7주 1강

소프트웨어 구현

이번 주차에는…

소프트웨어 구현

- 프로그램 언어의 역사
- 표준 코딩 규칙의 필요성
- 주요 표준 코딩 규칙

1. 포트란

- FORTRAN[FORmula TRANslation]
  - 사람이 이해하기 쉬운 형태로 만들어진 초창기 고급 언어
  - IBM이 우주항공 분야에서 사용하기 위해 만들었다.

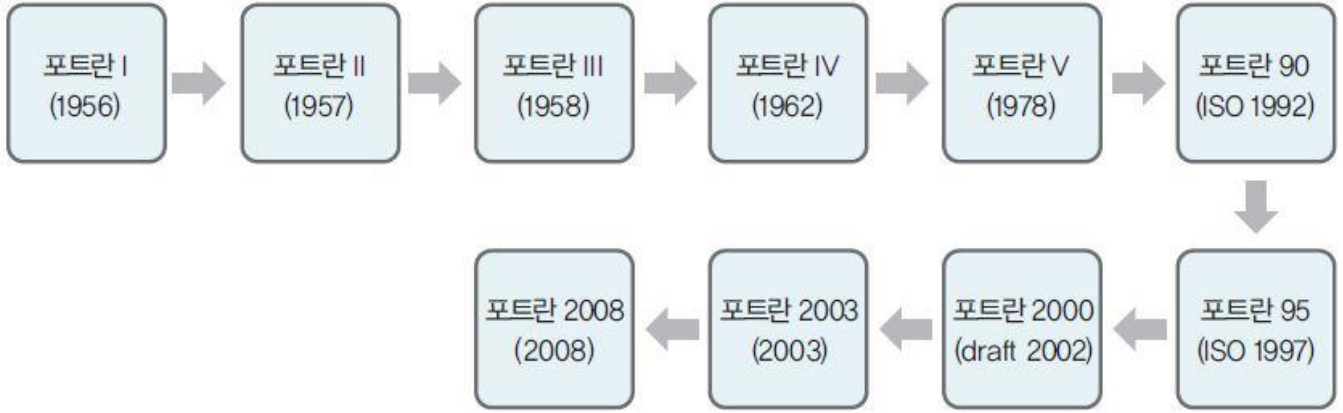

그림 7-1 포트란의 역사

2. 베이직

- BASIC Beginner's All-purpose Symbolic Instruction Code
  - 쉬운 문법을 사용하여 교육용으로 만든 언어

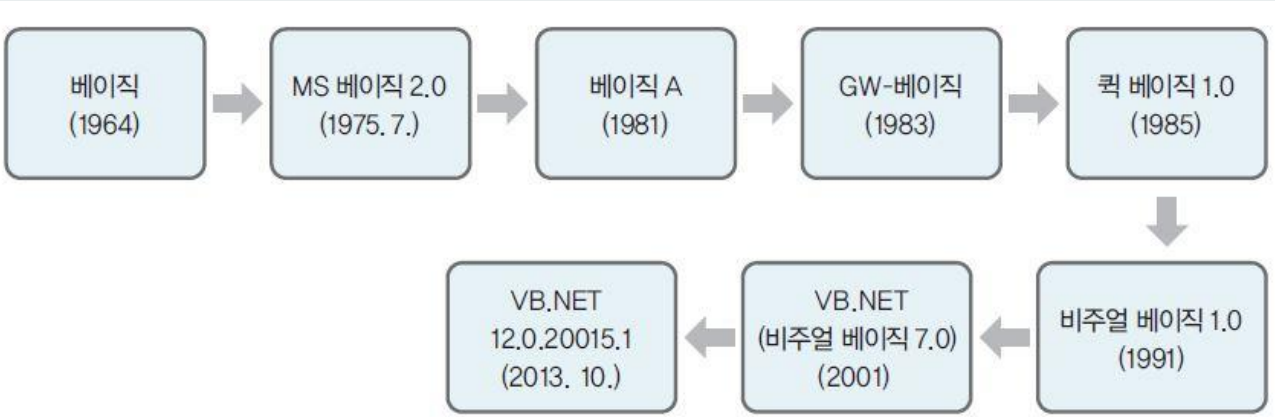

그림 7-2 베이직 언어의 역사

3. C 언어

- C
  - 현재 가장 널리 쓰이는 명령형 언어

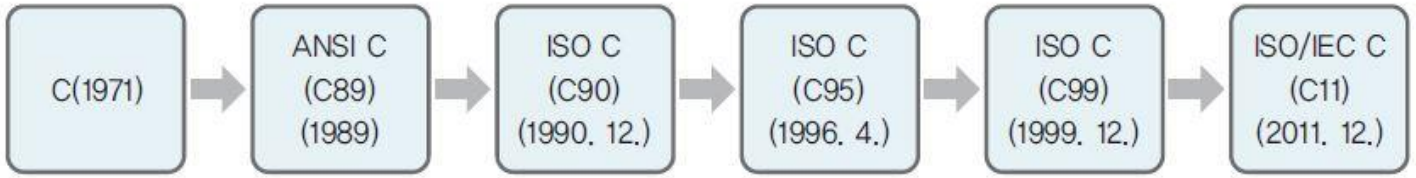

그림 7-3 C 언어의 역사

4. C++

- C++
  - C 언어에 객체지향의 개념을 더한 언어

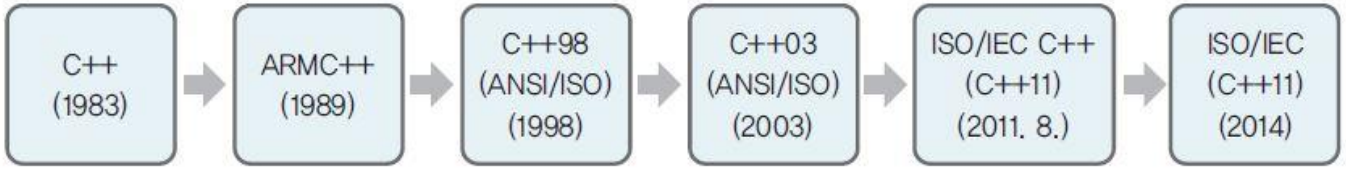

그림 7-4 C++ 언어의 역사

5. C#

- C#
  - 마이크로소프트가 닷넷 프레임워크의 일부로 만든 객체지향 프로그래밍 언어

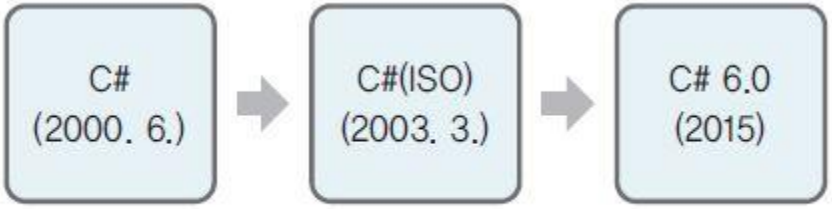

그림 7-5 C# 언어의 역사

6. 자바

- 자바
  - 선 마이크로시스템즈의 제임스 고슬링이 개발한 객체지향 언어
  - JVM(자바 가상 머신)이라는 프로그램을 사용하기 때문에 컴파일된 코드가 각각의 플랫폼에 대해 독립적

그림 7-6 자바 언어의 역사

7. 표준 코딩 규칙의 장점

- 높은 가독성
  - 가독성: '인쇄물이 얼마나 쉽게 읽히는가 하는 능률의 정도'
  - 프로그래밍에서 가독성: '프로그램을 읽기 쉬운 정도'
  - 가독성을 높이는 방법: 주석, 적절한 줄 바꿈, 들여쓰기, 적절한 공백 사용

- 간결하고 명확한 코딩
  - 읽기, 이해하기, 특정 부분을 찾아 변경하기 쉽다 → 유지보수 향상

- 개발 시간의 단축
  - 개발 과정에서 정확도 및 효율의 향상 → 개발 시간 단축

8. 표준 코딩 규칙을 따른 예와 따르지 않은 예

나쁜 예	좋은 예

int main() {
int score, char grade;
printf("test");
scanf("%d", &score);
if(num >= 90)
grade = 'A';
else if(score >= 80)
grade = 'B';
else
grade = 'F'
return 0; }
	int main() {
	  int score;
	  char grade;
	
	  printf("test");
	  scanf("%d", &score);
	
	  if(num >= 90)
	  grade = 'A';
	  else if(score >= 80)
	  grade = 'B';
	  else
	  grade = 'F'
	  return 0;
	}

9. 명칭에 관한 규칙(1)

- 명칭은 31자 이내

나쁜 예	좋은 예
int abcdefghijklmnopqrstuvwxyabcdef = 0;	int sum = 0;

- 변수명과 함수명은 다르게 사용
  - 개발자가 코딩 중에 혼란스러울 수 있어 변수명과 함수명은 별도의 이름으로 구별

나쁜 예	좋은 예

int num;
int sum = 0;
int sum( );
	int num;
	int sum = 0;
	int getSum( )

10. 명칭에 관한 규칙(2)

- 명칭의 규칙 따르기
  - 다른 프로그래밍 언어와 명칭이 호환되도록 특수 문자는 _(언더바)만 사용하고, 다음의 대소문자 사용 규칙을 따른다.
    - 매크로명 : _ 및 대문자 사용
    - 상수명 : _ 및 대문자 사용
    - 변수명 : 소문자로 시작
    - 함수명 : 소문자로 시작, 첫 번째 단어는 동사로 작성
    - 포인터명 : 참조하는 변수명의 첫 글자는 대문자 사용

나쁜 예	좋은 예

int num&sum = 0; // 허용되지 않은 특수 문자 사용
#define size 10 // 매크로명에 소문자 사용
const int hight = 20; // 상수명에 소문자 사용
int Sum = 0; // 변수명이 대문자로 시작
int Data( ) { } // 함수명이 대문자로 시작,
// 동사로 시작하지 않음
	int num_sum = 0; // 허용된 특수 문자 사용
	#define SIZE 10 // 매크로명에 대문자 사용
	const int HIGHT = 20; // 상수명에 대문자 사용
	int sum = 0; // 변수명이 소문자로 시작
	int getData( ) { } // 함수명이 소문자로 시작,
	// 동사로 시작함

11. 명칭에 관한 규칙(3)

- 포인트 변수명은 p를 붙인다

나쁜 예	좋은 예
int a = 0; int *sum += a;	int a, sum = 0; int *pSum += a;

12. 소스 형식에 관한 규칙(1)

- 소스 파일 하나는 200줄 이내로 작성
  - 200줄이상: 문맥을 파악 및 파일 관리가 어렵고, 유지보수의 효율성이 저하

- 한 줄의 길이는 800자 이내로 작성
  - 한 줄이 너무 길면 가독성이 떨어짐

- 함수의 내용은 70줄 이내로 작성
  - 함수의 크기는 한 화면에 전체 내용이 보일 수 있도록 70줄 이내로 작성
  - 함수가 너무 길어 화면이 분할되면 가독성이 떨어져 문맥을 파악하기가 어렵다.

13. 소스 형식에 관한 규칙(2)

- 여는 중괄호 {는 문장의 끝, 닫는 중괄호 }는 문장의 시작의 위치에
  - 중괄호 하나가 한 줄을 차지하면 코드가 길어져 스크롤해야 하므로 가독성이 떨어짐

나쁜 예	좋은 예

if(a > b)
{
printf("YES")
}
	if(a > b) {
	printf("YES")
	}

- 하나의 문장을 2줄로 작성하는 경우
  - 80자가 넘어 쉼표(,)가 오면 다음 문자는 새 줄로 시작
  - 둘째 줄의 시작은 다음의 좋은 예처럼 이전 줄의 표현식과 같게 한다.

나쁜 예	좋은 예
int a = k(parame ter_A, parameter_B, pa rameter_C)	int a = k(parameter_A, parameter_B, parameter_C)

14. 소스 형식에 관한 규칙(3)

- 수준이 동일한 문장은 시작 위치를 동일하게
  - 수준이 같은 문장은 동일하게 들여쓰기를 하는 것이 좋다.

나쁜 예	좋은 예

if(a > b) {
x = a + b;
return x;
}
	if(a > b) {
	x = a + b;
	return x;
	}

15. 주석에 관한 규칙(1)

- **주석의 내용**
  - **최초 작성자, 최초 작성일, 최초 변경일, 목적, 개정 이력(변경자, 변경 일자, 변경 내용) 저작권**

나쁜 예	좋은 예

// 주석이 없음
class grade{
……
}
	/*
	• 최초 작성자 : 홍길동
	• 최초 작성일 : 20○○.11.20.
	• 최초 변경일 : 20○○.11.30.
	• 목적 : 성적 프로그램
	• 개정 이력 : 홍길동, 20○○.11.20.(ver. 01)
	홍길동, 20○○.11.30.(ver. 02)
	• 저작권 : 길동company
	*/
	class grade{
	……
	}

16. 주석에 관한 규칙(2)

- 메서드 정의 앞에 다음 내용을 주석으로 추가하고 시작
  - 목적 : 함수의 용도(목적)
  - 매개변수 : 함수의 인자로 사용되는 변수에 대한 설명
  - 반환 값 : 함수의 결과 값에 대한 설명
  - 변경 이력 : 함수를 변경한 변경자, 변경 일자, 변경 내용

나쁜 예	좋은 예

// 메서드에 관한 주석이 없음
class grade{
……
}
	/*
	• 목적 : 등급 구하기
	• 매개변수 : kor(국어 점수)
	eng(영어 점수)
	mat(수학 점수)
	• 반환 값 : grade(등급-A, B, C, D, F)
	• 변경 이력 : 홍길동, 20○○. 11. 20.(ver. 01)
	홍길동, 20○○. 01. 20.(ver. 02)
	*/
	class grade{
	……
	}

17. 주석에 관한 규칙(3)

- 원시 코드와 주석을 명확히 구분
  - 방법: 원시 코드와 주석 사이 공백(또는 탭)을 두고 주석을 작성

- 원시 코드와 주석은 항상 일치하도록
  - 시간이 지나 원시 코드가 변경되어도 주석을 수정하여 일치시킨다.
  - 주석에 담을 내용:
  - 함수 인자에 대한 설명
  - 복잡한 논리식
  - 간단하지 않은 자료구조

18. 변수 선언 및 자료형에 관한 규칙(1)

- 용도가 같은 변수는 한 줄에 작성

나쁜 예	좋은 예
int a = 0; int b = 0;	int a = 0, int b = 0;

- 필요한 변수만 선언

나쁜 예	좋은 예

int a = 0; // a 변수를 사용하지 않는데 선언함	int a = 0; // a 변수 사용
	functA(a);

- 배열 선언 시 요소 수를 명시하거나 초기화

나쁜 예	좋은 예
int score[ ];	int score[5]; int score[ ] = {1, 4, 7, 9, 10};

19. 변수 선언 및 자료형에 관한 규칙(2)

- 배열을 초기화할 때는 중괄호를 적절히 사용

나쁜 예	좋은 예
int score[2][2] = {1, 4, 7, 9}	int score[2][2] = { {1, 4}, {7, 9} };

- 지역 변수는 선언 시 초기화
  - 초기화를 하지 않으면 쓰레기 값으로 초기화

나쁜 예	좋은 예
int a; // 초기화 안 됨	int a = 0; // 초기화됨

- 부호 없는 자료형은 끝에 u를 붙인다

나쁜 예	좋은 예
#define MIN 20;	#define MIN 10u;

20. 변수 선언 및 자료형에 관한 규칙(3)

- 포인터 변수에 주소나 정수 값을 저장할 때는 자료형 일치

나쁜 예	좋은 예
int *pKim; char chi = 'SU'; pKim = χ	int *pKim; int chi = '5'; pKim = χ

- 비트 필드는 unsigned/signed int형으로만 선언
  - 다른 자료형으로 선언하면 동작이 정의되지 않아 원치 않은 결과를 얻을 수 있다

나쁜 예	좋은 예

struct st {
char a : 2; // 비트 필드에 char형 사용
unsigned int b : 3;
};
	struct st {
	unsigned int a : 2; // 비트 필드에 unsigned int형 사용
	unsigned int b : 3; // 비트 필드에 unsigned int형 사용
	};

21. 상수에 관한 규칙(1)

- 상수는 10진수, 16진수로 표현
  - 상수를 표현할 때 8진수 표현은 가독성이 떨어져 10진수나 16진수 표기법 사용

나쁜 예	좋은 예
int octal = 0377; // 8진수 표현	int hexa = 0×FF; // 16진수 표현 int chi = 10; // 10진수 표현

- 숫자 리터럴은 const 변수를 사용(C 언어의 경우)
  - 원시 코드에 직접 표현(하드 코딩)한 숫자 리터럴은 그 의미를 파악하기 어렵게 한다.

나쁜 예	좋은 예

int tri_area = 10*5/2;
(삼각형 면적 = 밑변×높이/2)
	const int base_line = 10;
	const int hight = 5;
	int tri_area = base_line×hight/2

22. 상수에 관한 규칙(2)

- 상수는 부호 있는 자료형 사용, 부호 없는 자료형에는 u를 붙인다

나쁜 예	좋은 예
#define SIZE 20;	#define SIZE 20u;

23. 수식에 관한 규칙(1)

- 단항 연산자의 바른 표기
  - 단항 연산자(++, --)는 피연산자와 붙여 써야 피연산자가 어떤 것인지 금방 알 수 있다.

나쁜 예	좋은 예
a ++	a++

- 이항 연산자의 바른 표기
  - 이항 연산자(+, -, *, /)는 전후에 공백을 넣는 것이 좋다.

나쁜 예	좋은 예
a=b+c+d;	a = b + c + d;

24. 수식에 관한 규칙(2)

- 삼항 연산자의 바른 표기
  - 삼항 연산자(?:)는 알아보기 쉽게 맨 앞의 수식을 괄호(( ))로 묶어준다.

나쁜 예	좋은 예
a > b ? x: -x;	(a > b) ? x: -x;

- 증감 연산자의 바른 표기
  - 증감 연산자는 다른 연산자와 섞어 사용하지 않는 것이 좋다.

나쁜 예	좋은 예
sum = kor + (++eng);	++eng; sum = kor + eng;

25. 수식에 관한 규칙(3)

- 연산자가 3개 이상인 경우의 바른 표기
  - 연산자가 3개 이상일 경우: 우선순위를 쉽게 알 수 있도록 괄호로 묶어주는 것이 좋다.

나쁜 예	좋은 예
if(a == 0&&b == 0)	if((a == 0) && (b == 0))

- sizeof 인자의 바른 표기
  - 수식을 sizeof 함수의 인자에 사용하지 않는다.

나쁜 예	좋은 예
sizeof(a = b + c)	sizeof(a)

26. 문장에 관한 규칙(1)

- switch 문에서 case 문을 빠져나오기 위해 break 문을 사용
  - break 문이 없을 때는 다음 case 문으로 제어가 넘어간다는 사실을 주석으로 달아 준다.

나쁜 예	

case 1:
  statements
case 2:
  ~~
	case 1:
	  statements // break 문이 없어 case 2로 넘어감
	case 2:
	  ~~

- switch 문에서 case 문이 다 끝나면 default 문 사용
  - 만족하는 case 문이 없을 때 마지막 default 문에서 후속 처리를 해주기 위해서이다.

나쁜 예	좋은 예

case 1:
  statements
  break;
case 2:
  statements
  break;
	case 1:
	  statements
	  break;
	case 2:
	  statements
	  break;
	default:

27. 문장에 관한 규칙(2)

- goto 문을 사용 자제

- for 문을 제어하는 수식에 실수 값 사용 불허

나쁜 예	좋은 예
float i = 1.1 for(i = 1.1; i < 2.5; i = i + 0.1)	int i = 1 for(i = 1; i < 8; i = i++)

- for 문을 제어하는 수치 변수의 바른 사용
  - 계산에 사용하면 반복 횟수를 예측할 수 없게 된다.

나쁜 예	좋은 예

int i = 0, j = 10
for(i = 0; i < 10; i++)
  i = i + j;
	int i = 0, j = 10
	for(i = 0, j = 10; i < j; i++, j++)

28. 문장에 관한 규칙(3)

- **break 문은 가능하면 한 번만 사용**
  - 반복문에서 반복 중단을 위한 break 문을 여기저기에서 사용하면 프로그램의 동작을 예측하기가 어렵다.

나쁜 예	좋은 예

// 반복을 중단하기 위해 여러 번 break 문 사용
int i = 0;
while(1){
if(i == 0){
break;
}
else if(i == 20){
break;
}
}
	// 반복을 중단하기 위해 한 번 break 문 사용
	int i = 0;
	while(1){
	……
	if(i == 0) || (i == 20){
	break;
	}
	……
	}

29. 문장에 관한 규칙(4)

- if ~ else 문의 끝은 else 문으로 종료
  - else 문으로 종료하지 않으면, 문법적 오류는 아니지만 처리가 누락되어 원하는 결과가 나오지 않을 수 있다.

나쁜 예	좋은 예

int score;
char grade;
 	
printf("점수 입력:");
scanf("%d", &score);
if(score >= 70)
printf("PASS");
	int score;
	char grade;
	
	printf("점수 입력:");
	scanf("%d", &score);
	if(score >= 70)
	printf("PASS");
	else
	printf("failure");

다음 시간

UML 기본 개념 1