

1-2

데이터 타입으로 알 수 있는 정보

- range of values
- set of operations
- precision : 실수형

데이터 타입 목적

- 실제세계의 대상을 표현
- 공간의 효율성
- 컴파일 타임에 타입정보로 오류찾아줌

2-4. l-value, r-value

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    int x = 3;
    const int y = x;
    int z = x + y;
    int o = 8;

    cout << string("one");

    ++x;
    ++x;
}
```

l-value

object를 표기할 수 있는 것,
변수의 주소가 저장되어 있음.

r-value

object를 표기할 수 없는 것,
변수의 값이 저장되어 있음

차이점 : 공간의 위치값의 여부
= 문법적으로 object의 위치를
알아낼 수 있는가

예약어 : 컴파일러와 약속된 단어

식별자 : 사용자가 만든 단어 (1. 명어, 숫자, - 만 가능)
2. 숫자로 시작X

2-4

2-5

(1 < 2) < 3

int a = 1 < 2 < 3; 으로 계산한다

if (a = 0) => a에 0넣고 조건 if
-> 활용 : while (ch = getchar())

2-6

자동형변환

java는 작아진 만 됐는데,
C는 다됨.

타입캐스팅 연산자가
; 사칙 연산보다
; 우선순위 높다.
;

3-3

Subprogram 사용 이유

- ▶ 반복되는 코드를 한번만 작성함으로써 코드 작성 용이 (효율성)
- ▶ 프로그램 모듈화가 용이
- ▶ 큰 문제는 작은 문제로 나누어 해결하는 것이 일관적인 문제 해결 접근방법
- ▶ 작은 문제 각각은 subprogram으로 구현하고 한 프로그램을 여러 subprogram들이 집합으로 정의
- ▶ 각 subprogram은 독립적으로 작성, 수정, 테스트됨
- ▶ 개별 컴파일 (separate compilation) 가능
- ▶ subprogram들은 대개 서로 독립적이어서, 각 subprogram은 독립적으로 컴파일 가능
- ▶ 컴파일된 subprogram들은 linking을 통해서 실행가능한 한 프로그램이 됨
- ▶ 한 subprogram이 수정되면, 해당 subprogram이 다시 컴파일해야 됨
- ▶ 한 subprogram의 수정으로 프로그램 전체를 다시 compile 할 필요가 없고
- ▶ 규모가 큰 프로그램의 경우 컴파일 시간을 상당히 절약 가능



Subprogram의 특성

- ▶ 한 subprogram의 실행 시작 지점은 하나임 (one-entry)
- ▶ 한 subprogram의 실행이 종료되는 지점은 하나 이상일 수 있음
- ▶ Subprogram의 caller와 callee의 동작 과정
 - ▶ Caller subprogram이 호출한 도구는 Call(호출된 subprogram)의 실행이 완료될 때 까지 실행이 유보됨
 - ▶ Caller의 실행이 완료되면 프로그램의 실행 제어는 Caller에게로 넘어감
 - ▶ Callee subprogram을 호출한 지점의 다음부터 실행을 개시



용어, 개념 정의

Subprogram의 종류	정의(Definition)
▶ 함수(function), procedure	▶ header는 header를 포함
이름	선언(Declaration)
▶ 다른 subprogram과 구별	▶ 선언은 선언을 포함
형식인자 (formal parameter)	호출(call)
▶ header나 선언에 포함된 이름	▶ subprogram을 실행하도록 하는 일련의 표현
▶ Subprogram 내부에서 사용됨	▶ 호출의 순서는 선언의 순서가 아님
리턴 타입	실 인자(actual parameter)
▶ Subprogram이 함수(function)인 경우 리턴 타입	▶ 호출된 때 사용되는 값이나 주소
▶ Function을 실행할 때 리턴하는 값의 타입	Header
프로토타입 (prototype)	▶ Subprogram의 동작으로 사용될 인터페이스를 표현
▶ Separate 프로토타입, 단일 인터페이스, 주소 지정이 아닌 식	Body
	▶ Subprogram의 실행구문을 표현

```
int main()
{
    int foo(int, int);
    ...
    bigger = foo(a, b);
    ...
}

int foo(int a, int b)
{
    int b2;
    b2 = b;
    if (b2 < 1)
        b2 = 1;
    return b2;
}
```

Prototype

- ▶ C는 One-Pass Compiler
- ▶ 소스코드를 한 번만 보고 컴파일을 끝냄
- ▶ A가 B보다 먼저 나왔어 B가 A를 호출할 수 있음
- ▶ 함수 Prototype을 먼저 선언하면 순서가 문제되지 않음

```
#include <stdio.h>
int square(int m);
int square_add(int, int);
```

```
int square(int m);    ① 선언
int square(int m){    ② 정의
    return m*m;
}

square(num);          ③ 호출
```

3-4

static 변수

- 지역변수와 같은 접근 방식
- + 한번만 초기화되고 프로그램 종료까지 계속 메모리에 있음

3-4

실인자 : 인자

형식인자 : 매개변수

3-4

함수의 지역참조 환경

- : 일반적으로 지역변수로 구성됨. 어떤 자정에서 접근 가능한 변수, 형식인자, 함수 등을 모두 모은 것

4-1

- 스택프레임
 - 7개 변수
 - 지역 변수
 - 리턴주소
 - 리턴값
- top에 있는 스택프레임만 활성화
- 스택프레임 크기는 컴파일 타임에 결정 (매개변수, 지역변수의 개수, 타입을 고려해서 결정)
- f push $\Rightarrow$ f pop $\Rightarrow$ g push
- 하게 되면 g를 수행할 때 f의 기록이 남아있으므로 반드시 지역변수를 초기화 후 사용하자

4-3 전역변수 단점

- 프로그램 종료까지 존재 $\rightarrow$ 메모리 ↑
- 변수명 제한 ↑
- 여러 함수가 동일한 변수 사용 $\rightarrow$ 디버깅 난이도 ↑
- 외도록 지역변수를 인자로 전달하여 사용하기!

4-3

C 언어는 call-by-value 만을 지원한다.

4-3 ?

Quiz

- * 다음 중 가장 거리가 먼 것을 모두 고르시오.
- (1) 지역변수와 전역변수가 동일한 이름인 경우 지역변수를 따른다.
- (2) variable는 변수의 위치 및 접근 권한을 나타내며 접근하는 곳이다.
- (3) register는 레지스터에 변수를 할당하라는 뜻이다.
- (4) 지역변수의 lifetime은, 자신을 정의한 함수가 사라지면 종료된다.
- (5) static 변수는 scope만 전역변수와 동일하고 lifetime은 지역변수와 유사하다.

5-2 문자열 함수

strcpy (~에, ~을)

strcmp (앞, 뒤) = 앞-뒤

6-1 포인터 주소가 0번 인덱스보다 크면 포인터에서 마이너스 인덱스도 가능하다

6-3

```
int main() {
    char * s1 = "your team";
    s1 = "Our team";           // OK
    s1[2] = 'a';               // XX (불변!)
    ...
}

int main() {
    char s2[] = "its team";
    s2 = "Their team";         // XX (불변!)
    s2[2] = 'a';               // OK
}
```

- s1[2]의 'u'를 'a'로 바꿔 치기 할 수 없음
이유: 자동 할당 된 불변 문자열
- s2에 "Their team"을 지정할 수 없음
이유: 배열 이름이므로 (앞페이지!)

```
char a[] = "Hello";
char * p[] = {a, a+1, a+2, a+3, a+4};
```

6-3

```
int main() {
    printf("%s %c \n", a, *a);
    printf("%s %c \n", a+1, *(a+1));
    printf("%p %p %p %p", p, *p, *p);
    printf("%s %s %s %s\n", *p, *(p+1), *(p+2), *(p+3));
}
```

```
int a[] = {0, 1, 2, 3, 4};
```

6-2

```
int main() {
    int i, *p;
    for (i=0; i<4; i++) printf("%d \n", a[i]);
    for (p = &a[0]; p <= &a[4]; p++) printf("%d \n", *p);
    for (p = &a[0]; i=1; i<=5; i++) printf("%d \n", p[i]);

    for (p=a, i=0; p+i <= a+4; p++, i++) printf("%d \n", *(p+i));
    for (p=a+4, i=0; p>=(a+i); p--, i++) printf("%d \n", *p);
    for (p=a+4, i=0; i<=4; i++) printf("%d \n", p[-i]);
    for (p=a+4; p>=a; p--) printf("%d \n", a[p-a]);
}
```

7-1 불가능!

```
int arr[] = {0, 1}
int ptr[] = arr
```

7-2 Call-by-value 특징

- 디버깅, 재사용 Good
- swap() 같은 함수 구현 불가능

Quiz

함수 f, g가 아래와 같은 타입으로 선언되어 있을 때 다음 설명 중 다음 중 거리가 먼 것을 두 개만 고르시오.

```
void f(const char * a) {...}
void g(char * const a) {...}
```

- ① char s1[] = "hello"; 와 같이 선언된 s1는 g(s1)와 같이 호출 가능하다.
- ② char * s2 = "hello"; 와 같이 선언된 s2는 f(s2)와 같이 호출 가능하다.
- ③ 함수 f 내부에서는 a를 변경할 수 없다.
- ④ 함수 g 내부에서는 a를 변경할 수 없다.
- ⑤ 함수 f 내부에서는 *a를 변경할 수 없다.
- ⑥ 함수 g 내부에서는 *a를 변경할 수 없다.

7-3

7-3 Const with pointer

- `const int * ptr`
 - ptr이 가리키는 값의 타입이 `const int`
 - ptr의 값은 변경 가능하다 (ptr이 const가 아님)
 - ptr이 가리키는 값의 변경이 불가능하다
 - ex) `char* s = "abc"`
- `int* const ptr`
 - ptr이 가리키는 값의 타입이 `int`
 - ptr이 const 이므로 ptr의 값은 변경 불가능
 - ptr이 가리키는 값은 변경 가능
 - ex) `int arr[] = {1, 2, 3}`
`↳ arr = p 불가능`

8-1 바열의 종류

- Static 바열
 - C 전역 바열 변수
 - index 최대범위가 compile 할때 결정 됨
 - compile 할때 메모리 할당 됨
- Fixed Dynamic
 - index 최대 범위 compile 에 결정 (= Static 바열)
 - 메모리가 실행 중 함수 호출될 때 할당
 - C 지역바열 → stack에 할당, Java, C# → Heap에 할당
- (Non-fixed) Dynamic 바열
 - 최대 index 범위 실행 해보아야 함
 - Ada → stack, Java ArrayList → Heap

8-1 바열관리방법

- ▶ 컴파일하는 동안 Descriptor 를 가짐
 - ▶ 메모리 할당하는 코드 생성이 목적

원소타입	원소타입
Index 타입	Index 타입
Index 상한	차원 수
Index 하한	Index 1 상한
(시작 주소 관련 정보)	Index 1 하한
	Index 2 상한
	Index 2 하한
	(시작 주소 관련 정보)

1차원 배열의
컴파일시간 descriptor 예
(C는 파란색 부분만 지원)

다차원 배열의
컴파일 시간 descriptor 예
(C는 파란색 부분만 지원)

- ▶ 런타임에도 Descriptor 를 가지면?
 - ▶ 안전 vs. 실행속도
 - ▶ C는 원칙적으로 실행중 Descriptor 를 가지지 않음
 - ▶ Java는 실행 중 Descriptor 를 가짐
`java.lang.ArrayIndexOutOfBoundsException`