C++ Espresso

제2장 제어문과 함수

이번 장에서 학습할 내용

- 조건문
- 반복문
- break, continue
- 함수 개념
- 함수 정의
- 함수 원형
- 매개 변수
- 순환 호출

switch 문

```
int main()
{
      int number;

      cout << "정수를 입력하시오:";
      cin >> number;
      switch(number)
      {
            case 0:
                  cout << "없음\n";
                  break ;
            case 1:
                  cout << "하나\n";
                  break ;
            case 2:
                  cout << "둘\n";
                  break ;
            default:
                  cout << "많음\n";
                  break;
      }
}
```

정수를 입력하시오: 1
하나

중간 점검 문제

1. case 절에서 break 문을 생략하면 어떻게 되는가? 변수 fruit의 값이 각각 1, 2, 5일 때, 다음의 코드의 출력을 쓰시오.

```
switch(fruit) {
            case 1: cout <<"사과";
                    break;
            case 2: cout <<"배";
            case 3: cout <<"바나나";
                    break;
            default:cout <<"과일";
}
```

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

int main()
{
        int vowel=0, consonant=0;
        char ch;

        cout << "영문자를 입력하고 콘트롤-Z를 치세요" << endl;
        while(cin >> ch)
        {
                switch (ch) {
                        case 'a':
                        case 'i':
                        case 'e':
                        case 'o':
                        case 'u':
                                vowel++;
                                break;
                        default:
                                consonant++;
                                break;
                }
        }
        cout << "모음: " << vowel << endl;
        cout << "자음: " << consonant << endl;

        return 0;
}
```

입력 : a b c d e f g 일 때
출력은 ?

의도적인 break 생략

영문 소문자 대문자 변환 관련

- 아스키코드에서
- 'a'=97, 'b'=98 … 'z' = 122
- 'A'=65(=97-32), 'B'=66(=98-32) … 'Z' = 90(=122-32)

- 문자형 변수는 숫자로도 사용 가능(→ 크기 비교 가능)
  - 다음은 영문 소문자 아닌 경우 참

```
cin >> letter;
if (letter < 'a' || letter > 'z')
```

- 영문 소문자를 대문자로 변경

```
letter -= 32;// 소문자 → 대문자
```

1000001	A	1100001	a
1000010	B	1100010	b
1000011	C	1100011	c
1000100	D	1100100	d
1000101	E	1100101	e
1000110	F	1100110	f
1000111	G	1100111	g
1001000	H	1101000	h
1001001	I	1101001	i
1001010	J	1101010	j
1001011	K	1101011	k
1001100	L	1101100	l
1001101	M	1101101	m
1001110	N	1101110	n
1001111	O	1101111	o

break, continue 예제

```
// 소문자를 대문자로 변경한다.
#include <iostream>
using namespace std;

int main()
{
  char letter;

  while(1)
  {
      cout << "소문자를 입력하시오: ";
      cin >> letter;      // 공백 문자 제외

      if( letter == 'Q' )
          break;
      if( letter < 'a' || letter > 'z' )
          continue;

      letter -= 32;                    // 소문자 -> 대문자
      cout << "변환된 대문자는 " << letter << "입니다.\n";
  }

  return 0;
}
```

break는 반복문을 탈출한다.

continue는 다음 반복을 시작한다.

소문자 아닌 경우 조사

실행 결과

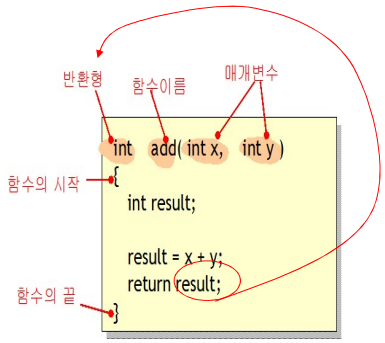
```
소문자를 입력하시오: a
변환된 대문자는 A입니다.
소문자를 입력하시오: b
변환된 대문자는 B입니다.
소문자를 입력하시오: c
변환된 대문자는 C입니다.
소문자를 입력하시오: Q
```

함수의 개념

- *함수(function)*: 특정한 작업을 수행하는 독립적인 부분
- *함수 호출(function call)*: 함수를 호출하여 사용하는 것
- 함수는 입력을 받으며 출력을 생성한다.

함수의 구조, 예제 #2

- 두개의 정수중에서 큰 수를 계산하는 함수

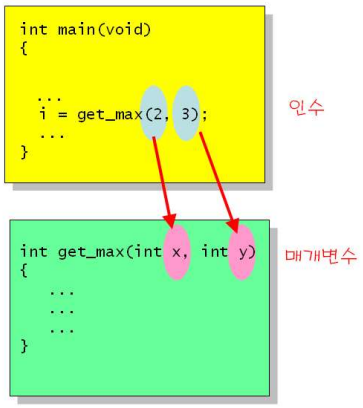


반환값: int
함수 이름: get_max
매개 변수: int x, int y

```
int get_max(int x, int y)
{
  if( x > y ) return(x);
  else return(y);
}
```

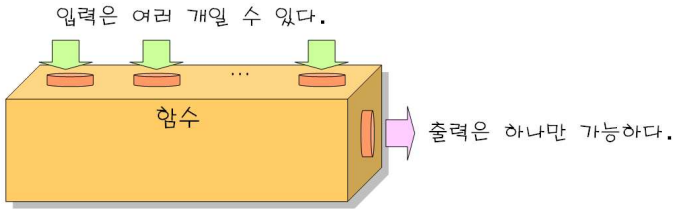
인수와 매개 변수

- *인수(argument):* 실인수, 실매개 변수라고도 한다.
- *매개 변수(parameter)*: 형식 인수, 형식 매개 변수라고도 한다.(인자)



반환값

- *반환값(return value)*: 호출된 함수가 호출한 곳으로 작업의 결과값을 전달하는 것
- 인수는 여러 개가 가능하나 반환값은 하나만 가능




```
return 0;
return(0);
return x;
return x+y;
return x*x+2*x+1;
```

함수 원형

- *함수 원형(function prototyping)*: 컴파일러에게 함수에 대하여 미리 알리는 것

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

int square(int n);                          // 함수 원형
int main()
{
    int i, result;
    for(i = 0; i < 5; i++)
    {
        result = square(i);                 // 함수 호출
        cout <<  result << endl;
    }
    return 0;
}
int square(int n)                           // 함수 정의
{
    return(n * n);
}
```

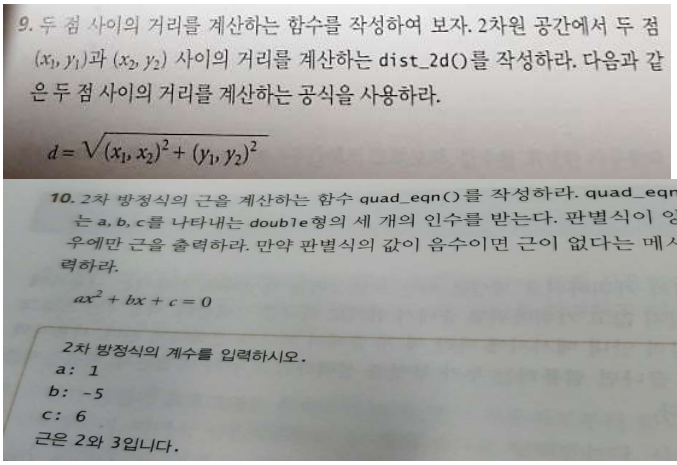
함수 원형

Report

- 제출처
  - DOOR → 수업결과 → 수업활동일지 → 해당 과제란
  - 수업활동 일지의 과제란에 있는 한글 파일에 코드 작성하여 제출
- 제출기한 엄수

Report

- 81쪽 9번 : 2점 거리 구하는 문제, 두점좌표 (x1, y1), (x2, y2) 입력 받음.
- 82쪽 10번 : 2차 방정식 근 구하는 문제,


9. 두 점 사이의 거리를 계산하는 함수를 작성하여 보자. 2차원 공간에서 두 점 (x_1, y_1)과 (x_2, y_2) 사이의 거리를 계산하는 dist_2d()를 작성하라. 다음과 같은 두 점 사이의 거리를 계산하는 공식을 사용하라.

$$d = \sqrt{(x_1, x_2)^2 + (y_1, y_2)^2}$$

10. 2차 방정식의 근을 계산하는 함수 quad_eqn()를 작성하라. quad_eqn() 함수는 a, b, c를 나타내는 double형의 세 개의 인수를 받는다. 판별식이 양수인 경우에만 근을 출력하라. 만약 판별식의 값이 음수이면 근이 없다는 메시지를 출력하라.

$$ax^2 + bx + c = 0$$

```
2차 방정식의 계수를 입력하시오.
a: 1
b: -5
c: 6
근은 2와 3입니다.
```

Report

- 81쪽 9번 : 2점 거리 구하는 문제, (x1, y1), (x2, y2) 입력 받음.
- 82쪽 10번 : 2차 방정식 근 구하는 문제,
  - 2차 방정식 계수 a, b, c 입력 받음.
- 필요 내용
  - #include <cmath>
  - sqrt( ) 함수
  - pow(x, 2) → x*x

```
void main(){
      p9( );
      P10();
}
```

```
xxx dist_2d(…){ … }

void P9(){  ….}


xxx quad_eqn(…){ … }

void P10(){  ….}
```

Report

```
void main(){
      p9( );
      P10();
}
```

```
void P9(){  ….}

void P10(){  ….}
```

```
xxx dist_2d(…){ … }

xxx quad_eqn(…){ … }
```

- P9() 함수 → 두점 거리 계산 위해 dist_2d( ) 함수 호출
  - dist_2d( ) 함수 호출시 두 점 좌표를 매개변수로 전달
  - dist_2d( ) 함수는 두 점간 거리를 반환
  - p9() 함수에서는 두 점 좌표, 거리를 출력
- p10() 함수 → 2차 방정식 근을 구하기 위하여 quad_eqn( ) 함수 호출
  - quad_eqn( ) 함수 호출시 이차 방정식 계수(a, b, c)를 매개변수로 전달
  - quad_eqn( ) 함수는 계수(a, b, c)와 계산한 근을 출력

중복 함수

- 중복 함수(overloading functions):
  - 같은 이름을 가지는 함수를 여러 개 정의하는 것
  - 매개변수(인자)가 달라야 함(개수, type) → 반환형은 관계없음

```
// 정수값을 제곱하는 함수
int square(int i)
{
    return i*i;
}
// 실수값을 제곱하는 함수
double square(double i)
{
    return i*i;
}
```

예제

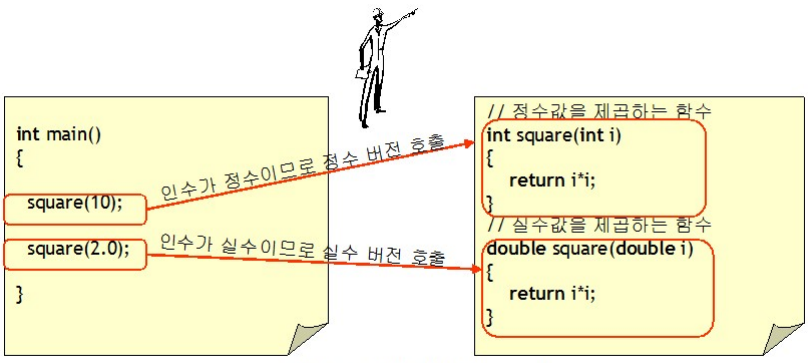


그림 2.9 중복 함수의 개념

중복 함수의 장점

- 중복 함수를 사용하지 않은 경우:

```
square_int(int int);
square_double(double int);
square_short(short int);
```

- 중복 함수를 사용하는 경우

```
square(int int);
square(double int);
square(short int);
```

함수 이름의 재사용이 가능

주의할 점

- int sub(int);

- int sub(int, int);// 중복 가능!
- int sub(int, double);// 중복 가능!
- double sub(double);// 중복 가능!

- double sub(int);// 오류!! - 반환형이 다르더라도 중복 안됨!
  // 인자 동일
- float sub(int, int);// 오류!! - 반환형이 다르더라도 중복 안됨!
  // 인자 동일

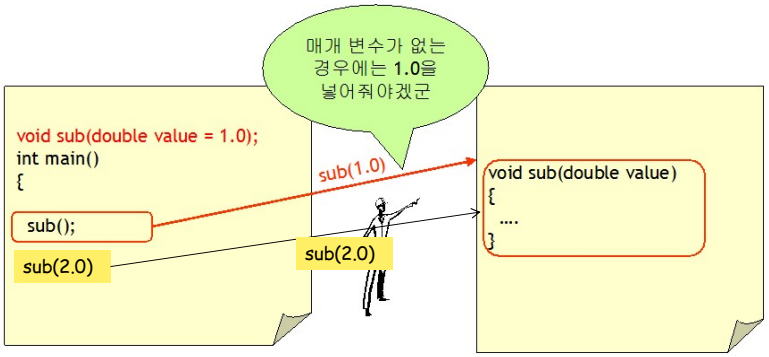
예제

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

// 정수값을 제곱하는 함수
int square(int i)
{
   cout << "square(int) 호출" << endl;
   return i*i;
}

// 실수값을 제곱하는 함수
double square(double i)
{
   cout << "square(double) 호출" << endl;
   return i*i;
}

int main(void)
{
   cout << square(10) << endl;
   cout << square(2.0) << endl;
   return 0;
}
```

디폴트 매개 변수

- 디폴트 매개 변수(default parameter): 매개변수 값을 전달하지 않아도 디폴트값을 대신 넣어주는 기능 → // 함수 원형에서 정의
  - 인수를 전달하면 전달된 인수값을 사용

```
void sub(double value = 1.0);          // 함수 원형에서 정의
```



주의할 점

- 디폴트 매개 변수는 뒤에서부터 앞쪽으로만 정의할 수 있다.

```
void sub(int p1, int p2, int p3=30);       // OK!          ... (1)
    호출 방법 → sub(1, 2);     sub(1, 2, 3);          → but, sub(1) 은 error

void sub(int p1, int p2=20, int p3=30);   // OK!
    호출 방법 → sub(1);        sub(1, 2);             sub(1, 2, 3);

void sub(int p1=10, int p2=20, int p3=30);          // OK!
    호출 방법 → sub();         sub(1);    sub(1, 2);             sub(1, 2, 3);

void sub(int p1, int p2=20, int p3);       // 컴파일 오류!  ... (2)
void sub(int p1=10, int p2, int p3=30);   // 컴파일 오류!
```

- sub(1, 2) 호출시
  (1) 에서 매개변수는 1, 2, 30 으로 받음
  (2) 에서 매개변수는 1, 2(?) ??? ← <매개변수는 순서대로 받음>

예제

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

int calc_deposit(int salary=300, int month=12);

int main()
{
    cout << "0개의 디폴트 매개 변수 사용"<< endl;
    cout << calc_deposit(200, 6) << endl;

    cout << "1개의 디폴트 매개 변수 사용" << endl;
    cout << calc_deposit(200) << endl;

    cout << "2개의 디폴트 매개 변수 사용" << endl;
    cout << calc_deposit() << endl;
    return 0;
}

int calc_deposit(int salary, int month)
{
    return salary*month;
}
```

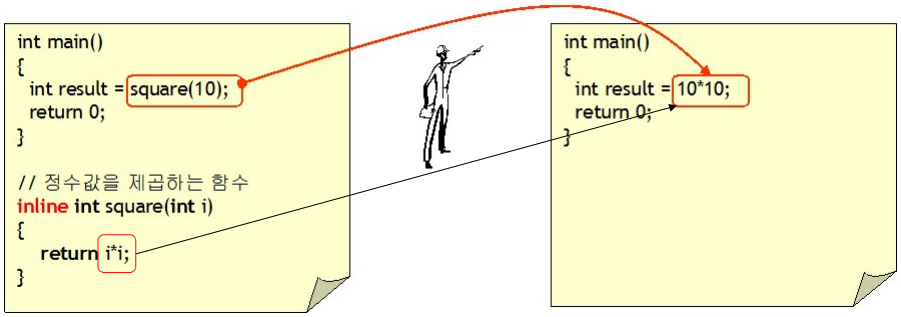
```
0개의 디폴트 매개 변수 사용
1200
1개의 디폴트 매개 변수 사용
2400
2개의 디폴트 매개 변수 사용
3600
계속하려면 아무 키나 누르십시오 . . .
```

인라인 함수

- 인라인 함수(inline function): 본체 길이 짧은 함수에 사용

- 함수 호출을 하지 않고 컴파일시 코드를 복사하여서 넣는 것
- 함수 호출하면 함수에 갔다가 와야 함. (시간 소요)

inline 인 경우에는 함수 몸체를 호출한 곳에 삽입합니다.



컴파일 결과

예제

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

// 실수값을 제곱하는 함수
inline double square(double i)
{
        return i*i;
}
int main()
{
        cout << "2.0의 제곱은" << square(2.0) << endl;
        cout << "3.0의 제곱은" << square(3.0) << endl;
        return 0;
}
```

```
2.0의 제곱은 4
3.0의 제곱은 9
계속하려면 아무 키나 누르십시오 . . .
```

중간 점검 문제

1. 다음의 함수 선언이 잘못된 이유를 설명하라.

```
int moveto(int x=0, int y, int z=0);
```

2. 다음과 같은 함수 선언을 가지는 함수 호출 중 잘못된 것은?

```
int sub(int a, int b=100, int c=100);
```

(a) sub(0, 0, 0); (b) sub(0); (c) sub() (d) sub(0, 0);

3. 다음은 중복 함수와 디폴트 매개 변수를 함께 사용한 함수 선언이다. 잠재적인 문제는 무엇인가?

```
void print(char *s=NULL);
void print(void);
```

→ print( ); 하면 어느 것이 수행 ??

C++ 에서는 어디서나 지역 변수 선언이 가능

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

int fib(int n)
{
  int fib0 = 0, fib1 = 1;
  for (int i = 1; i <= n; i++)
  {
    int tmp = fib0 + fib1;
    fib0 = fib1;
    fib1 = tmp;
  }
  return fib0;
}
```

C++언어에서 지역 변수는 어디서나 선언이 가능하다.

for 문 안에서도 지역 변수의 선언이 가능하다.

블록 안에서도 지역 변수의 선언이 가능하다.

전역 변수

- 전역 변수(global variable): 함수의 외부에 선언되는 변수

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

int x = 1;          // 전역 변수

void sub()
{
   x++;
}
int main(void)
{
   x++;
   ...
}
```

전역 변수는 어디서나 접근이 가능하다.

저장 유형 지정자 static(정적 변수)

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

void sub(void)
{
        int i = 0;
        static int s = 0;

        i++;
        s++;
        cout << "i: " << i << " s: " << s << endl;
} // 함수 나가면서 지역변수 i 는 파괴(내용도 없어짐),
  // 정적변수는 계속 존재(값 유지)
int main()
{
        sub();
        sub();
        sub();
        return 0;
}
```

```
i: 1 s: 1
i: 1 s: 2
i: 1 s: 3
계속하려면 아무 키나 누르십시오 . . .
```

static을 붙이면 지역 변수가 정적 변수로 된다.

정적(static) 변수 - 전역 변수

```
int count = 0;          // 전역변수
void func1( void ){
    printf( "%d ₩n", ++count );
}

void main( void ){
    func1();
    count = 9;    // 전역 변수에 접근이 가능
    func1();
}

<결과>
count = 1
count = 10
```

```
void func1( void ){
    static int count = 0;
    printf( "%d ₩n", ++count );
}

void main( void ) {
    func1();
   // count = 9;   ,  컴파일 에러,
    func1();
}
<결과>
count = 1
count = 2
```

=> 전역변수와 static 변수는 비슷하게 동작(함수 빠져 나와도 값이 남아 있음)

- 전역변수는 모든 함수가 사용할 수 있는 변수이다.
- static 변수는 전역변수이다.
  - 단, 선언한 함수만 사용(접근)할 수 있는 전역변수이다.
  - ➔ **"전역 변수를 선언하고, 특정 함수에서만 사용하고 싶을 때"' 사용**

재귀함수, 순환(recursion)이란?

- 생략 ➔ 따로 공부하지 않아도 됨
- 알고리즘이나 함수가 수행 도중에 자기 자신을 다시 호출하여 문제를 해결하는 기법
- 팩토리얼의 정의 $n! = \begin{cases} 1 & n = 1 \\ n * (n-1)! & n \geq 2 \end{cases}$
- 함수 작성법

```
int fact(int n) {
        if(n==1)  return 1;
        else      return( n * fact(n-1) );
}
```

Q & A