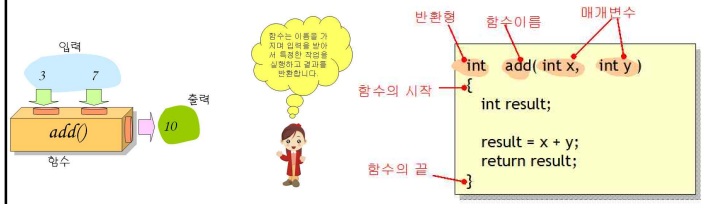
제2장 함수

이번 장에서 학습할 내용

- 함수 개념
- 함수 정의
- 함수 원형
- 매개 변수
- 순환 호출

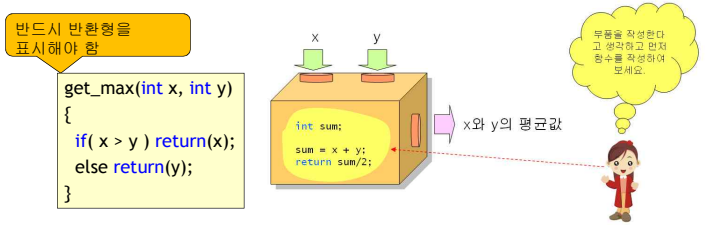
함수의 개념과 구조

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



반환형

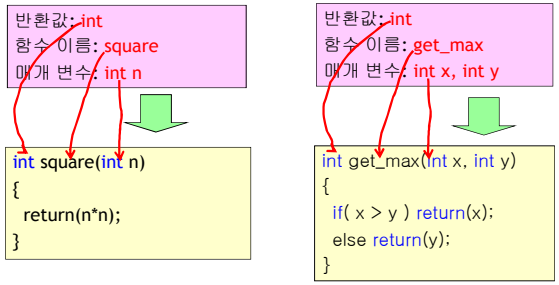
- C언어에서는 만약 반환형을 명시하지 않으면 int 형을 가정하였다.
- 하지만 C++에서는 반드시 반환형을 명시하여야 한다.
- 만약 값을 반환하지 않는다면 void라고 표시한다.



- 함수를 프로그램을 이루는 부품이라고 가정하자.
- 입력을 받아서 작업한 후에 결과를 생성한다.

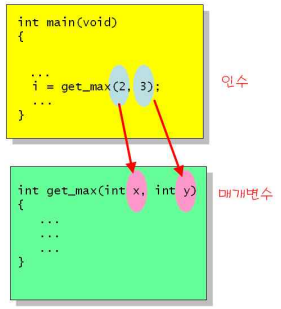
예제

- 정수의 제곱값을 계산하는 함수
- 두 개의 정수중에서 큰 수를 계산하는 함수



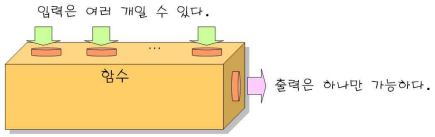
인수와 매개 변수

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



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);
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

함수로 만들자

- 간단한 프로그램
  - 섭씨 화씨 변환
    - 입력:
    - 출력:
  - 세금 계산
    - 입력:
    - 출력:
  - 소수 찾기 프로그램
    - 입력:
    - 출력:
- **함수에서 입 출력문은?**

참조자(reference)

- 참조자(reference): 변수에 별명을 붙이는 것

  int &ref = var;
- "참조자 ref은 변수 var의 별명(alias)이다"

예제

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

int main()
{
        int var;
        int &ref = var;              // 참조자선언

        var = 10;
        cout << "var의값: " << var << endl;
        cout << "ref의값: " << ref << endl;

        ref = 20;          // ref의 값을 변경하면 var의 값도 변경된다.
        cout << "var의값: " << var << endl;
        cout << "ref의값: " << ref << endl;

        return 0;
}
```

```
var의 값: 10
ref의 값: 10
var의 값: 20
ref의 값: 20
```


예제

- int n=10, m=20;
- int &ref = n;
- ref = m;// 컴파일 오류!!
- int &ref;// 컴파일 오류!!
- int &ref = 10;// 컴파일 오류!!

변경 불가

초기화되지 않았음

상수로 초기화할 수 없음

call-by-value와 call-by-reference

- call-by-value와 call-by-reference

```
void swapValue(int x, int y)
{
        int tmp;
        tmp = x;
        x = y;
        y = tmp;
}
```

```
void swapReference(int& x, int& y)
{
        int tmp;
        tmp = x;
        x = y;
        y = tmp;
}
```

```
// call-by-value와 call-by-reference로 두 변수의 교환을 시도하는 함수 테스트
main()
{
        int a=1, b=2;
        printf("swap을 호출하기 전: a=%d, b=%d\n", a,b);
        swapValue(a, b);                    // call-by-value: a와 b가 바뀌지 않음
        printf(" swapValue()을 호출한 다음: a=%d, b=%d\n", a,b);
        swapReference(a, b);  // call-by-reference: a와 b가 바뀜
        printf("swapReference()를 호출한 다음: a=%d, b=%d\n", a,b);
}
```


여러 개의 값을 반환하는 함수??

- 여러 개의 결과 반환
  - 이차방정식 근을 구하는 함수
    - 입력:
    - 출력:
    - 함수의 원형은?
  - 두 점의 좌표가 주어졌을 때 선분의 방정식 구하기
    - 입력: x1, y1, x2, y2
    - 출력: y=ax+b 의 a와 b
    - 함수의 원형은?

중복 함수

- 중복 함수(overloading functions):
  같은 이름을 가지는 함수를 여러 개 정의하는 것

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


예제

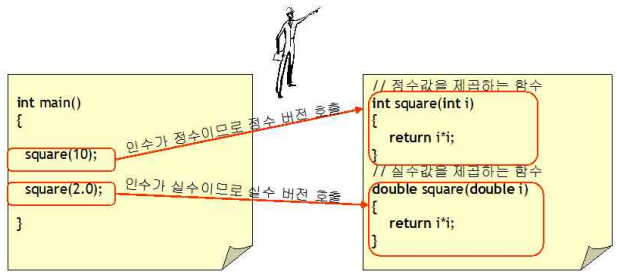


그림 2.9 중복 함수의 개념

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

예제

```
int add ( int a, int b )
{
        int sum;
        sum = a + b;
        return sum;
}
```

```
float add ( float a, float b )
{
        float sum;
        sum = a + b;
        return sum;
}
```

```
double add ( double a, double b )
{
        float sum;
        sum = a + b;
        return sum;
}
```

```
main()
{
        int r1 = add(1, 2);
        float r2 = add(1.0f, 2.0f);
        double r3 = add(1.0, 2.0);
}
```

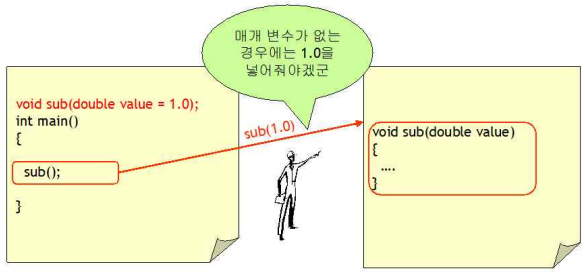

다양한 구구단 함수

- 한 단을 출력하는 함수
  - 매개변수:
  - 반환형:
  - 함수 원형:
- 한 줄에 여러 단이 출력되도록 하는 함수
  - 매개변수:
  - 반환형:
  - 함수 원형:
- 19구단을 출력하는 함수

디폴트 매개 변수

- 디폴트 매개 변수(default parameter): 인자를 전달하지 않아도 디폴트값을 대신 넣어주는 기능

```
void sub(double value = 1.0);        // 함수 원형 정의시
```



주의할 점

- 예:

  void drawLine(int x1, int y1, int x2, int y2, int r, int g, int b, int lw, int lStyle);

  ➔ 이 함수를 어떻게 사용하지???

  void drawLine(int x1, int y1, int x2, int y2, int r=255, int g=255, int b=255, int lw=1, int lStyle=PS_SOLID);

- 디폴트 매개 변수는 뒤에서부터 앞쪽으로만 정의할 수 있다.

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


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 . . .
```

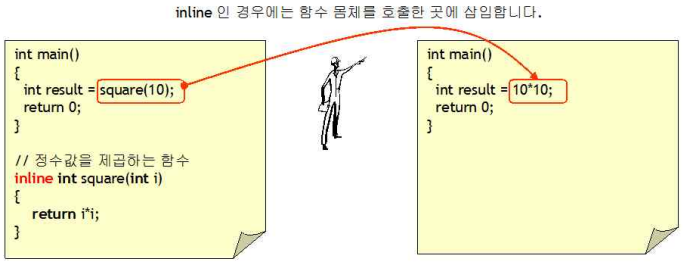

디폴트 구구단 처리 함수

- 한 줄에 여러 단이 출력되도록 하는 함수
  - 함수 원형:

인라인 함수

- 인라인 함수(inline function):

함수 호출을 하지 않고 코드를 복사하여서 넣는 것



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 . . .
```


함수 매크로와의 차이점

- 함수 매크로는 기계적인 대치라서 잘못 사용될 수 있다.
  #define SQUARE(x) (x*x)
  SQUARE(y++); //-> y++*y++로 확장되어서 y의 값이 2번 증가된다.
- 함수 매크로에서는 타입 검사를 할 수 없다.

인라인 함수

- 판별식 구하는 함수
  - 입력:
  - 출력:
  - 함수의 원형은?
- 최대값, 최소값, 절대값을 구하는 함수
  - 입력:
  - 출력:
  - 함수의 원형은?
- 주사위 던지기 함수
  - 입력:
  - 출력:
  - 함수의 원형은?

변수의 범위

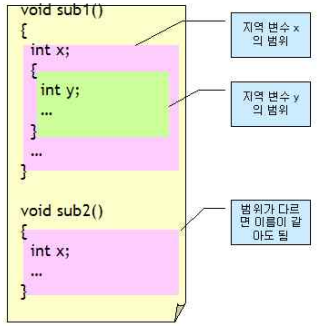
- 블록 안에서만 정의되는 변수는 지역 변수(local variable)
- 함수의 외부에서 선언되는 변수는 전역 변수(global variable)

지역변수

전역변수

지역 변수

- 지역 변수(local variable): 블록 안에서 선언되는 변수



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

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

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

        i++;
        s++;
        cout << "i: " << i << " s: " << s << endl;
}

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.

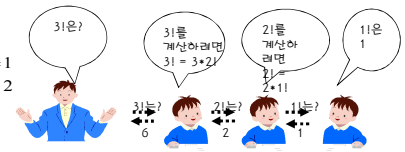
중간 점검 문제

1. 지역 변수를 블록의 중간에서 정의할 수 있는가?
2. 똑같은 이름의 지역 변수가 서로 다른 함수 안에 정의될 수 있는가?
3. 지역 변수가 선언된 블록이 종료되면 지역 변수는 어떻게 되는가?
4. 지역 변수의 초기값은 얼마인가?
5. 전역 변수는 어디에 선언되는가?
6. 전역 변수의 생존 기간과 초기값은?
7. 똑같은 이름의 전역 변수와 지역 변수가 동시에 존재하면 어떻게 되는가?

순환(recursion)이란?

- 알고리즘이나 함수가 수행 도중에 자기 자신을 다시 호출하여 문제를 해결하는 기법
- 팩토리얼의 정의

$$n! = \begin{cases} 1 & n = 1 \\ n*(n-1)! & n \geq 2 \end{cases}$$

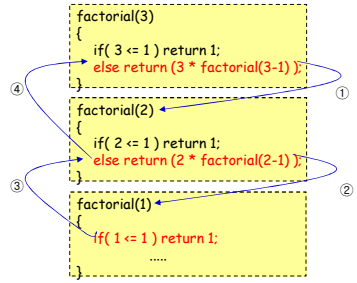


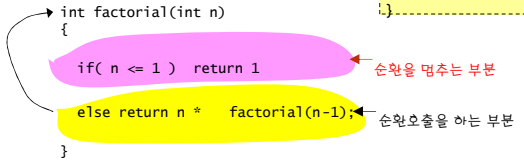

```
int factorial(int n)
{
   if( n <= 1 ) return(1);
   else return (n * factorial(n-1) );
}
```


팩토리얼 구하기

- 팩토리얼의 호출 순서

```
factorial(3) = 3 * factorial(2)
             = 3 * 2 * factorial(1)
             = 3 * 2 * 1
             = 3 * 2
             = 6
```





함수를 호출한 횟수

- 피보나치 수열을 구하는 순환 함수
  - 함수가 호출된 횟수가 출력되려면?
    - 전역변수 (배열) 사용
    - 최초에 초기화
- 하노이탑 문제
  - 함수가 호출된 횟수가 출력되려면?

게임을 위해 필요한 함수

- 난수 발생

난수 발생 함수

- stdlib.h 파일을 소스에 포함시킴

  #include <stdlib.h> // stdlib.h에 관련 함수들이 정의되어 있음

- rand() 함수 호출

  0 ~ RAND_MAX 사이의 임의의 정수 반환

seed 사용

- time.h 파일을 추가로 포함시킴 // time() 함수 사용을 위해

  #include <time.h> // time.h에 time() 함수가 정의되어 있음

- srand(time(NULL)) 함수 호출

  현재 시각 정보를 이용해 seed 설정

  프로그램 시작부에서 한번만 호출하면 됨 (여러 번 호출 할 필요 없음)

게임을 위해 필요한 함수

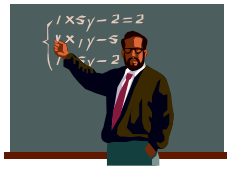
- 실행시간 측정

```
#include <cstdio>
#include <cstdlib>
#include <ctime>
void main( void )
{
        clock_t start, finish;
        double duration;
        start = clock();
        // 실행시간을 측정하고 하는 코드....
        // ....
        finish = clock();
        duration = (double)(finish - start) / CLOCKS_PER_SEC;
        printf("%f 초입니다.\n", duration);
}
```


프로그래밍 프로젝트

- 랜덤 함수 사용
- 조금은 어려워진 게임
  - 구구단 게임
- 약간은 게임 같은 게임
  - 가위바위보
  - 주사위 게임
  - 파워볼 게임 기초
  - 윷놀이 게임 기초

Q & A



41/40