문법  
**1 파일의 구성**zlang의 코드가 담긴 파일은 크게 2가지로 구분할 수 있다. 헤더 파일과 소스 파일로 구분할 수 있으며, 파일의 구분은 확장자로 구분한다.  
헤더 파일은 \*.h나 \*.zh의 확장자를 사용한다. 소스 파일의 경우 \*.z나 \*.zz, \*.zs의 확장자를 사용한다.  
헤더 파일은 소스 파일에서 포함시켜 사용한다. 헤더 파일은 직접적으로 컴파일에 참여하지는 않고, 소스 파일에 포함되어 소스 파일이 컴파일 될 때 소스 파일의 일부가 되어 컴파일 된다.  
**2 옵션**변수, 상수, 함수, 클래스에는 옵션을 줄 수 있다. 옵션을 주는 방법은 다음과 같다.  
<([옵션 1],[옵션 2] …) 선언>  
또한, 줄 수 있는 옵션의 종류는 다음과 같다.  
- extern  
- static  
- virtual  
- abstract  
- const  
- final  
**2.1 extern**extern은 zlang 처리용 프로그램에 등록된 변수, 상수, 함수, 클래스를 사용하기 위해 사용한다.  
**2.2 static**클래스 내부에 있는 함수, 변수, 상수에만 사용할 수 있다. 클래스의 인스턴스를 생성하지 않고 접근할 수 있도록 해준다. virtual, abstract와 같이 사용할 수 없다. 자세한 것은 함수 문단에서 다룬다.  
**2.3 virtual**클래스 내부에 있는 함수에만 사용할 수 있다. 함수를 가상 함수로 만든다. 자세한 건 함수 문단에서 다룬다. static, abstract와 같이 사용할 수 없다.  
**2.4 abstract**클래스 내부에 있는 함수에만 사용할 수 있다. 함수를 추상 함수로 만든다. 자세한 건 함수 문단에서 다룬다. static, virtual과 같이 사용할 수 없다.  
**2.5 const**클래스 내부에 있는 함수에만 사용할 수 있다. 함수를 읽기 전용 함수로 만든다. 읽기 전용 함수는 필드를 읽기만 할 수 있고 수정할 수는 없다. 대신 상수로 메모리에 저장된 인스턴스에서도 호출할 수 있다.  
**2.6 final**클래스 내부에 있는 가상/추상 함수나 클래스에만 사용할 수 있다. 클래스에 사용하면 더 이상 상속 하지 못하도록 하고, 함수에 사용하면 더 이상 재 정의 하지 못하도록 한다.  
**3 변수**변수는 변할 수 있는 수를 줄인 말이다. 언제는 변수에 저장된 값을 바꿀 수 있다. 변수는 선언을 한 뒤 사용한다. 변수의 선언은 다음과 같이 한다.  
[var 이름 타입]  
타입에 대해서는 다음 문단에서 다루도록 하겠다. 변수를 선언과 동시에 초기화 하고자 하면 다음과 같이 한다.  
[var 이름 타입]=초기 값  
**3.1 타입**타입은 Type을 그대로 발음한 것이며, 자료 형 정도로 번역할 수 있다. 타입은 변수나 상수 등에 어떤 값을 저장할지 나타낸다. 타입이 없다면 메모리에 쓰여진 값을 어떻게 해석할지 알 수 없어 0과 양의 정수만을 저장할 수 있을 것이다.  
타입은 zlang에서 기본적으로 제공하는 타입과, 사용자가 직접 정의한 클래스, 표준 라이브러리에 있는 클래스 등으로 다양하다.  
**3.1.1 기본 타입**  
zlang에서 기본적으로 제공하는 타입을 기본 타입이라 한다. 기본 타입의 종류는 다음과 같다.  
- bool 1바이트 논리형이다.  
- int8 1바이트 부호 있는 정수이다.  
- uint8 1바이트 부호 없는 정수이다.  
- int16 2바이트 부호 있는 정수이다.  
- uint16 2바이트 부호 없는 정수이다.  
- int32 4바이트 부호 있는 정수이다.  
- uint32 4바이트 부호 없는 정수이다.  
- int64 8바이트 부호 있는 정수이다.  
- uint64 8바이트 부호 없는 정수이다.  
- real32 4바이트 단정도 부동 소수점이다.  
- real64 8바이트 배정도 부동 소수점이다.  
- char 2바이트 UTF-16 문자이다.  
**3.2 상수**상수는 말 그대로 변할 수 없는 수를 말한다. 상수의 사용법은 변수의 사용법과 동일하다. 하지만 상수는 선언과 동시에 값을 설정해 줘야하며, 한번 설정한 값은 절대로 바꿀 수 없다는 점이 변수와 다르다. 상수의 선언은 다음과 같다.  
[var 이름 (const 타입)]=값  
**4 함수**zlang에서의 함수는 수학에서의 함수와 비슷하면서도 다르다. zlang에서의 함수는 명령어들을 수행하는 집합을 말한다. 함수는 선언을 한 뒤 나중에 정의하는 방법과, 바로 정의를 하는 방법 2가지가 있다.  
함수를 선언하는 방법은 다음과 같다.  
[func 이름 반환값 매개변수]  
만약 반환 값이나 매개변수가 없다면, 반환 값 또는 매개 변수 칸에 void를 적으면 된다. 매개 변수를 적는 방법은 다음과 같다.  
([var 이름 타입],[var 이름 타입] …)  
함수를 정의하는 방법은 다음과 같다.  
[func 이름 반환값 매개변수 (/\* 코드 \*/)]  
반환 값이 void가 아니면 값을 반드시 반환 해야 한다. 값은  
ret 반환 값  
위와 같이 사용하며, 값을 반환하고 함수를 즉시 종료한다. 반환 값이 void일 경우에는  
ret  
위와 같이 사용하며, 즉시 함수를 종료한다.  
함수를 호출하는 방법은 다음과 같다.  
이름(인수)  
만약 인수가 없다면 인수 부분을 void로 설정하거나 비워두면 된다.  
**4.1 함수 오버로딩**함수의 경우 이름이 같아도 매개 변수가 다르면 같은 이름을 사용할 수 있는데, 이것을 함수 오버 로딩이라 한다.  
**4.2 가상 함수와 추상 함수**가상 함수와 추상 함수는 자식 클래스에서 재정의할 수 있으며, 이를 오버 라이딩이라 한다. 가상 함수는 부모 클래스에서 반드시 정의 해야 하고, 자식 클래스에서의 오버라이딩은 선택이지만 추상 함수는 부모 클래스에서 정의 할 수 없고, 자식 클래스에서 오버라이딩을 강제로 해야한다. 추상 함수를 포함한 클래스는 추상 클래스가 된다. 추상 클래스는 클래스 문단에서 자세히 다룬다. 추상 함수가 있는 클래스를 상속 받고서 추상 함수를 정의하지 않으면 상속을 받은 클래스도 추상 클래스가 된다.  
**4.3 정적 함수**정적 함수는 클래스의 인스턴스를 생성하지 않고 사용하는 함수이다.  
클래스 이름.함수 이름(인수)  
처럼 바로 사용한다.  
**5 클래스**클래스 역시 함수처럼 선언을 한 뒤 나중에 정의하는 방법과, 바로 정의 하는 방법이 있다.  
클래스의 선언은 다음과 같다.  
[class 이름]  
클래스의 정의는 다음과 같다.  
[class 이름 상속 (/\* 코드 \*/)]  
상속에 대해서는 다 다음 문단에서 다룰 것이다.  
클래스 내부에 있는 변수, 함수에는 접근 한정자를 지정해 주어야 한다. 접근 한정자를 생략하면 자동으로 private로 처리 된다.  
[접근한정자 (/\* 코드 \*/)]처럼 사용하며 접근 한정자 안에는 접근 한정자를 넣을 수 없다. 접근 한정자의 종류는 다음과 같다.  
- public 모든 곳에서 접근할 수 있다.  
- protected 현재 클래스와 자식 클래스에서 접근할 수 있다.  
- private 현재 클래스에서만 접근할 수 있다.  
**5.1 메서드**클래스 내부에 있는 함수를 메서드 또는 멤버 함수라 부른다. 메서드의 선언 방법 및 바로 정의하는 방법은 일반 함수와 같다. 다만 메서드를 선언한 후 나중에 정의 할 때 정의하는 방법이 조금 다르다.  
[func 클래스이름.메서드이름 매개변수 (/\* 코드 \*/)]  
이와 같이 클래스 이름을 명시한다. 전역 함수 등과 충돌하지 않게 하기 위해서이다.  
**5.2 상속**상속은 부모 클래스를 지정한 뒤, 지정한 부모 클래스의 메서드와 필드(클래스 내부에 있는 변수. 멤버 변수라고도 한다.)를 그대로 가져온다.  
상속을 하지 않으려면 상속 부분을 void로 설정하면 된다. 상속을 받는 방법은 다음과 같다. 반드시 클래스의 정의 부분에만 적어야 한다.  
[class 이름 ([접근한정자 이름],[접근한정자 이름] …)]  
접근 한정자가 public이면 있는 그대로 상속을 받으며, 부모 클래스와 업 캐스팅/다운 캐스팅을 할 수 있다. protected이면 부모 클래스의 public 메서드, 필드를 protected로 바꾸어 상속 받으며, private는 부모 클래스의 public, protected인 메서드, 필드를 private로 바꾸어 상속 받는다.  
만약 부모 클래스가 추상 클래스인데, 클래스에서 추상 함수를 정의하지 않으면 그대로 추상 클래스가 된다. 추상 클래스에 대해서는 다음 문단에서 다룰 것이다.  
**5.3 추상 클래스**만약 클래스 내부에 추상 함수가 있다면 추상 클래스가 된다. 추상 클래스는 인스턴스를 생성할 수 없게 된다.  
**6 네임스페이스**네임스페이스는 전역 네임스페이스에 클래스, 함수 등이 집중되어 이름 충돌이 쉽게 일어날 수 있는 상황을 최대한 막기 위해 나온 개념이다. 네임스페이스는 다음과 같이 사용한다.  
[namespace 이름 (/\* 코드 \*/)]  
만약 전역 네임스페이스에 클래스 등을 넣고 싶다면 namespace를 생략하면 된다.  
**7 열거형**열거형은 여러가지 멤버 중에서 하나를 고를 수 있는 타입이다.  
열거형 역시 클래스처럼 선언을 한 후 나중에 정의 하는 방법과 바로 정의해서 사용하는 방법이 있다. 열거형은 반드시 헤더 파일에 정의되어야 한다. 열거형은 다음과 같이 선언한다.  
[enum 이름]  
열거형을 정의하는 방법은 다음과 같다.  
[enum 이름 (/\* 멤버 \*/)]  
멤버의 구분은 콤마(,)로 하며, 멤버 이름 뒤에 = 를 붙이고 숫자를 적어줄 수 있다. 숫자를 적어주면 해당 멤버는 값을 저장 할 때 지정한 숫자로 메모리에 저장된다. 만약 생략하게 될 경우 0부터 자동으로 값을 부여하게 된다. 만약 멤버 위에 있는 멤버들 중 값을 수동으로 설정한 멤버가 있다면 그 멤버를 기준점으로, 아래로 갈수록 1씩 더해가며 값을 부여한다.  
**8 예약어**예약어는 키워드라고도 한다. 예약어는 네임스페이스, 타입, 변수, 상수의 이름으로 사용할 수 없다. 예약어의 종류는 다음과 같다.  
- var  
- func  
- class  
- namespace  
- enum  
- const  
- 접근 한정자  
- null  
- 옵션  
- operator  
- if  
- for  
- do  
- while  
- type  
- 기본 타입  
- true  
- false  
- void  
- break  
- continue  
- ret  
- sizeof  
**8.1 null**포인터에 사용할 수 있다. 보통 메모리 주소 0을 가리킨다.  
**8.2 true와 false**bool에 사용할 수 있다. True는 참, false는 거짓이다.  
**8.3 sizeof**컴파일 시간에 그 것의 크기를 구한다.  
**9 주석**주석은 컴파일 할 때 무시한다. 주석은 한 줄 주석과 여러 줄 주석이 있다. 한 줄 주석은 다음과 같이 사용한다.  
// 메시지  
여러 줄 주석은 다음과 같이 사용한다.  
/\* 메시지 \*/  
**10 참조**참조는 크게 좌측 값 참조, 우측 값 참조로 나눌 수 있다. 좌측 값은 말 그대로 좌측 값을 참조하며, 우측 값은 우측 값을 참조한다. 참조는 선언과 동시에 초기화 해야 한다. 참조는 변수의 별명이라고 보면 된다. 참조가 가리키는 변수는 절대 변경할 수 없다. 좌측 값 참조는 다음과 같이 사용할 수 있다.  
[var 이름 타입&]=가리킬 변수  
우측 값 참조는 다음과 같이 사용할 수 있다.  
[var 이름 타입&&]=가리킬 변수  
**11 포인터**포인터는 어떤 메모리 주소를 가리키는 변수이다. zframework 에서 구동 시 플랫폼에 상관 없이 항상 8바이트이다. 기계어로 컴파일 하게 될 경우 플랫폼에 따라 크기가 달라진다. 보통 64비트 OS의 경우 8바이트, 32비트 OS의 경우 4바이트이다. 플랫폼에 상관 없이 무조건 부호가 없다. 포인터는 다음과 같이 사용할 수 있다.  
[var 이름 타입\*]  
주소 참조 연산자를 이용해 변수나 상수, 함수의 주소를 저장할 수 있고, 값 참조 연산자를 이용해 포인터 변수에 저장된 값을 가져올 수도 있다. 포인터 접근 연산자를 이용해 더 간편하게 메서드나 필드에 접근할 수도 있다. 타입 뒤에 \*를 n개 붙이면 n차원 포인터라고 하며, n은 자연수이며 무한하게 붙일 수 있다.  
**12 조건문  
12.1 if문**런타임에 평가한다.  
[if (식) (/\* 코드 \*/)]  
만약 아닐 경우 처리가 필요하다면 else를 사용한다.  
[if (식) (/\* 코드 \*/) else (/\* 코드 \*/)]  
아닐 경우 식의 평가가 필요하다면 elif를 사용한다.  
[if (식) (/\* 코드 \*/) elif (식) (/\* 코드 \*/) else (/\* 코드 \*/)]  
if는 꼭 포함되어야 하며, elif는 여러 개 추가할 수 있고, else는 단 한 개여야만 하며 맨 마지막에 위치해야 한다.  
**12.2 삼항 연산자**필요에 따라 컴파일 타임에 평가할 수도 있고 런타임에 평가할 수도 있다.  
[(식) (/\* true일 경우 \*/) (/\* false일 경우 \*/]  
true일 경우, false일 경우는 생략할 수 있으며 생략을 원한다면 void를 적으면 된다. 단, true일 경우 또는 false일 경우 둘 중 하나는 반드시 필요하다.  
**13 반복문  
13.1 for문**[for 초기값 조건문 증감형태 (/\* 코드 \*/)]의 형태로 사용한다.  
초기값, 조건문, 증감형태는 생략할 수 있고, 생략을 원하면 void를 적으면 된다.  
for는 조건문이 true이면 반복하고, 조건문 평가 -> 초기값 설정 -> 코드 수행-> 증감 -> 조건문 평가 -> 코드 수행 -> 증감 -> 조건문 평가 -> 코드 수행 … 이런 식으로 평가한다.  
**13.2 while문**[while 조건문 (/\* 코드 \*/)]의 형태로 사용한다. 조건문이 true이면 반복한다.  
**13.3 do-while문**[do (/\* 코드 \*/) [while 조건문]]의 형태로 사용한다. 한번은 무조건 수행하고, 그 후부터는 조건문이 true이면 반복한다.  
**13.4 break, continue**break은 현재 반복중인 반복문을 즉시 탈출하는 예약어이고, continue는 현재 수행중인 반복문을 건너뛰는 예약어이다.  
**14 연산자**연산자는 특정 동작을 수행하는 미리 정해진 함수이다. 연산자의 종류는 다음과 같다.  
- = 대입  
- +,-,\*,/,% 사칙 연산  
- +=,-=,\*=,/=,%= 사칙 연산 후 대입  
- &,|,^,<<,>>,~ 각각 And, Or, Xor, 왼쪽 시프트 연산, 오른쪽 시프트 연산, 1의 보수 연산 (비트 연산)  
- &=,|=,^=,<<=,>>= 각각 And, Or, Xor, 왼쪽 시프트 연산, 오른쪽 시프트 연산 후 대입  
- - 부호 뒤집기 연산  
- \* 값 참조 연산 (포인터)  
- & 주소 참조 연산  
- -> 포인터 접근 연산  
- &&,|| 각각 논리 And, 논리 Or (논리 연산)  
- ==,!=,>,>=,<,<= 비교 연산  
대입, 사칙 연산, 비트 연산, 사칙 연산 후 대입, 비트 연산 후 대입, 논리 연산, 비교 연산은 A 연산자 B의 형태로 사용한다.  
부호 뒤집게 연산, 값 참조 연산은 연산자 이름 (예: &a)의 형태로 사용한다.  
포인터 접근 연산은 A->B의 형태로 사용한다.  
**14.1 연산자 오버로딩**논리 연산, 주소 참조 연산을 제외한 모든 연산자는 연산자 오버로딩 이라는 것을 할 수 있다. 연산자 오버로딩은 프로그래머가 직접 연산자가 어떻게 동작을 수행할지 정하는 것이다. 일반적으로는 클래스 내부에 메서드 형태로 구현하는 것이 정석이지만, 사칙 연산, 비트 연산, 부호 뒤집기, 비교 연산은 클래스 밖에서도 구현할 수 있다.  
함수의 형태로 구현하며, 이름을 operator오버로딩할 연산자 이런 식으로 정한다.  
또한 명시적 형 변환도 오버로딩 할 수 있다. 이름을 operator변환될 타입 이런 식으로 정하면 된다.  
**15 템플릿**템플릿은 컴파일 시간에 처리된다. 제너릭처럼 사용할 수도 있고, 컴파일 시간에 계산할 수도 있다. 템플릿은 다음과 같이 사용한다.  
{(템플릿 매개 변수) 선언 또는 정의}  
템플릿 매개 변수는 1개라도 꼭 있어야 한다.  
[이름 타입],[이름 타입] …  
이와 같이 매개 변수를 적는다.  
타입에 들어갈 수 있는 것 중 하나인 type에 대해 다뤄보겠다. 템플릿 매개 변수에 type을 넣으면 임의의 어떤 타입을 매개 변수로 받는다. 템플릿 스코프 내에서는 해당 type을 정한 이름으로 사용할 수 있다.  
**15.1 비 타입 템플릿 인자**템플릿 매개 변수에는 type 외에도 여러가지를 넣을 수 있다. 기본 타입이라면 모두 넣을 수 있다.  
**15.2 가변 매개 변수 템플릿**템플릿 매개 변수가 가변일 경우 가변 매개 변수 템플릿이라 한다. 맨 마지막에 있는 템플릿 매개 변수는 가변 매개 변수를 사용할 수 있다.  
[이름 타입…]  
이와 같이 하면 가변 매개 변수가 된다. 가변 매개 변수의 크기는 sizeof…으로 구할 수 있다. 0 이상의 값이 나오게 되며 sizeof…은 컴파일 타임에 평가된다.  
**16 함수 매개 변수 기본 값과 가변 매개 변수**함수의 매개 변수에는 기본 값을 줄 수 있다. 단, 맨 오른쪽부터 사용할 수 있다. 중간에 갑자기 기본 값을 주거나 할 수는 없다는 것이다.  
[var 이름 타입]=기본 값,[var 이름 타입]=기본 값 …  
이처럼 사용해 기본 값을 준다.  
이제 가변 매개 변수를 다뤄보겠다. 가변 매개 변수 템플릿처럼 맨 마지막에만 위치할 수 있다.  
[var 이름 타입…] 이처럼 사용하며 기본 값을 줄 수 없다. 그렇기 때문에 가변 매개 변수와 매개 변수 기본 값은 같이 사용할 수 없다. 배열에 값이 담기게 되며, 다양한 타입의 가변 매개 변수를 사용 하고 싶다면 가변 매개 변수 템플릿을 사용해야 한다.  
**17 전처리기**전처리기는 컴파일을 하기 전 한꺼번에 수행된다.  
**17.1 포함**#include <헤더 파일>  
또는  
#include “헤더 파일”  
와 같이 사용한다. 전자는 라이브러리 헤더 파일, 후자는 현재 파일의 위치를 기준으로 한 모든 헤더 파일을 포함하고자 할 때 사용한다.  
**17.2 정의**#define A  
#define B Replace  
A는 단순한 정의, B는 치환이다. 이름 뒤에 매개 변수를 붙일 수 있으며 타입은 생략한다. 매개 변수를 사용하려면 이름을 …으로 하면 되며 맨 오른쪽에만 사용할 수 있다. 가변 매개 변수를 가져오려면 \_\_VA\_ARGS\_\_를 사용하면 된다.  
**18 타입 캐스팅**타입을 변환 하는 것을 타입 캐스팅이라 한다. 암시적 캐스팅은 할 수 없고 반드시 명시적으로 타입을 캐스팅 해야 한다.  
[cast 타입 변환할것]  
위와 같이 사용한다. 포인터끼리는 자유롭게 변환할 수 있고 (몇 차원인지는 무시) 참조의 경우 부모/자식 관계에 있을 때 자유롭게 변환할 수 있고, 일반 변수는 연산자 오버로딩으로 구현된 경우만 가능하다.