민 선 생 코 딩 학 원 시 작 반

수업노트 LV-11

배우는 내용

1. 함수값 주고 받기
2. 포인터의 이해
3. Flag방식
4. MAX
5. 다른 배열에 값 옮기기

복습 : Call by value

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

void ryan(int a, int b, char ch)
{
    cout << a << b << ch;
}

int main()
{
    ryan(1, 2, 'E');

    return 0;
}
```

- Call by value : 함수 호출할 때 값도 같이 보내기

- **용어 암기**

  보내는 값을 Argument (아규먼트, 인자값)

  받는 값을 Parameter (파라미터, 매개변수)

복습 : 함수에서 값 return하기

- tomtom 함수는
  숫자 2개를 입력받고, 합을 return 해주는 함수
- main함수에서는
  변수 a에다가 리턴받은 수를 넣고, a를 출력한다.
- return 받는 값은 단 한 개의 값만 리턴받을 수 있음

[주의]
Main함수에 있는 변수 a와
Tomtom함수에 있는 변수 a는
서로 다른 변수이다.

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

int tomtom( )
{
    int a, b;
    cin >> a >> b;

    return a + b;
}

int main( )
{
    int a;

    a = tomtom( );

    cout << a;

    return 0;
}
```

값을 주고 받는 함수

- 값을 보내는 것과 받는 것을 동시에 할 수 있음
- 보내는 값 (아규먼트)는 한번에 여러 값 들을 보낼 수 있지만 return 받는 값은 단 한 개의 값만 리턴받을 수 있음

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

int SUM(int a, int b)
{
    return a + b;
}

int main()
{
    int a, b;
    cin >> a >> b;

    int ret = SUM(a, b);

    cout << ret;

    return 0;
}
```

값을 주고 받는 함수- 예제2

- 다음 소스코드를 이해 해 보세요

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

int passOrNo(int num)
{
    if (num >= 90)
    {
        return 'A';
    }
    else if (num >= 80)
    {
        return 'B';
    }

    return 'C';
}
```

passOrNo 함수는

숫자를 넣으면

A / B / C 를 리턴해주는 함수

```cpp
int main()
{
    int num;

    cin >> num;

    char ret = passOrNo(num);

    if (ret == 'A' || ret == 'B')
    {
        cout << "PASS";
    }
    else
    {
        cout << "FAIL";
    }

    return 0;
}
```

함수 선언하기

- main 아래에다가 함수를 쓰고 싶으면 함수 선언을 해 주어야 한다.

함수선언

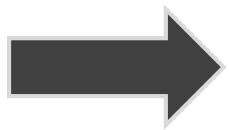
```
#include <iostream>
using namespace std;

void bbq();

int main()
{
    bbq();

    return 0;
}

void bbq()
{
    cout << "BBQ";
}
```

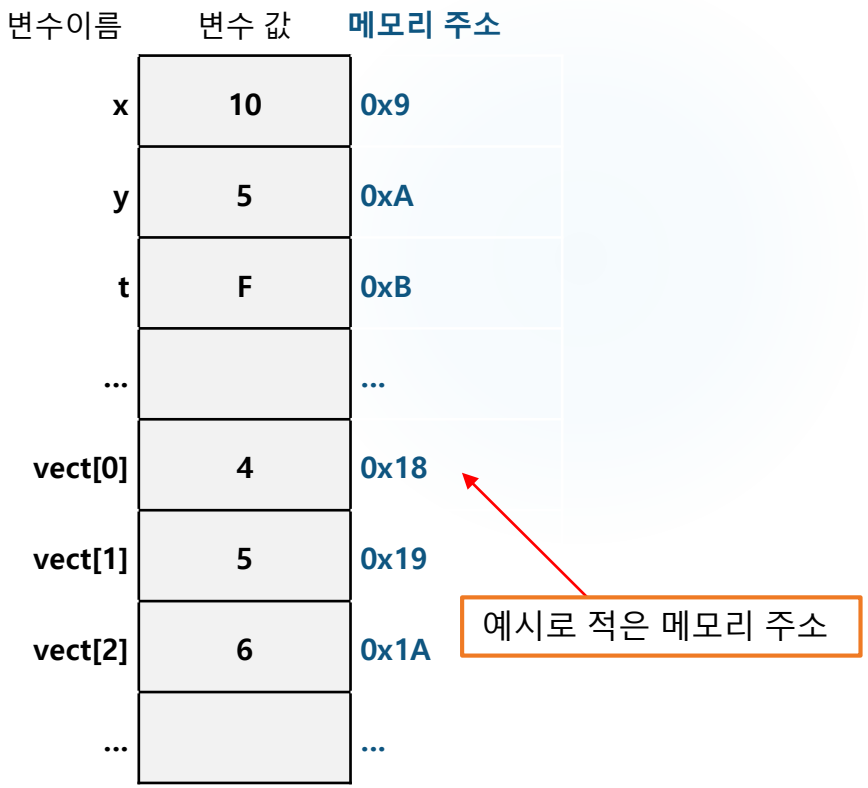
함수선언

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

int kfc(int a, int b, int c);

int main()
{
    int ret = kfc(1, 2, 3);

    cout << ret;

    return 0;
}

int kfc(int a, int b, int c)
{
    return a + b + c;
}
```

변수 선언에 따른 메모리 구조

```
int x = 10;
int y = 5;
char t = 'F';

int vect[3] = {4, 5, 6};
```





- 변수와 배열을 만들면 오른쪽과 같이 메모리에 값이 채워 짐
- 값이 들어가는 각 칸마다 "**메모리 주소**"가 있다
- 메모리 주소는 16진수로 표기 함

Visual Studio에서 메모리 주소 확인방법

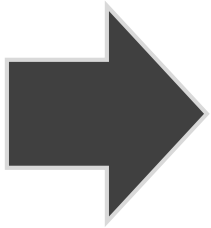
- 변수 앞에 **&**를 붙이면 그 변수의 메모리 주소 확인 가능
- **vect[0]** ~ **vect[2]**라는 공간은 존재 하지만
  배열 이름인 vect라는 **메모리 공간은 존재하지 않음을 알아 둘 것**

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

int main()
{
    int x = 10;
    int y = 5;
    char t = 'F';
    int vect[3] = { 4, 5, 6 };

    return 0;
}
```

조사식 1

검색(Ctrl+E) 검색 심도: 3

이름	값	형식
x	10	int
&x	0x00fbfc9c {10}	int *
y	5	int
&y	0x00fbfc90 {5}	int *
vect[0]	4	int
vect[1]	5	int
vect[2]	6	int
&vect[0]	0x00fbfc7c {4}	int *
&vect[1]	0x00fbfc80 {5}	int *
&vect[2]	0x00fbfc84 {6}	int *

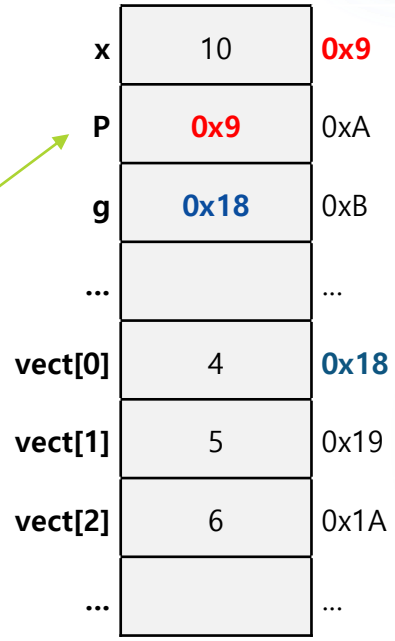


x	10	0xFBFC9C
y	5	0xFBFC90
t	F	0xFBFC73
…		…
vect[0]	4	0xFBFC7C
vect[1]	5	0xFBFC80
vect[2]	6	0xFBFC84
…		…

포인터란?

- 포인터 변수는 **메모리 주소를 저장하는 변수**
- 포인터 변수 or 포인터라고 부름
- 포인터 선언하는 방법

ex) **int *p; // p라는 이름의 포인터를 만듬**



- **포인터 사용 예제**

```
int x = 10;
int *p; //포인터 p를 선언함

p = &x; //p에다가 변수 x의 주소를 넣음
```

```
int vect[3] = {4, 5, 6};
int *g; //포인터 G를 선언함

g = &vect[0]; //G에다가 vect[0]의 주소를 넣음
```

int형 포인터 / char형 포인터

- int형 포인터는 **int 변수의 주소**를 저장하는 변수를 뜻함

```
int *p;
int x = 5;

p = &x;
```

p에는 x의 값인 5가 들어있는 것이 아니라
x의 주소값이 들어 있음

- char형 포인터는 **char 변수의 주소**를 저장하는 변수를 뜻함

```
char *p;
char b = 'A';

p = &b;
```

p에는 b의 값인 'A'가 들어있는 것이 아니라
b의 주소값이 들어 있음

int형 포인터는 int의 주소를 저장해야 함

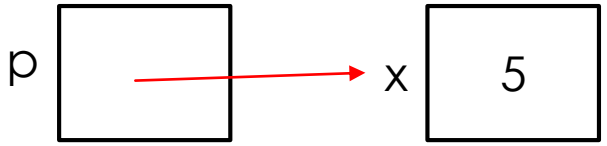
만약 int형 포인터에다가 char형 변수의 주소를 저장하면 **Compile Error가 발생**

“가리킨다” 라는 말 뜻

```
int *p;
int x = 5;

p = &x;
```

- 왼쪽의 소스코드는
  포인터 p에다가 x의 주소를 저장하는 코드

- 포인터 p가 x의 주소를 저장하고 있을 때
  “**p는 x를 가리킨다**” 라고 표현한다 ➔ **p = &x;**



가리킨다 라는 말 뜻을 반드시 기억 해 주세요

포인터변수의 원격 접근 기능

- 포인터는 어떤 변수를 가리키고 있을 때, 포인터 변수 앞에 *을 붙이면 특별한 능력을 쓸 수 있다

  → **포인터로 가리키고 있는 변수를 원격조정 가능!**

```
int x = 5;
int *p;
p = &x;

*p = 100;
```

변수 x의 값이 100으로 바뀜

```
int x = 5;
int *p = &x;

cout << *p;
```

변수 x의 값인 5가 출력 됨

변수 x를 사용하지 않고,
***p**로 변수 x값을 마음대로 읽고 쓰기가 가능!

포인터를 언제 쓰나요?

- 함수는 한 개의 값만 return 할 수 있음

  포인터를 활용하면 **여러 값들을 return할 수 있음**

  (Level 13에서 배울 예정)

- Linked List라는 자료구조에 활용 (훈련반2. Level 24에서 배울 예정)

버그 찾아보기

- 이 프로그램에서 나올 수 있는 버그는 무엇일까?

숫자 하나를 입력받고,
입력 받은 숫자가 아래 배열에 존재한다면 “발견” 출력
존재하지 않다면 “미발견” 출력


```
int vect[7] = {5, 6, 3, 2, 6, 6, 1};
int input;
cin >> input;

for (x=0; x<7; x++)
{
   if (vect[x] == input)
   {
      cout << “발견”;
      break;
   }
   else
   {
      cout << “미발견”;
   }
}
```

만약 3을 입력하면, “**미발견 미발견 발견**” 이렇게 출력되는 버그가 발생 함

Flag 코딩 기법

- Flag 변수를 하나 만들고 활용하는 방식
- 배열에 어떤 값이 존재하는지 판단할 때 사용하는 코딩 기법

- Counting VS flag 기법

  Counting 기법으로 코딩할 수 있지만,
  Counting은 몇 개인지 셀 때 쓰이고
  flag는 존재여부를 판단할 때 쓰인다.

Flag 소스코드 분석하기

- 어떤 값이 존재하는지 여부를 판단할 때 사용하는 코딩 기법

  입력 받은 숫자 존재여부 확인 문제

5	6	3	2	6	6	1

- 입력받은 숫자를 찾았으면 for문을 더 이상 진행 할 필요가 없기 때문에 break;를 걸어 줌

- Counting 방식으로도 이 문제를 풀 수 있지만
  Counting은 몇 개인지 셀 때,
  Flag는 존재여부 판단할 때 사용할 것

```
int vect[7] = {5, 6, 3, 2, 6, 6, 1};
int flag;
int input;
cin >> input;

flag = 0;

for (x=0; x<7; x++)
{
   if (vect[x] == input)
   {
      flag = 1;
      break;
  }
}

if (flag == 1)
{
   cout << "발견";
}
else
{
   cout << "미발견"
}
```

Flag 방식을 쓰기 위한 3단계를 기억하세요

1. **Flag 초기화**
   - 검사를 시작하기 전 초기값을 Flag 변수에 저장
   - 초기화를 안하는 실수가 많이 발생

2. **조건 검사**
   - 특정 조건이 발견될 때 까지 검사를 수행

3. **Flag 확인**
   - 검사가 끝나고 결과를 확인

훈련반 / 실무에서 굉장히 많이 쓰이는 방식입니다

```
int vect[7] = {5, 6, 3, 2, 6, 6, 1};
int flag;
int input;
cin >> input;
//단계1. flag변수 초기화
flag = 0;

//단계2. 검사하기
for (x=0; x<7; x++)
{
   if (vect[x] == input)
   {
      flag = 1;
      break; //더 이상 진행 할 필요없음
   }
}

//단계3. 결과 확인하기
if (flag == 1)
{
   cout << "발견";
}
else
{
   cout << "미발견"
}
```

가장 큰 값과 가장 작은 값 찾기 (MAX / MIN)

5	6	3	12	6	6	1

- 배열에서 가장 큰 값 찾는 방법

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

int main()
{
    int vect[7] = { 5, 6, 3, 12, 6, 6, 1 };
    int max;
    int x;

    //max는 가장 작은 수 부터 시작
    max = -999;

    //숫자 하나씩 탐색
    for (x = 0; x < 7; x++)
    {
        //만약 max보다 더 큰 원소가 나타나면?
        if (vect[x] > max)
        {
            max = vect[x]; //max값 갱신
        }
    }

    cout << max;

    return 0;
}
```

출력결과 : 12

- 배열에서 가장 작은 값 찾기

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

int main()
{
    int vect[7] = { 5, 6, 3, 12, 6, 6, 1 };
    int min;
    int x;

    min = 999;

    //숫자 하나씩 탐색
    for (x = 0; x < 7; x++)
    {
        //만약 min보다 작은 요소를 찾았다면
        if (vect[x] < min)
        {
            min = vect[x]; //갱신
        }
    }

    cout << min;

    return 0;
}
```

출력결과 : 1

MAX 예제

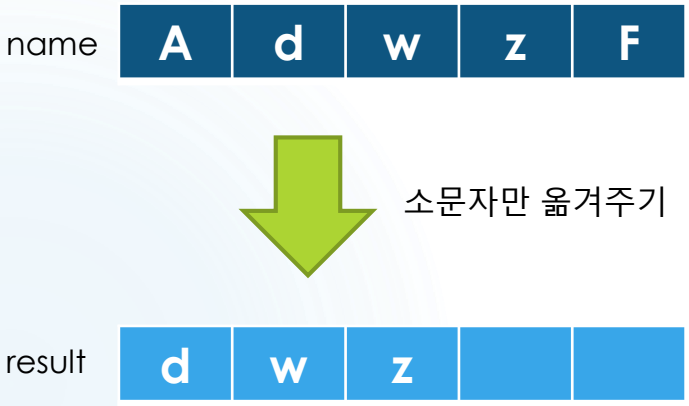
A	E	H	W	A

- 사전순으로 가장 큰 문자 찾아내는 예제

- 문자도 아스키코드 번호로 되어있기 때문에 숫자처럼 MAX, MIN 검색이 가능하다.

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

int main()
{
    char name[5] = { 'A', 'E', 'H', 'W', 'A' };
    char max;
    int x;

    max = 0;
    for (x = 0; x < 5; x++)
    {
        if (name[x] > max)
        {
            max = name[x];
        }
    }

    cout << max;

    return 0;
}
```

다른 배열에 값 옮기기




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

int main()
{
    char name[5] = { 'A', 'd', 'w', 'z', 'F' };
    char result[5];
    int t, x;

    t = 0;
    for (x = 0; x < 5; x++)
    {
        if (name[x] >= 'a' && name[x] <= 'z')
        {
            result[t] = name[x];
            t++;
        }
    }

    for (x = 0; x < t; x++)
    {
        cout << result[x];
    }

    return 0;
}
```

변수 t 활용

출력결과 : dwz