C Programming – Day 2

2018.05.03

JunGu Kang
Whols

아주대학교 Whols

함수

함수

- 큰 작업을 작은 작업들로 나눈다.
- 누군가 만들어 쓴 함수를 가져다 쓰자.
- 굳이 알 필요가 없는 것들을 숨겨준다.
  - `printf()`의 동작 원리를 굳이 알아야 할까?
- 코드의 재사용성을 높인다.
  - 수정하기에도 편리하다.

함수

- 우리는 이미 함수를 사용해봤다.
  - printf(), scanf(), …
- 이런건 함수가 아니다.
  - if, for, while, …

함수

You don’t have to reinvent the wheel,
just attach it to a new wagon.

– Mark McCormack

함수

이미 많은 함수가 제공되고 있으니 잘 써먹자.

함수

내가 원하는 함수를 만들고 싶다면?

함수의 정의

- 필요로 하는 함수를 정의해서 사용할 수 있다.
- 우리는 이미 함수를 정의해서 쓰고 있었다.
  - `main()`
  - 프로그램이 실행되면 운영체제가 가장 먼저 호출하는 함수가 `main()` 이기 때문이다.

함수의 정의

```
return-type function-name(argument declarations) {
    // declarations and statements
}
```

함수의 정의

```
return-type function-name(argument declarations) {
    // declarations and statements
    return expression;
}
```

함수의 정의

- 값을 반환하지 않는 함수 정의
  - 2_1.c

함수의 정의

- 정수 값을 반환하는 함수 정의
  - 2_2.c

함수의 정의

- 실수 값을 반환하는 함수 정의
  - 2_3.c

함수의 정의

- 함수는 사용 전에 정의되어야 한다.
  - 2_4.c

Prototype

- 함수의 Prototype을 선언하면 함수의 내용을 나중에 정의할 수 있다.
  - 2_5.c
- 함수의 Prototype에서는 인자의 이름을 생략할 수 있다.
  - 2_6.c

헤더 파일

모든 함수는 사용하기 전에 선언해야 한다.

헤더 파일

printf()와 scanf()는 선언하지 않고 사용했는데?

헤더 파일

헤더파일에 선언되어 있기 때문이다.

헤더 파일

라이브러리 함수를 사용하기 위해서는
라이브러리 함수가 선언되어 있는 헤더파일을 추가해야 한다.

헤더 파일

- `stdio.h` : standard input / output
  - 입출력과 관련된 함수들
- `string.h`
  - 문자열과 관련된 함수들
- `math.h`
  - 수학과 관련된 함수들
- `stdlib.h`
  - 메모리 관리, 랜덤 등 다양한 함수 제공
- 이 외에도 다양한 표준 라이브러리 헤더파일이 있다.

헤더 파일

- stdio.h 파일을 include 하지 않고 printf()를 사용하면?
  - 2_7.c

Static Function

- `static` 키워드가 붙은 Static Function은 다른 소스 파일에서 사용할 수 없다.

변수와 Scope

변수

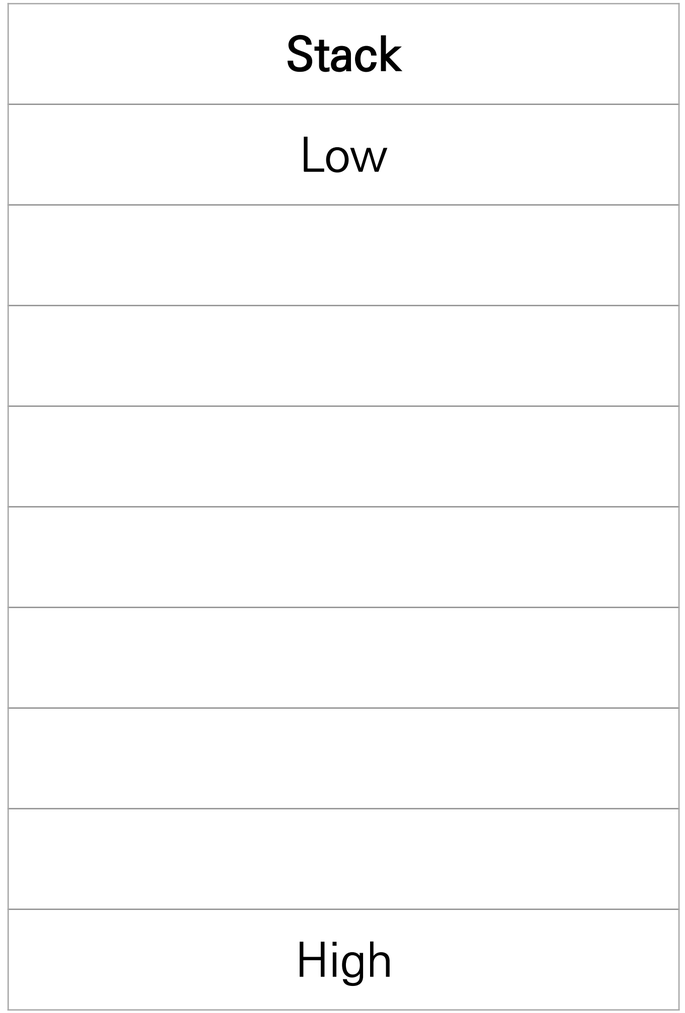
- Automatic(Local) Variable
- Global Variable
- External Variable
- Static Variable
- Register Variable
- Volatile Variable

Automatic Variable

- Automatic Variable은 함수가 호출될 때 할당되고, 함수가 종료될 때 소멸된다.

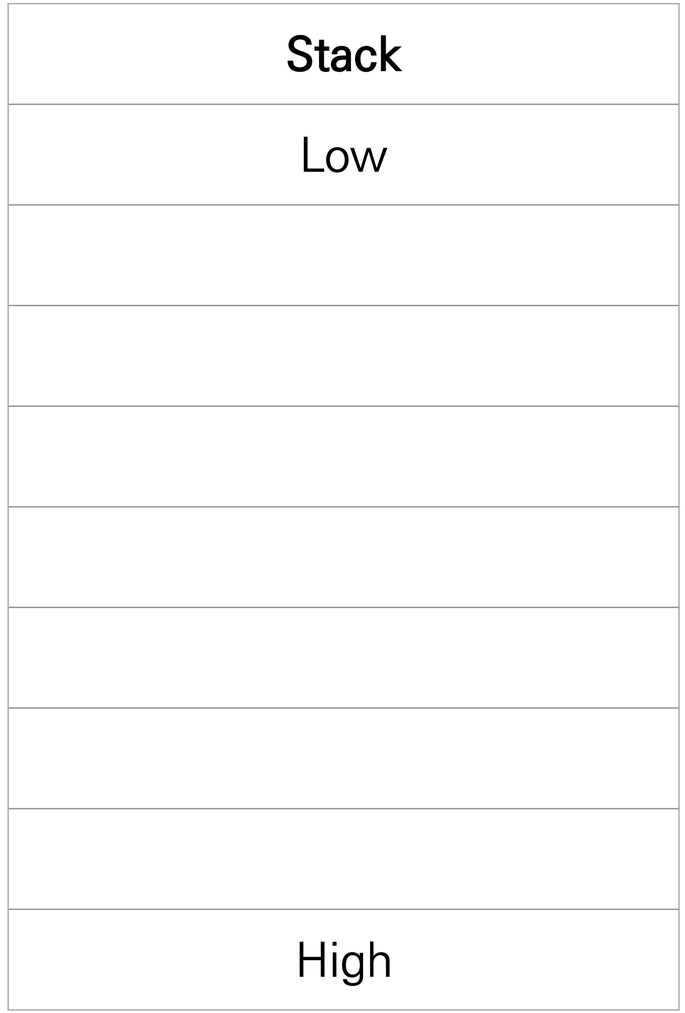
Automatic Variable

```
c() {
    int c = 15;
}

b() {
    int b = 5;
    b += 5;
    c();
}

main() {
    int a = 10;
    b();
}
```



Automatic Variable

```
c() {
    int c = 15;
}

b() {
    int b = 5;
    b += 5;
    c();
}

main() {
    int a = 10;
    b();
}
```



Automatic Variable

```
c() {
    int c = 15;
}

b() {
    int b = 5;
    b += 5;
    c();
}

main() {
    int a = 10;
    b();
}
```

Automatic Variable

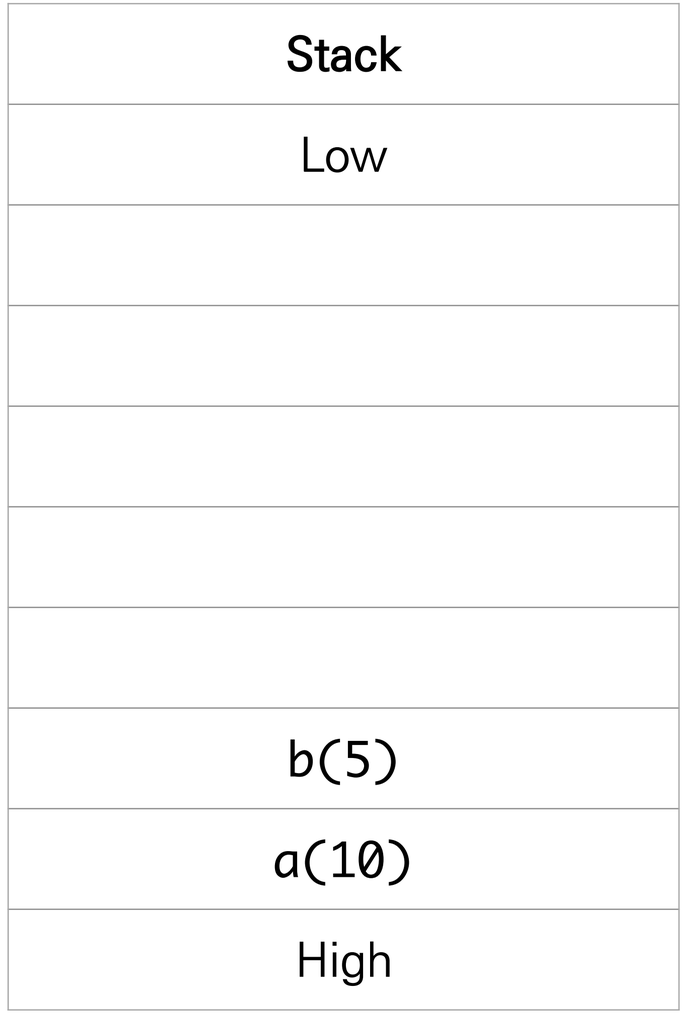
```
c() {
    int c = 15;
}

b() {
    int b = 5;
    b += 5;
    c();
}

main() {
    int a = 10;
    b();
}
```

Automatic Variable

```
c() {
    int c = 15;
}

b() {
    int b = 5;
    b += 5;
    c();
}

main() {
    int a = 10;
    b();
}
```

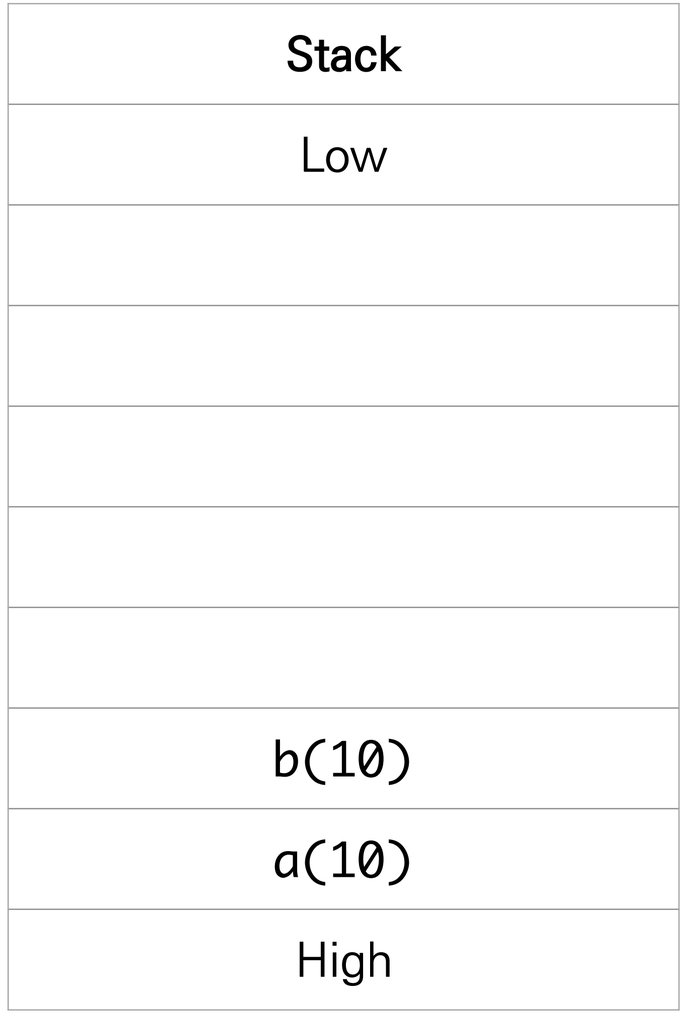
Automatic Variable

```
c() {
    int c = 15;
}

b() {
    int b = 5;
    b += 5;
    c();
}

main() {
    int a = 10;
    b();
}
```



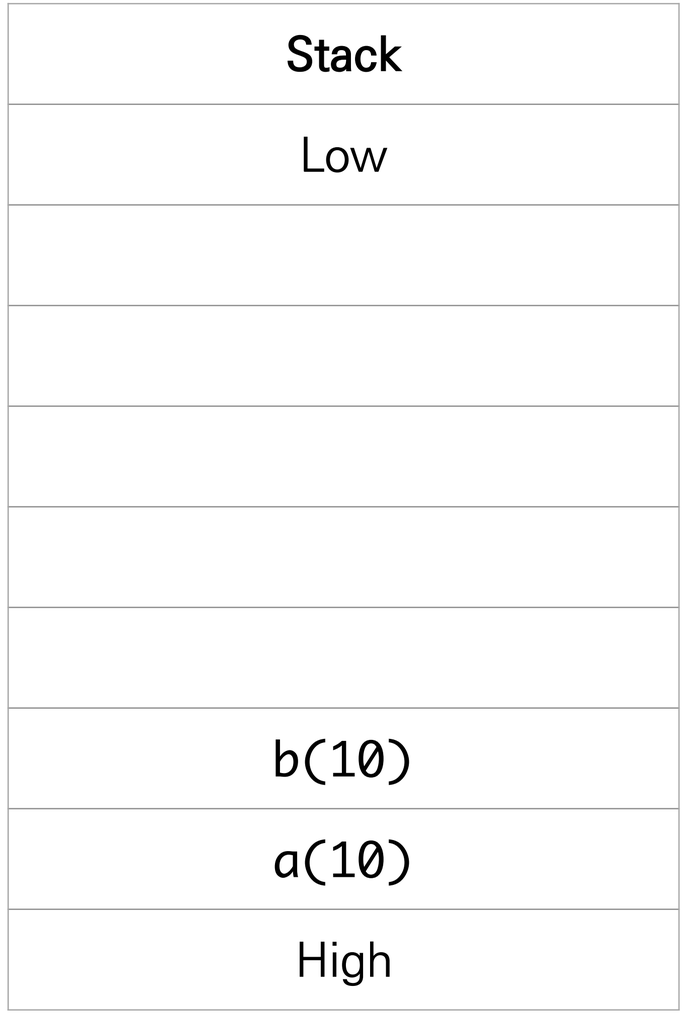
Automatic Variable

```
c() {
    int c = 15;
}

b() {
    int b = 5;
    b += 5;
    c();
}

main() {
    int a = 10;
    b();
}
```



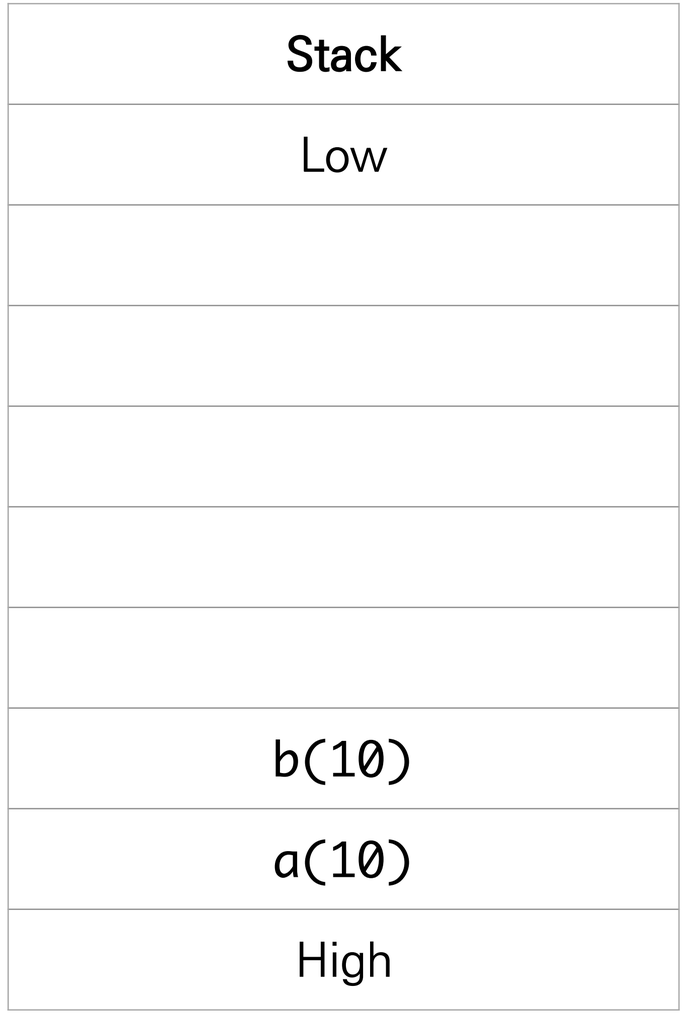
Automatic Variable

```
c() {
    int c = 15;
}

b() {
    int b = 5;
    b += 5;
    c();
}

main() {
    int a = 10;
    b();
}
```



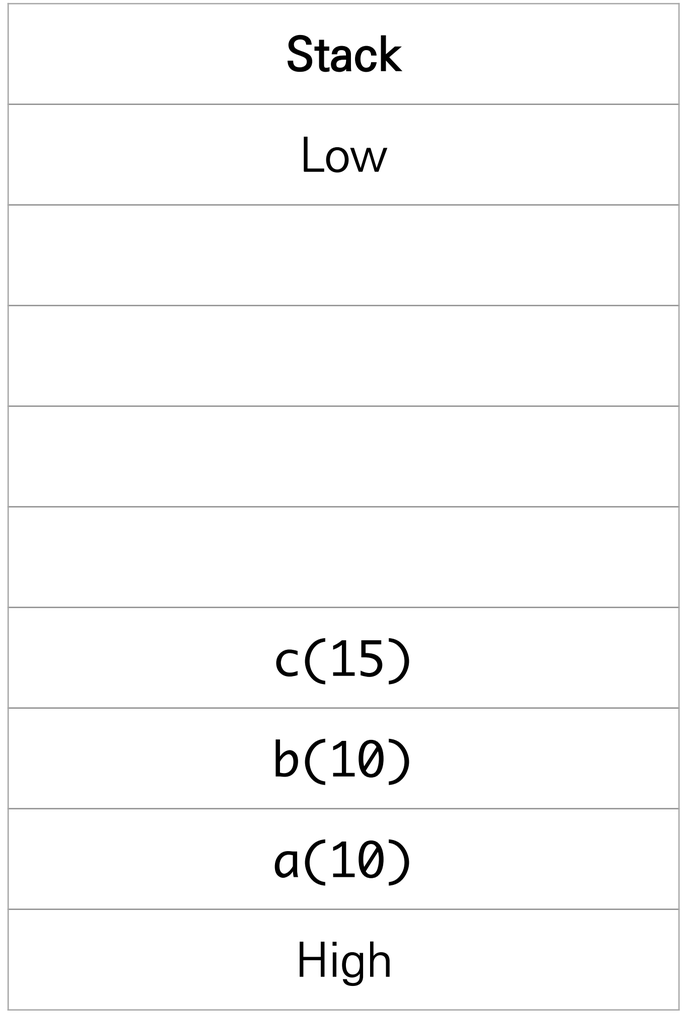
Automatic Variable

```
c() {
    int c = 15;
}

b() {
    int b = 5;
    b += 5;
    c();
}

main() {
    int a = 10;
    b();
}
```



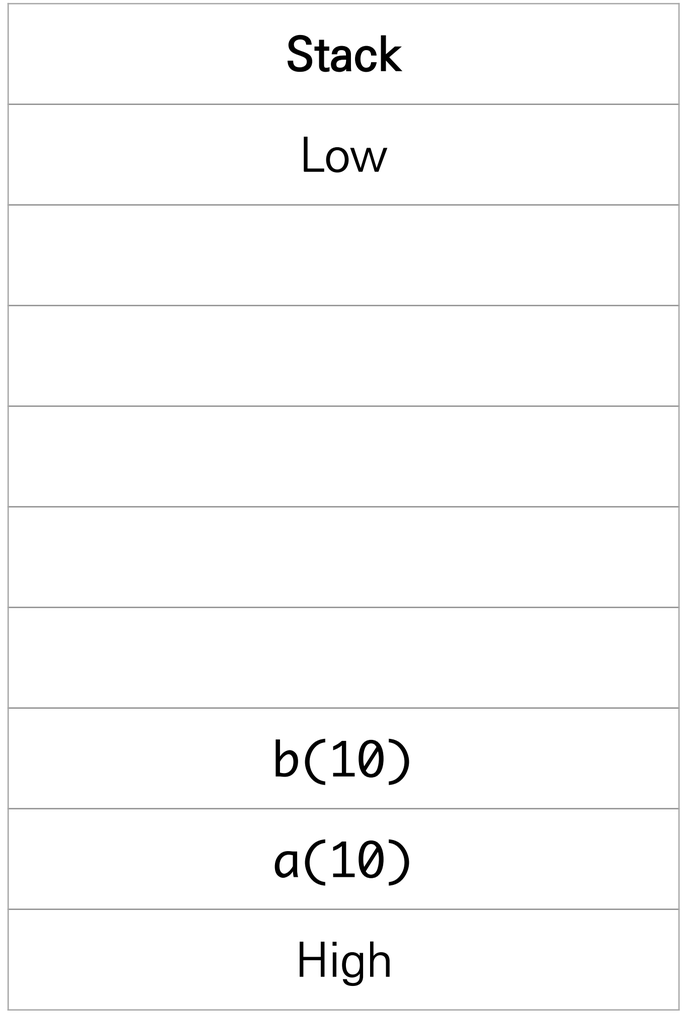
Automatic Variable

```
c() {
    int c = 15;
}

b() {
    int b = 5;
    b += 5;
    c();
}

main() {
    int a = 10;
    b();
}
```



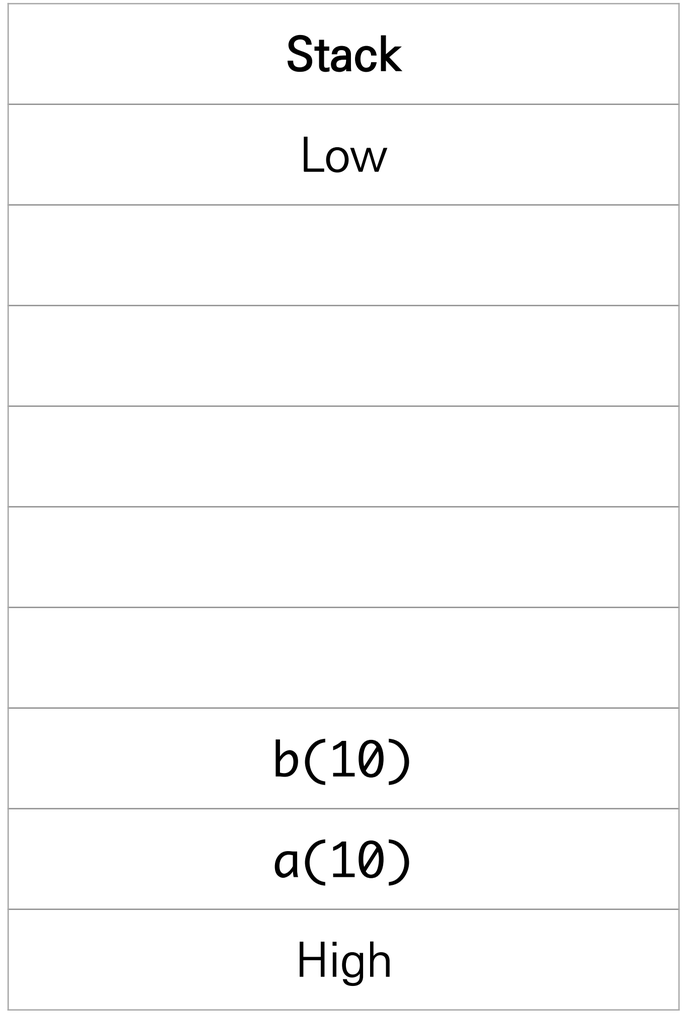
Automatic Variable

```
c() {
    int c = 15;
}

b() {
    int b = 5;
    b += 5;
    c();
}

main() {
    int a = 10;
    b();
}
```



Automatic Variable

```
c() {
    int c = 15;
}

b() {
    int b = 5;
    b += 5;
    c();
}

main() {
    int a = 10;
    b();
}
```



Automatic Variable

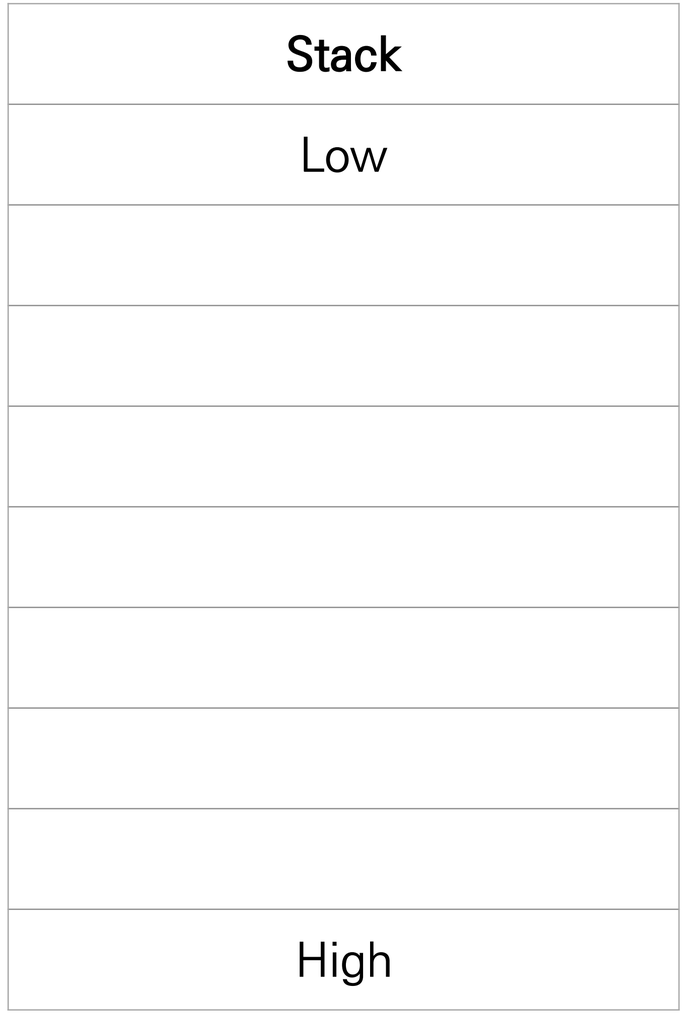
```
c() {
    int c = 15;
}

b() {
    int b = 5;
    b += 5;
    c();
}

main() {
    int a = 10;
    b();
}
```

Automatic Variable

```
c() {
    int c = 15;
}

b() {
    int b = 5;
    b += 5;
    c();
}

main() {
    int a = 10;
    b();
}
```

Automatic Variable

```
c() {
    int c = 15;
}

b() {
    int b = 5;
    b += 5;
    c();
}

main() {
    int a = 10;
    b();
}
```



Automatic Variable

- 함수 내의 Automatic Variable을 함수 밖에서 접근
  - 2_9.c

Global Variable

- 모든 함수 밖에 선언한다.
- 코드 전체에서 접근이 가능하다.
- 전역변수가 반드시 필요한 상황이 아니라면 지역변수를 사용한다.

Global Variable

```
int a = 5;

func_one() {
    a += 5;
}

func_two() {
    a += 5;
}

main() {
    func_one();
    func_two();
}
```

Data
Low
a(5)
High

Global Variable

```
int a = 5;

func_one() {
    a += 5;
}

func_two() {
    a += 5;
}

main() {
    func_one();
    func_two();
}
```

Data
Low
a(5)
High

Global Variable

```
int a = 5;

func_one() {
    a += 5;
}

func_two() {
    a += 5;
}

main() {
    func_one();
    func_two();
}
```

Data
Low
a(5)
High

Global Variable

```
int a = 5;

func_one() {
    a += 5;
}

func_two() {
    a += 5;
}

main() {
    func_one();
    func_two();
}
```

Data
Low
a(5)
High

Global Variable

```
int a = 5;

func_one() {
    a += 5;
}

func_two() {
    a += 5;
}

main() {
    func_one();
    func_two();
}
```

Data
Low
a(10)
High

Global Variable

```
int a = 5;

func_one() {
    a += 5;
}

func_two() {
    a += 5;
}

main() {
    func_one();
    func_two();
}
```

Data
Low
a(10)
High

Global Variable

```
int a = 5;

func_one() {
    a += 5;
}

func_two() {
    a += 5;
}

main() {
    func_one();
    func_two();
}
```

Data
Low
a(10)
High

Global Variable

```
int a = 5;

func_one() {
    a += 5;
}

func_two() {
    a += 5;
}

main() {
    func_one();
    func_two();
}
```

Data
Low
a(10)
High

Global Variable

```
int a = 5;

func_one() {
    a += 5;
}

func_two() {
    a += 5;
}

main() {
    func_one();
    func_two();
}
```

Data
Low
a(10)
High

Global Variable

```
int a = 5;

func_one() {
    a += 5;
}

func_two() {
    a += 5;
}

main() {
    func_one();
    func_two();
}
```

Data
Low
a(15)
High

Global Variable

```
int a = 5;

func_one() {
    a += 5;
}

func_two() {
    a += 5;
}

main() {
    func_one();
    func_two();
}
```

Data
Low
a(15)
High

Global Variable

```
int a = 5;

func_one() {
    a += 5;
}

func_two() {
    a += 5;
}

main() {
    func_one();
    func_two();
}
```

Data
Low
a(15)
High

Global Variable

```
int a = 5;

func_one() {
    a += 5;
}

func_two() {
    a += 5;
}

main() {
    func_one();
    func_two();
}
```

Data
Low
a(15)
High

External Variable

- 다른 소스코드 파일에서 선언한 변수를 이 소스코드 파일에서 사용한다.

Static Variable

- 전역변수처럼 프로그램 시작시에 할당되고, 프로그램 종료시에 소멸된다.
- 그러나 선언된 함수 내에서만 사용할 수 있다.
- 함수 밖에 선언되면 이 소스코드 내에서만 사용할 수 있다.
- 가능하면 최대한 Global Variable을 Static Variable로 대체한다.

Register Variable

- 이 변수는 레지스터에 저장하는 것이 좋다고 컴파일러에게 알려준다.
- `register` 키워드를 붙인다고 해서 항상 레지스터에 저장되지는 않는다.

Volatile Variable

- 이 변수를 최적화하지 않는다.
  - 항상 직접 값을 읽어서 확인한다.

Volatile Variable

```
main() {
    int a = 10;
    while(a == 10) {  // a를 변경하는 구문이 없으므로 이 조건은 항상 참이다.
        statements;
    }
}
```

Volatile Variable

```
main() {
     int a = 10;
     while(true) {  // 컴파일러는 이렇게 최적화를 시도한다.
          statements;
     }
}
```

Volatile Variable

- 이렇게 최적화 하는 것이 옳지만, 프로그램 외부에서 이 변수의 값을 바꿀 수 있다면 문제가 될 수 있다.

Shadowing

- Shadowing
  - 2_11.c

재귀

Recursion

함수가 자기 자신을 다시 호출한다.

Recursion

```
int function() {
    statement;
    if(expression) return;  // 반드시 탈출 조건을 주어야 한다.
    else function();
}

main() {
    function();
}
```

Shadowing

- 1부터 100까지의 합 재귀적으로 구하기
  - 2_12.c

배열

배열

- 같은 자료형을 가지는 변수들의 집합.
- 각 변수들을 원소(element)라고 한다.

배열의 선언

- ~~배열의 길이는 상수로만 선언할 수 있다.~~
- (C99 이상 표준은) 배열의 길이를 상수 뿐만 아니라 변수로 선언할 수 있다.

배열의 선언

- 배열의 선언
  - 2_13.c
- 가변 길이 배열
  - 2_14.c

배열의 선언과 초기화

- 배열의 선언과 초기화
  - 2_15.c
- 길이를 지정하지 않은 배열의 초기화
  - 2_16.c

배열 원소에 접근

- 배열 원소에 접근할때는 index를 사용한다.
- 배열의 index는 0부터 시작한다.

배열 원소에 접근

- 배열의 원소에 접근
  - 2_17.c
- 반복문을 이용한 접근
  - 2_18.c

배열 원소에 접근

- 배열의 원소를 참조할 때 index의 실제 존재 유무와 관계없이 접근이 가능하다.
  - 길이가 100인 배열에서 200번 index의 원소에 접근하거나,
  - 길이가 10인 배열에서 -10번 index의 원소에 접근할 수 있다.
  - 즉, 접근이 허가되지 않은 메모리 공간에 접근할 수 있다.
- 사용자가 입력한 index에 따라 참조할 때에는 반드시 유효한 index인지 검사해야 한다.

배열의 크기와 길이

- 배열의 크기(size)는 메모리에 할당된 공간의 크기.
- 배열의 길이(length)는 배열을 구성하는 원소의 수.

배열의 크기와 길이

- 배열의 크기와 길이 구하기
  - 2_19.c

배열과 문자열

- 앞서, 문자열은 0개 이상의 문자들로 이루어진 Sequence라고 했다.
- 즉, 문자열은 char형 변수들을 이어붙인 배열이다.

배열과 문자열

- 배열에 문자열 입력받고 출력하기
  - 2_20.c

배열과 문자열

문자열의 끝에는 반드시 NULL 문자가 붙는다.

'h'	'e'	'e'	'l'	'o'	0

배열과 문자열

그래야 이런 데이터 속에서 문자열의 끝을 찾을 수 있다.

'h'	'e'	'e'	'l'	'o'	0	76	26	47	86

다차원 배열

- 변수는 꼭 한 줄(일차원)로만 이어붙일 필요는 없다.
  - 다차원의 배열을 만들 수 있다.
- 배열의 차원에는 제한이 없지만, 4차원 이상은 쓰지 말자.
  - 머리아프다.

다차원 배열의 선언과 초기화

- 다차원 배열의 선언과 초기화
  - 2_21.c

다차원 배열 원소에 접근

- 다차원 배열의 원소에 접근
  - 2_22.c
- 반복문을 이용한 접근
  - 2_23.c

다차원 배열의 크기와 길이

- 다차원 배열의 크기와 길이 구하기
  - 2_24.c

Stack

Stack

- First In, Last Out(Last In, First Out)
- 데이터는 순서대로 쌓인다.
  - 쌓여있는 데이터의 꼭데기에 새 원소를 추가하거나,
  - 쌓여있는 데이터 중 가장 위에 있는 원소를 삭제할 수 있다.
- 쌓여있는 책

Stack Abstract Data Type

- top : 스택의 맨 위(다음에 원소가 들어갈 위치)를 가리킨다.
- push(value) : 스택의 맨 위에 새로운 값을 추가한다.
  - 스택이 모두 차있는지 확인한다.
  - 모두 차있지 않다면 top이 가리키는 위치에 새로운 값 value를 추가한다.
  - top을 1 증가시킨다.
- pop() : 스택의 맨 위에 있는 값을 삭제하고 반환한다.
  - 스택이 비어있는지 확인한다.
  - 스택이 비어있지 않다면 (top-1)이 가리키는 위치에 있는 값을 삭제하고 반환한다.
  - top을 1 감소시킨다.

Stack Abstract Data Type

- peek() : 스택의 맨 위 원소를 확인한다.
  - (top-1)이 가리키는 위치에 있는 값을 반환한다.
- is_full() : 스택이 모두 차있는지 확인한다.
  - top이 스택의 최대 길이(배열의 길이)와 같은지 확인한다.
  - 같다면(모두 차있다면) true, 아니라면 false를 반환한다.
- is_empty() : 스택이 비었는지 확인한다.
  - top이 0인지 확인한다.
  - 0이라면(비어있다면) true, 아니라면 false를 반환한다.

Queue

Queue

- First In, First Out(Last In, Last Out)
- 데이터는 순서대로 큐에 들어간다.
  - 데이터의 맨 뒤에 새로운 원소를 추가하거나,
  - 데이터의 맨 앞에 있는 원소를 제거할 수 있다.
- 대기열.

Queue Abstract Data Type

- front : 큐의 시작(첫 원소)을 가리킨다.
- rear : 큐의 끝(다음 원소가 들어갈 위치)을 가리킨다.
- enqueue(value) : 큐의 끝에 새로운 원소를 추가한다.
  - 큐가 모두 차있는지 확인한다.
  - 모두 차있지 않다면 rear이 가리키는 위치에 새로운 값 value를 추가한다.
  - rear을 1 증가시킨다.
- dequeue() : 큐의 맨 앞에 있는 값을 삭제하고 반환한다.
  - 큐가 비어있는지 확인한다.
  - 큐가 비어있지 않다면 front가 가리키는 위치에 있는 값을 삭제하고 반환한다.
  - front를 1 증가시킨다.

Queue Abstract Data Type

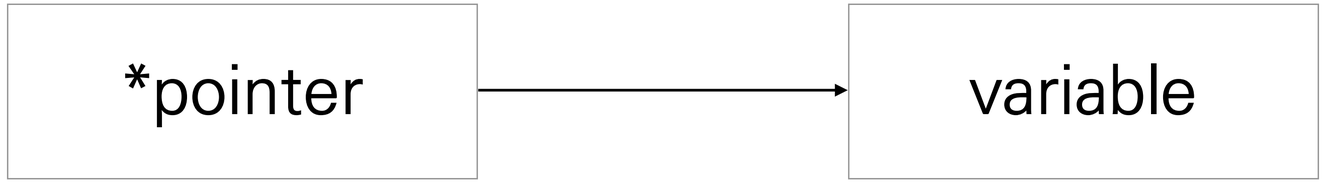
- peek() : 큐의 맨 처음 원소를 확인한다.
  - front가 가리키는 위치에 있는 값을 반환한다.
- is_full() : 큐가 모두 차있는지 확인한다.
  - rear이 배열의 길이와 같은지 확인한다.
  - 같다면(모두 차있다면) true, 아니라면 false를 반환한다.
- is_empty() : 스택이 비었는지 확인한다.
  - front와 rear이 같은지 확인한다.
  - 같다면(비어있다면) true, 아니라면 false를 반환한다.

포인터

포인터

- 변수는 메모리에 할당되고, 메모리에는 주소값이 있다.
- 포인터는 메모리의 주소를 가리키는 변수.
  - 포인터 변수의 값은 메모리의 주소값.
- 포인터 변수는 메모리의 주소를 저장하기 때문에 가리키는 대상의 자료형과 관계없이 같은 크기를 가진다.
  - 32bit 시스템에서는 32bit(4Byte), 64bit 시스템에서는 64bit(8Byte).
- 포인터의 자료형은 포인터가 가리키는 변수를 어떻게 읽고 쓸지 결정한다.

포인터



포인터 변수 선언

```
type * name;
```

포인터 변수 선언

- 포인터 선언
  - 2_25.c

포인터 변수의 초기화

- 포인터 변수는 반드시 초기화해야 한다.
  - 초기화하지 않으면 어디를 가리키고 있을지 예측할 수 없다.
- 그러나 아무 주소나 마음대로 초기화해서는 안 된다.
  - 마찬가지로 그 주소에 어떤게 있을지 예측할 수 없다.
  - 가리키는 대상으로 초기화하는게 아니라면, 반드시 NULL로 초기화한다.

포인터 변수의 초기화

- 포인터 변수의 선언과 초기화
  - 2_26.c

포인터 연산자

- * 연산자는 피연산자가 가리키는 곳으로 가서 데이터를 가져온다(역참조).
  - 절대 참조(Referencing)가 아니라 역참조(Dereferencing)이다.
- &는 피연산자가 위치한 주소를 알려준다.
- *와 &는 반대 관계이다.
  - *(&var) == var
  - &(*ptr) == ptr

포인터 연산자

- 변수 역참조
  - 2_27.c

포인터 연산

- 포인터 변수는 다른 변수 값과 연산 결과가 다르다.
- 포인터가 가리키는 자료형의 크기 단위로 증가 또는 감소한다.

포인터 연산자

- 변수 역참조
  - 2_28.c

함수로의 인자 전달

인자(Argument)

- 함수를 호출할 때 전달하는 값.
- 모든 인자는 함수 내에서만 접근 가능한 Automatic Variable이다.

Call by Value

- 함수의 인자로 값을 전달한다.
- 함수 내에서 사용하는 것은 원본 변수가 아니라 복사된 값일 뿐이다.

Call by Value

- 두 변수의 값 바꾸기(1)
  - 2_29.c

Call by Reference

- 함수의 인자로 변수의 주소값(포인터)를 전달한다.
- 함수 내부에서 원본 변수에 직접 접근할 수 있도록 한다.

Call by Reference

- 두 변수의 값 바꾸기(2)
  - 2_30.c

C Programming – Day 2

2018.05.03

JunGu Kang
Whols

아주대학교 Whols