *pd;  
  
cout << "정수형 포인터 크기 :" << sizeof(pi) << endl;  
cout << "문자형 포인터 크기 :" << sizeof(pc) << endl;  
cout << "실수형 포인터 크기 :" << sizeof(pf) << endl;  
cout << "배정도형 포인터 크기 :" << sizeof(pd) << endl;
```

# 레퍼런스 변수

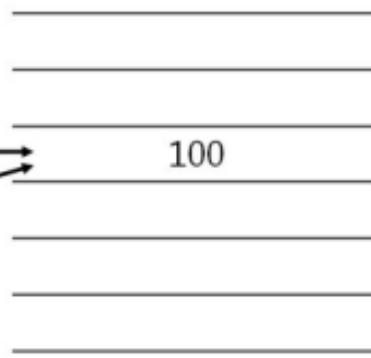
- + 이미 선언한 변수를 다른 이름을 부르는 것
- + 레퍼런스 변수 선언시 반드시 초기화 해야 함!

자료형 & 변수이름=변수;

```
int a=100;  
int &ra=a;
```



a  
ra



# 동적 할당 1

## 자료 저장을 위한 기억장소 할당 방법

### - 정적 할당

- 프로그램에서 필요한 변수를 선언
- 프로그램 실행 시작에서 필요한 변수에 대한 기억공간이 할당됨

### - 동적 할당

- 프로그램에서 필요한 기억공간의 크기를 할당하여 그 시작 주소를 기억 (포인터변수 사용!!)
- 프로그램 실행 중 기억공간이 할당되고, 사용을 마친 후 할당한 기억공간을 해제

# 동적 할당 2

- + 기억장소 할당 : new
- + 기억장소 해제 : delete

```
자료형 *포인터변수 = new 자료형[개수];
```

```
delete 포인터변수; //하나의 기억장소 해제  
delete [] 포인터변수; //여러 개의 기억장소 해제
```

```
double *dp = new double; //한 개의 배정도형 기억공간 할당  
delete dp;
```

```
int *ip=new int[20];  
delete [] ip;
```

# 구조체

## 구조체

- 자료의 논리적 표현 단위
- 사용자가 필요에 의해 여러 자료를 하나의 자료형으로 정의
- 데이터 베이스의 레코드 형식을 하나의 자료형으로 정의

필드

필드

이 름	전화번호
박돌쇠	010-3698-0000
황아름	010-1478-0000
정철수	010-2313-0000
최보람	010-2589-0000
김갑돌	010-8741-0000

레코드

레코드

레코드

레코드

레코드

# 구조체 정의

- 구조체 태그 : 구조체 식별자, 생략 가능함
- 구조체 멤버 : 구조체를 구성하는 항목

//구조체 선언하여 변수 선언하기1

```
struct [태그이름]
{
    구조체 멤버 선언;
};
```

struct [태그 이름] 변수이름1, 변수이름2;

//구조체 선언하여 변수 선언하기2

```
struct [태그이름]
{
    구조체 멤버 선언;
} 변수이름1, 변수이름2;
```

- 구조체 → 하나의 자료형!!

`int SampleVar;`

자료형      변수이름

`struct  
{  
 char Name[30];  
 char MPhoneNum[20];  
} friend;`

변수이름      자료형

`int/float/double/char`

동급  
=

`struct  
{  
 char Name[30];  
 char MPhoneNum[20];  
};`

선언한 변수 :  
**Name, MPhoneNum**

선언한 변수 : **friend**

일반 변수는 변수이름 그대로 사용 :

strcpy\_s(Name, 30, "김갑돌");

구조체형 변수의 멤버 사용 – 구조체 멤버 참조 연산자(.)를 사용함

strcpy\_s(friend.Name, 30, "김갑순");

# 구조체를 구조체 멤버로

- 구조체 멤버에 또 다른 구조체형 변수가 멤버로 사용이 가능함

```
struct POINT
{
    int x;
    int y;
};

struct RECT
{
    struct POINT LeftTop; //struct POINT 구조체형
    struct POINT RightBottom; //struct POINT 구조체형
    int area;
};
```