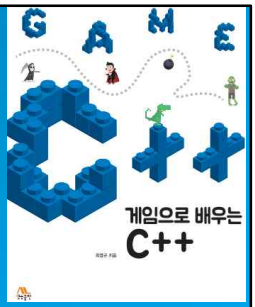
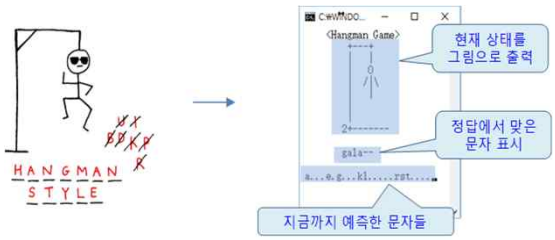
게임으로 배우는 C++



CHAPTER

06 클래스:구조체의 진화

행맨(hangman) 게임



- Hangman
  - 영어 단어 맞히기 게임
  - 문제로 주어진 단어에 들어갈 알파벳을 하나씩 추측함
  - 추측한 알파벳이 있으면 그 알파벳의 위치를 공개
  - 없으면 교수대에 캐릭터의 하나씩 나타남
  - 제한된 횟수 만에 맞히지 못하면 게임은 실패

2

6장 학습 목표

- 객체지향 프로그래밍의 개념과 클래스와 객체를 이해한다.
- 객체지향의 주요 개념들을 이해한다.
- 클래스와 객체의 선언과 활용 방법을 이해한다.
- 클래스를 구현하는 다양한 방법을 이해한다.
- UML과 클래스 다이어그램을 이해한다.
- C++ 표준 라이브러리에서 제공하는 입출력 객체와 파일 처리, string 클래스를 이해하고 활용할 수 있는 능력을 기른다.

3

6.1 클래스: 구조체의 진화

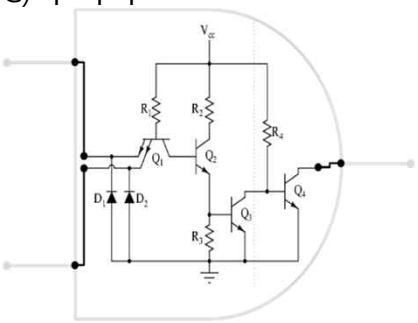
- 객체지향 프로그래밍이란?
- 클래스 객체와 IC부품 비교
- 구조체, 클래스, 객체 (Object)
- 객체 지향적 프로그래밍으로 구현

4

객체지향 프로그래밍이란?

- 소프트웨어를 집적회로(IC)화 하자

AND 게이트



- C++은 소프트웨어 개발에 모듈성, 유연성, 재사용성을 높이기 위해 추상화, 캡슐화, 상속, 다형성 등의 개념을 지원

5

클래스 객체와 IC부품 비교

반도체 부품	클래스의 객체
복잡한 기능의 회로를 개발하기 위해 반도체 부품들을 사용한다.	복잡한 소프트웨어를 개발하기 위해 클래스의 객체들을 사용한다.
칩의 내부 회로가 복잡해도 사용 방법은 간단함	클래스의 코드가 복잡해도 사용하는 방법은 간단함.
내부에 다양한 전압과 전류 값이 있지만 외부에서 직접 접근불가. 외부로 노출된 단자들 만을 사용.	많은 변수나 함수가 있지만 외부에서 모드 직접 접근할 수는 없음. 공개된 함수나 변수만 사용.
사용자가 실수할 수 있는 부분들을 감출 수 있으므로 칩을 오작동 시킬 가능성이 줄어듦.	잘못 사용할 수 있는 변수나 함수를 외부로부터 감춤. 프로그램의 오류 발생 가능성이 줄어듦.
동일한 인터페이스를 가진 더 좋은 부품이 나오면 그 부품만을 교체 가능.	개선된 버전이 나오면 그 클래스만 바꾸어 프로그램을 빌드 및 사용 가능.

6

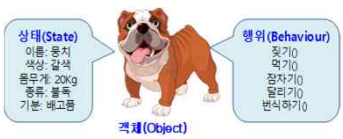
객체지향 프로그래밍(OOP)

- object-oriented programming
  - 실세계가 객체(object)로 구성되어 있는 것과 비슷하게 소프트웨어를 개발하고자 함
  - 상향식(bottom-up) 프로그래밍 기법 : 작은 모듈들을 먼저 만들고, 이들을 조합하여 원하는 큰 프로그램을 개발하는 방법
  - 소프트웨어 모듈의 핵심이 클래스
- "객체지향"이 더 프로그래밍하기가 쉬울까?
  - 개발자
    - 이전보다 할 일이 많아짐. 더 안전하고 쓰기 쉬운 클래스를 설계하고 구현
    - 사용자의 실수 가능성을 예상하여 적절히 감추고 필요한 부분만 공개
  - 사용자
    - 프로그램이 쉬워짐. 반도체 부품을 사용해 회로를 꾸미는 것과 동일
    - 클래스 내부의 복잡한 동작은 알 필요가 없음.
    - 클래스의 기능과 성능만을 이해하고, 적절한 인터페이스만 공부하고 구현

7

구조체, 클래스, 객체

- 객체는 상태와 행위를 가짐

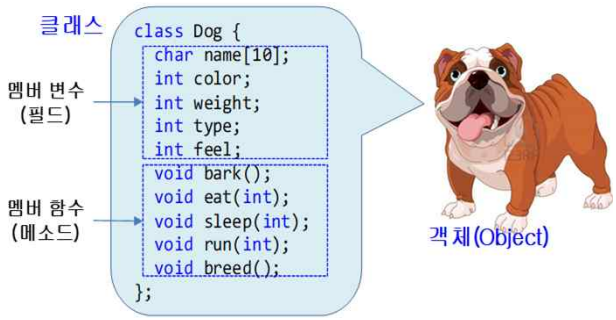


- **절차 지향적** 프로그래밍으로 구현: 구조체+함수

8

객체 지향적 프로그래밍으로 구현

- 구조체 데이터와 함수들을 묶는 것 ➔ 클래스
  - 속성: 멤버 변수(member variable) 또는 필드(field)
  - 행위: 멤버 함수(member function) 또는 메소드(method)

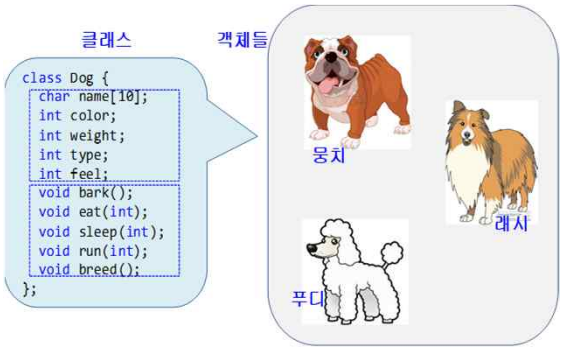


9

객체(Object)란?

- 클래스의 사례(instance)
  - 변수

```
int      x;
Dog      mungchi;
Dog      puddy;
Dog      Rashi;
Complex  a;
```



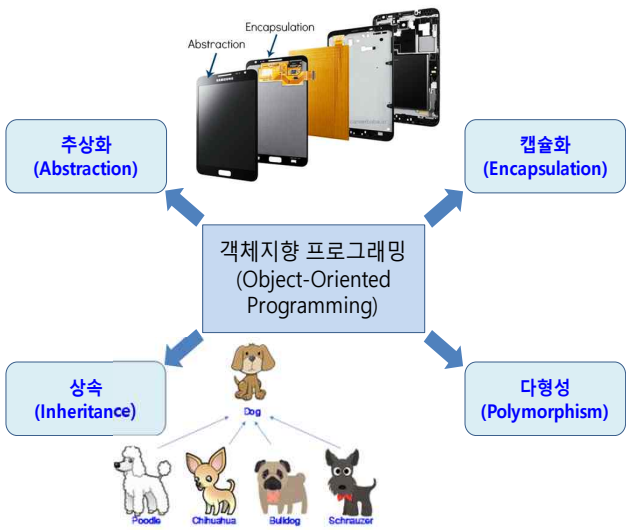
10

6.2 객체지향 프로그래밍의 주요 특징

- 추상화
- 캡슐화
- 상속
- 다형성

11

객체지향 프로그래밍의 주요 특징



12

6.3 클래스의 선언과 활용

- 클래스의 선언과 활용
- 객체의 생성과 멤버 접근

클래스의 선언과 활용

- 클래스 선언: class 또는 struct
  - 멤버 변수와 멤버 함수를 클래스 블록에 포함
  - 멤버 선언 위치에 상관없이 사용 가능

```
class 클래스명 {          // 새로운 클래스를 선언
private:                  // 멤버 접근 지정자(public, private, protected)
      멤버변수1;          // 멤버 변수는 객체의 속성을 나타냄
      멤버변수2;
      …
public:                   // 멤버에 대한 접근 지정자
      멤버함수1;          // 멤버 함수는 객체의 동작을 나타냄
      멤버함수2;
      …
};                        // 세미콜론(';')를 잊지 말아야 함.
```

클래스의 선언과 활용

- 접근 지정자
  - private: 전용. 외부에서 접근 불가
  - protected: 보호. 자식 클래스까지 접근 허용. 외부 접근 불가
  - public: 공용. 누구나 접근 가능
- class, struct 기본 접근 지정자만 다름
  - class: 전용(private)
  - struct: 공용(public)
  - 클래스 블록이 끝나면 반드시 세미콜론(;)

```
struct Complex
{
  // 클래스 몸체
};
```

정확히 동일 한 선언

```
class Complex
{
public:
  // 클래스 몸체
};
```


객체의 생성과 멤버 접근

- 복소수 클래스의 선언과 사용 예

```
class Complex {
private:
  double  re;       // 실수부
  double  im;       // 허수부
public:
  void set(double r, double i) {
    re = r;
    im = i;
  }
  void print() {
    printf("%4.1f +%4.1fi \n",
                    re, im);
  }
  double mag () {
    return sqrt(re*re + im*im);
  }
};
```

private 클래스 전용 멤버

클래스 전용 멤버는 클래스 내에서만 사용할 수 있음

클래스 전용 멤버는 클래스 외부에서 **접근 불가**

public 공용 멤버

```
void main()
{
  Complex a, b;
  a.set(1.0, 2.0);
  b.set(3.0, 4.0);
  a.print();
  b.print();
  printf("a의 크기=%lf\n",
                  a.mag ());
  a.re = 5.0; // 잘못된 문장
}
```

공용 멤버는 클래스 외부에서도 사용 가능

  - 객체 a의 멤버 함수 set을 호출
  - 객체 a에게 set하라는 메시지를 보냄(필요한 정보는 인수로)

6.4 사례: Complex의 다양한 변신

- Complex V1: 구조체와 일반 함수로 구현한 복소수
- Complex V2: 복소수를 클래스로 전환 (데이터 + 함수)
- Complex V3: 멤버 이름의 단순화
- Complex V4: 모든 멤버 함수를 inline으로 구현

17

Complex V1: 구조체와 함수

Complex.h

```
#pragma once
#include <cstdio>
struct Complex {
  double   real;
  double   imag;
};
inline void setComplex(Complex &c, double r, double i){
  c.real = r;
  c.imag = i;
};
extern Complex readComplex(char* msg = " 복소수 =");
extern void printComplex(Complex c, char* msg = "복소수 = ");
extern Complex addComplex(Complex a, Complex b);
```

Complex 구조체 선언

inline 함수 선언

함수 원형 선언

Complex.cpp

```
#include "Complex.h"
Complex readComplex( char* msg ) {
  Complex c;
  printf(" %s ", msg);
  scanf("%lf%lf", &c.real, &c.imag);
  return c;
}
void printComplex(Complex c, char* msg) {
  printf(" %s %4.2f + %4.2fi\n", msg, c.real, c.imag);
}
Complex addComplex(Complex a, Complex b) {
  Complex c;
  c.real = a.real + b.real;
  c.imag = a.imag + b.imag;
  return c;
}
```

복소수 객체 입력 함수

복소수 객체 출력 함수

두 객체의 합을 구해 반환하는 함수

ComplexTest.cpp

```
#include "Complex.h"
void main()
{
  Complex a, b, c;
  a = readComplex ( "A =" );
  b = readComplex("B =");
  c = addComplex(a, b);
  printComplex( a, " A = ");
  printComplex( b, " B = ");
  printComplex( c, " A+B = ");
}
```

Complex 객체 생성

복소수 객체 a와 b의 정보 입력

복소수 덧셈 연산 c = a + b

복소수 객체 출력

18

Complex V2: 클래스로 변환

Complex.h

```
#pragma once
#include <cstdio>
class Complex
{
  double   real;
  double   imag;
public:
  void setComplex( double r, double i ) {
    real = r;
    imag = i;
  }
  void readComplex( char* msg = " 복소수 = " );
  void printComplex( char* msg = "복소수 = " );
  void addComplex ( Complex a, Complex b );
};
```

데이터 멤버. 모두 private 로 선언됨

inline 으로 구현된 멤버 함수. 매개변수가 하나 줄고, 코드도 단순해 짐.

멤버 함수들

모든 멤버 함수에 범위 연산자(::)가 적용됨 모두 Complex 클래스의 멤버 함수임을 나타냄

Complex.cpp

```
#include "Complex.h"
void Complex::readComplex( char* msg ) {
  printf(" %s ", msg);
  scanf("%lf%lf", &real, &imag);
}
void Complex::printComplex( char* msg ) {
  printf(" %s %4.2f + %4.2fi\n", msg, real, imag);
}
void Complex::addComplex( Complex a, Complex b ) {
  real = a.real + b.real;
  imag = a.imag + b.imag;
}
```

반환형이 void로 바뀜 scanf() 의 인수가 단순해 짐

매개변수가 줄어듦 직접 real, imag 사용

반환형이 void로 바뀜 직접 real, imag 사용

ComplexTest.cpp

```
#include "Complex.h"
void main()
{
  Complex a, b, c;
  a.readComplex ( "A =" );
  b.readComplex ( "B =" );
  c.addComplex (a, b);
  a.printComplex( " A = ");
  b.printComplex( " B = ");
  c.printComplex( " A+B = ");
}
```

Complex 객체 생성

객체에게 복소수 값을 읽으라는 메시지를 보냄. 처리 결과는 그 객체에 저장됨. 반환 불필요

객체 c에게 a와 b를 더하라는 메시지를 보냄. 결과는 c에 저장되므로 반환이 필요 없음.

객체에게 자신의 정보를 출력하라는 메시지를 보냄. 객체를 매개변수로 보낼 필요가 없음.

19

Complex V3: 이름 단순화

Complex.h

```
#pragma once
#include <cstdio>
class Complex
{
  double   real;
  double   imag;
public:
  void set ( double r, double i ) {
    real = r;
    imag = i;
  }
  void read ( char* msg = " 복소수 = " );
  void print ( char* msg = " 복소수 = " );
  void add ( Complex a, Complex b );
};
```

일반 멤버 함수들의 이름을 단순하게 변경함. 이 클래스에서만 의미가 있으므로 예를 들어, readComplex() 대신에 read()만 하더라도 의미가 명확함

Complex.cpp

```
#include "Complex.h"
void Complex::read ( char* msg ) {
  printf(" %s ", msg);
  scanf("%lf%lf", &real, &imag);
}
void Complex::print ( char* msg ) {
  printf(" %s %4.2f + %4.2fi\n", msg, real, imag);
}
void Complex::add ( Complex a, Complex b ) {
  real = a.real + b.real;
  imag = a.imag + b.imag;
}
```

함수 이름 단순화에 따른 수정

ComplexTest.cpp

```
#include "Complex.h"
void main()
{
  Complex a, b, c;
  a.read ( "A =" );
  b.read ( "B =" );
  c.add (a, b);
  a.print ( " A = ");
  b.print ( " B = ");
  c.print ( " A+B = ");
}
```

멤버 함수 이름의 단순화에 따라 호출 코드도 간단해 짐.

20

Complex V4: inline 구현

Complex.h

```
#pragma once
#include <cstdio>
class Complex
{
  double   real;
  double   imag;
public:
  void set ( double r, double i ) {
    real = r;
    imag = i;
  }
  void read ( char* msg ="복소수 = ") {
    printf(" %s ", msg);
    scanf("%lf%lf", &real, &imag);
  }
  void print ( char* msg " 복소수 = ") {
    printf(" %s %4.2f + %4.2fi\n", msg, real, imag);
  }
  void add ( Complex a, Complex b ) {
    real = a.real + b.real;
    imag = a.imag + b.imag;
  }
};
```

모든 멤버함수를 inline으로 구현.
Complex.cpp가 필요 없음

ComplexTest.cpp

```
#include "Complex.h"
void main()
{
  Complex a, b, c;
  a.read ( "A =" );
  b.read ( "B =" );
  c.add (a, b);
  a.print ( " A = ");
  b.print ( " B = ");
  c.print ( " A+B = ");
}
```

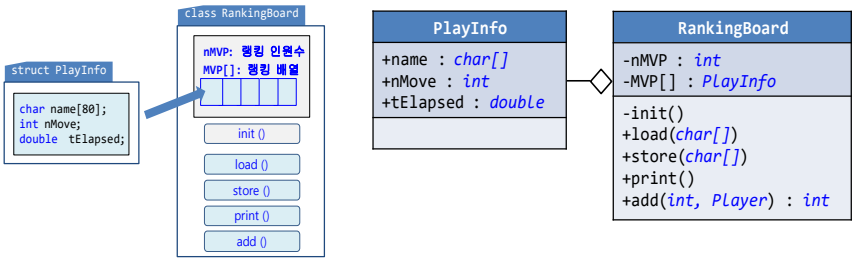
멤버 함수 이름의 단순화에 따라
호출 코드도 간단해 짐.

6.5 UML 클래스 다이어그램

- UML 클래스 다이어그램

UML 클래스 다이어그램

- 클래스의 구성과 관계를 나타내는 효율적인 방법
  - UML은 통합 모델링 언어(Unified Modeling Language)
- UML 클래스 다이어그램
  - - : 멤버의 접근 범위가 private
  - + : 멤버의 접근 범위가 public
  - # : 멤버의 접근 범위가 protected

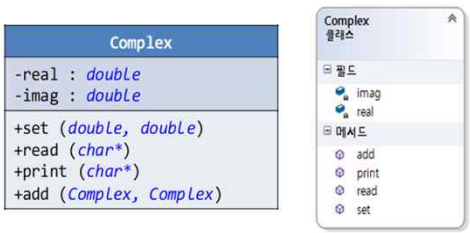


UML 클래스 다이어그램

- 클래스 사이의 관계를 나타내는 UML 심벌들

관계	UML Symbol	의미	예
inheritance	——▷	is-a	A book is a printed resource.
aggregation	——◇	has-a	A book has a publisher.

  - UML 클래스 다이어그램 / Visual C++의 클래스 다이어그램



6.6 응용: 기존 게임을 클래스로 변경

- Lab: 숫자 맞히기 게임의 클래스 변환
- Lab: 랭킹보드 클래스
- Lab: 4x4 퍼즐 게임의 클래스 변환

Lab: 숫자 맞히기 게임의 클래스 변환

UpAndDown.h

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
float playUpAndDown(int nDigit, int nMaxTry);
```

필요한 헤더 파일은 동일하게 포함

게임 클래스 정의

UpAndDown.cpp

```
#include "UpAndDown.h"
static int nMin, nMax;
static int nTries;
static void init(int nDigit){ … }
static int getRandomNumber(int maxNum ){ … }
float playUpAndDown(int nDigit, int nMaxTry){
 …
}
```

전역 변수를 멤버 변수로 변경. static은 없앰(다른 의미로 사용).
모두 private멤버(기본)

일반 함수를 멤버함수로. static 제거, 모두 private.

지금부터 public

이 멤버 함수만 public이 되어 외부에서 접근할 수 있음. 이름을 단순하게 변경함.

UpAndDown.h

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
class UpAndDown {
 int nMin, nMax;
 int nTries;
 void init(int nDigit) { … }
 int getRandomNumber(int maxNum ) { … }
public:
 float play(int nDigit, int nMaxTry) {
  …
 }
};
```

전역변수→멤버변수
함수→멤버함수
접근권한→static여부

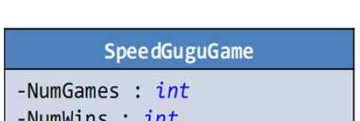
UpAndDownGame.cpp

```
#include "UpAndDown.h"
void main() {
 clock_t t1 = clock();
 float score = playUpAndDown(2, 8);
 clock_t t2 = clock();
 …
}
```

호출 방법이 바뀜.
먼저 객체를 만들고
객체에게 메시지를 보냄

UpAndDownGame.cpp

```
#include "UpAndDown.h"
void main() {
 UpAndDown game;
 clock_t t1 = clock();
 float score = game.play (2, 8);
 clock_t t2 = clock();
 …
}
```



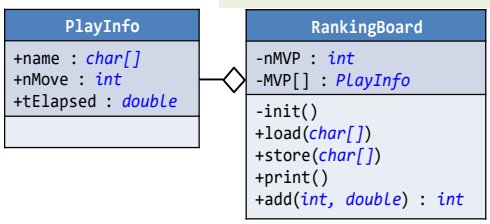
27

Lab: 랭킹보드 클래스

- 게임을 위한 고득점 랭킹 관리 클래스

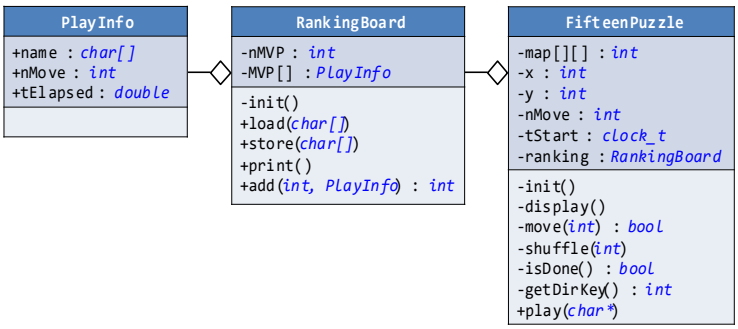
```c
#include <stdio.h>
#define NUM_MVP  5

struct PlayInfo {
  char   name[200];
  int    nMove;
  double tElapsed;
};
```

```c
class RankingBoard {
  PlayInfo MVP[NUM_MVP];
  int nMVP = NUM_MVP; // VS2013, 2010 ??
  void init() {...}
public:
  void load(char* filename) {...}
  void store(char* filename) {...}
  void print() {...}
  int add(int nMove, double tElap) {...}
};
```



28

Lab: 4x4 퍼즐 게임의 클래스 변환



29

6.7 C++ 표준 라이브러리 클래스

- 표준 입출력 객체와 이름 공간
- 입출력 조작자
- 파일 스트림 처리
- 문자열 클래스(string)

30

표준 입출력 객체와 이름 공간

- 표준 입출력 객체: cin, cout
  - cin은 표준 입력을 담당하는 istream 클래스의 객체.
  - cout은 표준 출력을 담당하는 ostream 클래스의 객체.
  - 헤더파일 <iostream>
  - >> 와 <<의 시프트 연산자를 중복하여 입출력 대상 지정.

- 이름 공간(name space)

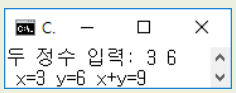
```
using namespace std;
```

- 범위 연산자 ::

```
std::cout
```


- 표준 입출력 객체를 사용하는 예

```
#include <iostream>                  // #include <stdio.h>
using namespace std;
void main()
{
  int x, y;
  cout << "두 정수 입력: ";          // printf("두 정수 입력: ");
  cin >> x >> y;                     // scanf("%d%d", &x, &y);
  cout <<" x="<< x <<" y="<< y;      // printf(" x=%d y=%d x+y=%d\n",
    << " x+y=" << x+y << endl;       //              x, y, x+y);
}
```




```
  cin >> x >> y;                     // stdin으로 부터 x 와 y를 읽음
  cout << "x=" << 3 << endl;         // stdout으로 한 라인(endl) 출력

  cerr << x << y << flush;           // stderr로 출력, 버퍼를 비움(flush)
  c = cin.get();                     // c = getchar();
  cin.get(c);                        // char를 읽음
  cin.getline(s, n, '\n');           // '\n'(default) 까지 한 라인을 읽음
  if (cin)                           // 파일의 끝이 아닌지 검사 (not EOF?)
```

입출력 조작자

- 입출력 조작자(manipulator)
  - 입출력 객체에서 특정한 형식을 사용하기 위함.
  - 헤더파일 <iomanip>를 포함

```
#include <iomanip>
...
cout << 2017 << endl;                  // "2017"을 출력
cout << setw(8) << 2017 << endl;       // "    2017"을 출력
cout << fixed;                         // 고정 소숫점 표기 설정
cout << setw(8)                        // 8자리로 표기
     << setprecision(3)                // 소숫점 이하 3자리
     << setfill('0')                   // 빈 자리에 0을 채움
     << 3.14159 << endl;               // "0003.142"를 출력
```


파일 스트림 처리

```
#include <iostream>
#include <fstream>
using namespace std;
void main() {
   int x=1, y=2;
   char s[100];
   ofstream f1("tmp.txt");          // 저장을 위한 파일 열기
   if (f1) {                        // 정상적으로 열렸으면
      f1 << x << " " << y <<endl;   // 파일로 int 값 저장
      f1 << "Game Over !\n\n";     // 파일로 문자열 저장
   }
   f1.close();                      // 파일 닫기
   ifstream f2("tmp.txt");          // 읽기를 위한 파일 열기
   if (f2) {                        // 정상적으로 열렸으면
      f2 >> x >> y;                 // 파일에서부터 x와 y 읽기
      f2 >> s;                      // 파일에서부터 문자열 읽기
      cout << " x=" << x << " y=" << y << endl;
      cout << s << endl;
   }
   f2.close();                      // 파일 닫기
}
```

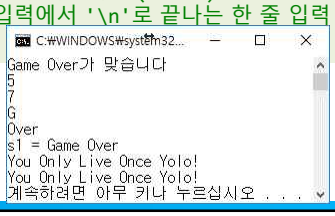
```
C:₩W...
x=1 y=2
Game
계속하려면 아무 키나 누
```

```
tmp.txt
1 2
Game Over !
```

문자열 클래스(string)

- string : 가변 길이 문자열(variable sized character array)

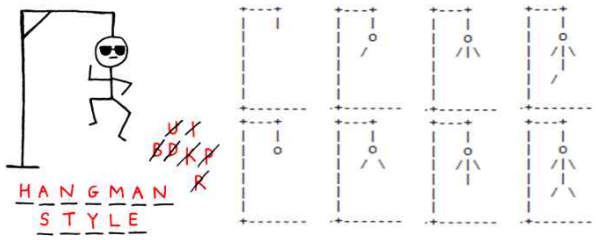
```
string s1, s2 = "Game";          // string객체 생성 및 초기화
s1.size(), s2.size();            // 문자열의 길이: 0, 5
s1 += s2 + ' ' + "Over";         // 문자열을 연결함: "Game Over"
if (s1 == "Game Over")           // 문자열의 비교 (<, >, !=, 도가능)
    cout << s1 << "가 맞습니다\n";
cout << s1.find("Over") << endl; // 문자열 검색. 위치 반환.
                                 // 없으면 string::npos(-1) 반환.
cout << s1.find("e", 4) << endl; // 4번 위치부터 "e"검색. 위치 반환.
                                 // 없으면 string::npos (-1) 반환
cout << s1[0] << endl;           // 0번째 요소 'G'
cout << s1.substr(5, 4) << endl; // s1[5]부터 길이 4의 문자열 추출
printf("s1 = %s\n", s1.c_str()); // 문자 배열 주소(char*) 추출
getline(cin, s1);                // 입력에서 '\n'로 끝나는 한 줄 입력
cout << s1 << endl;
```



6.8 응용: Hangman 게임

- Hangman 게임이란?
- 실행 결과
- 고찰

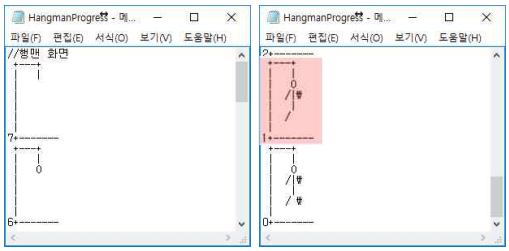
Hangman 게임이란?



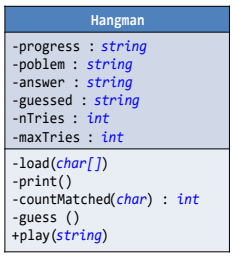
- 조건
  - string 클래스와 파일 입출력 클래스 사용
  - 각 단계별로 출력 상황은 파일에 저장
  - 알파벳 a부터 z까지를 위한 자리를 만듦
  - 이미 예측 되었으면 그 알파벳을 출력하고 아니면 '_'문자 출력
  - 게임이 종료되면 "성공" 또는 "실패"를 출력한다.

37

- "HangmanProgress.txt"파일(출력 상태 파일) 내용 예
  - 각 상태에 대한 출력 내용을 파일에 저장해서 사용함.



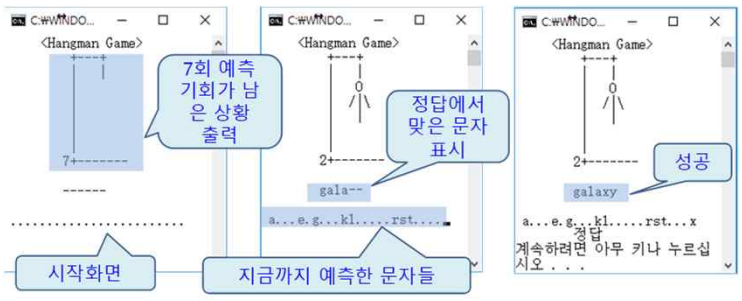
- Hangman의 UML 클래스 다이어그램



38

- 멤버 변수
  - string progress[64] : 각 단계별 화면 출력용 문자열 배열
  - string problem: 정답 단어
  - string answer: 현재 예측 중인 문자열.
  - string guessed: 전체 알파벳의 예측 상황을 나타내는 문자열
  - int nTries: 전체 실패 횟수
- 멤버 함수
  - load(): "HangmamProgress.txt"에서 상태 출력용 문자열 읽음.
  - print(): 현재의 상태를 화면에 출력함.
  - countMatched(): 예측한 문자와 정답의 일치되는 개수 반환
  - guess(): 하나의 문자를 읽어 정답 단어와 비교
  - play(): 게임의 주 함수
    - 매개변수로 정답 단어의 문자열을 받음
    - 최대 허용 횟수만큼 알파벳을 추측하고, 출력하는 등 게임을 진행함.

실행 결과



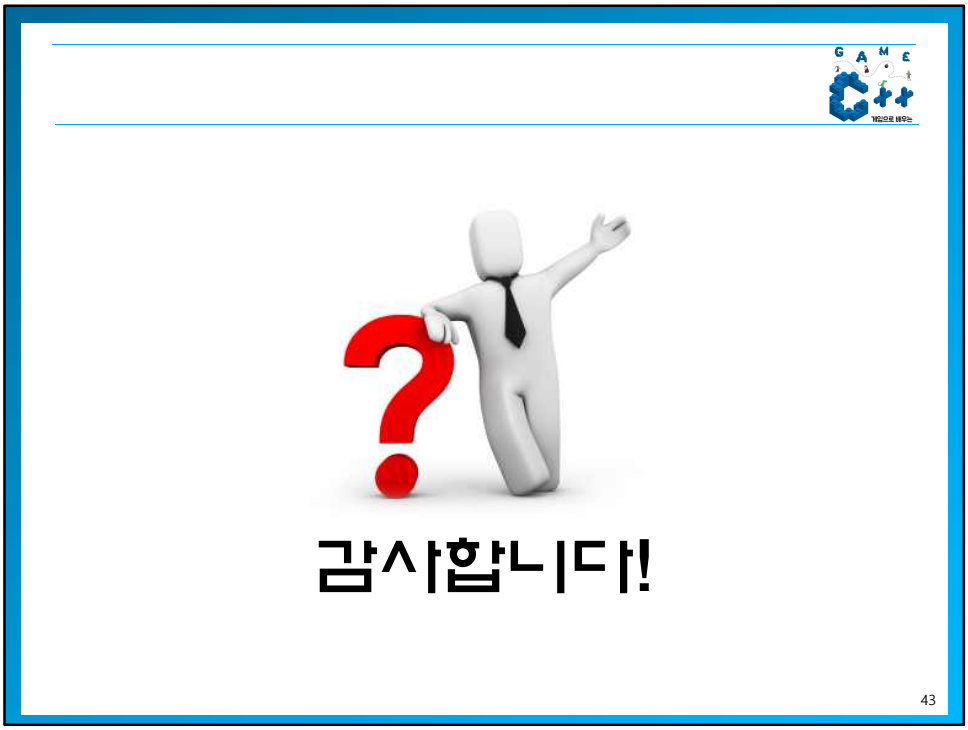
고찰

- 생성자를 이용한 객체 생성→다음 장
- string의 다양한 멤버 함수: find(), 인덱스 연산자 [] 등
- 잘 만들어진 클래스를 사용하는 것은 어렵지 않다.
- 프로그램 변형
  - 정답이 고정되지 않도록. 영어 단어도 공부하고, 게임도 하고
  - 영어로 된 격언을 맞추기. 공백 문자로 단어들을 분리 등.
  - 예측이 틀렸다고 교수대에 캐릭터를 매다는 것은 좀 심함. 다른 재미있는 그림으로 바꾸기

41

6장 요약문제, 연습문제, 실습문제

42


감사합니다!
43