1.关于channel，下面语法正确的是()

```
A. var ch chan int
B. ch := make(chan int)
C. <- ch
D. ch <-
```

解析: ABC A、B都是声明 channel；C 读取 channel；写 channel 是必须带上值，所以 D 错误。

2.下面这段代码输出什么？

```
A.0
B.1
C.Compilation error
```

```
type person struct {
    name string
}

func main() {
    var m map[person]int
    p := person{"mike"}
    fmt.Println(m[p])
}


//运行结果
 0
```

解析：
打印一个 map 中不存在的值时，返回元素类型的零值。这个例子中，m 的类型是 map[person]int，因为 m 中不存在 p，所以打印 int 类型的零值，即 0。

3.下面这段代码输出什么？

```
A.18
B.5
C.Compilation error
```

```
func hello(num ...int){
    num[0] = 18
}
func main(){
    i := []int{5,6,7,8}
    hello(i...)
    fmt.Println(i[0])
}
```

解析 ：操作切片是地址引用，所以函数hello操作的是传过来的地址
因为底层地址对应的数相应的更改了，所以是A 18

4.下面这段代码输出什么？

```
func main() {
    a := 5
    b := 8.1
    fmt.Println(a + b)
}
```

```
A.13.1
B.13
C.compilation error
```

解析：
C go语言是强类型转换语言，所以直接编译错误。
语法报错。

5. 下面这段代码输出什么？

```
func main() {
    a := [5]int{1, 2, 3, 4, 5}
    t := a[3:4:4]
    fmt.Println(t[0])
}
```

```
A.3
B.4
C.compilation error
```

解析： B.4 操作符 [i,j]。基于数组（切片）可以使用操作符 [i,j] 创建新的切片，
从索引 i，到索引 j 结束，截取已有数组（切片）的任意部分，返回新的切片，
新切片的值包含原数组（切片）的 i 索引的值，但是不包含 j 索引的值。i、j 都是可选的，
i 如果省略，默认是 0，j 如果省略，默认是原数组（切片）的长度。i、j 都不能超过这个长度值。
假如底层数组的大小为 k，截取之后获得的切片的长度和容量的计算方法：长度：j-i，容量：k-i。
截取操作符还可以有第三个参数，形如 [i,j,k]，第三个参数 k 用来限制新切片的容量，但不能超过原数组（切片）的底层数组大小。截取获得的切片的长度和容量分别是：j-i、k-i。
所以例子中，切片 t 为 [4]，长度和容量都是 1。

6.下面这段代码输出什么？

```
func main() {
    a := [2]int{5, 6}
    b := [3]int{5, 6}
    if a == b {
        fmt.Println("equal")
    } else {
        fmt.Println("not equal")
    }
}

 //# command-line-arguments
 //  .\main.go:8:7: invalid operation: a == b (mismatched types [2]int and
[3]int)
```

```
A. compilation error
B. equal
C. not equal
```

解析：数组长度必须是常量，且是类型的组成部分，
[2]int 和[3]int是不同类型
所以直接编译失败，确切说是语法不通过

7 关于 cap() 函数的适用类型，下面说法正确的是()

```
A. array
B. slice
C. map
D. channel
```

解析：ABD cap() 函数不适用 map，可去官网标准函数库中 builtin 包中查询cap()函数
具体支持的类型如下：
　手册具体如下：
数组：v中元素的数量，与 len(v) 相同
数组指针：*v中元素的数量，与len(v) 相同
切片：切片的容量（底层数组的长度）；若 v为nil，cap(v) 即为零
信道：按照元素的单元，相应信道缓存的容量；若v为nil，cap(v)即为零

8.下面这段代码输出什么？

```
func main() {
    var i interface{}
    if i == nil {
        fmt.Println("nil")
        return
    }
    fmt.Println("not nil")
}
```

```
A. nil
B. not nil
C. compilation error
```

解析：
A.nil
当且仅当接口的动态值和动态类型都为 nil 时，接口类型值才为 nil。
空接口(interface{})不包含任何的方法，正因为如此，所有的类型都实现了空接口，因此空接口可以存储任意类型的数值。
它有点类似于C语言的void *类型。

9. 下面这段代码输出什么？

```
func main() {
    s := make(map[string]int)
    delete(s, "h")
    fmt.Println(s["h"])
}
```

```
A. runtime panic
B. 0
C. compilation error
```

解析： B.0 删除 map 不存在的键值对时，不会报错，相当于没有任何作用；获取不存在的键值对时，返回值类型对应的零值，所以返回 0。

10下面属于关键字的是（ ）

```
A.func
B.struct
C.class
D.defer
```

解析： A B D Go 语言有 25 个关键字，看下图：

```
break        default      func         interface    select
case         defer        go           map          struct
chan         else         goto         package      switch
const        fallthrough  if           range        type
continue     for          import       return       var
```

```
    break       default        func         interface        select
    case        defer           go            map               struct
    chan        else            goto           package          switch
    const        fallthrough     if             range            type
    continue     for             import         return           var
```

11.下面这段代码输出什么？

```
func main() {
    i := -5
    j := +5
    fmt.Printf("%+d %+d", i, j)
}
//-5 +5
```

```
A. -5 +5
B. +5 +5
C. 0  0
```

解析： A. -5 +5
//%d表示输出十进制数字，+表示输出数值的符号。这里不表示取反

12.下面这段代码输出什么？

```
type People struct{}

func (p *People) ShowA() {
    fmt.Println("showA")
    p.ShowB()
}
func (p *People) ShowB() {
    fmt.Println("showB")
}

type Teacher struct {
    People
}

func (t *Teacher) ShowB() {
    fmt.Println("teacher showB")
}

func main() {
    t := Teacher{}
    t.ShowB()
}

//teacher showB
```

解析：知识点：结构体嵌套。在嵌套结构体中，People 称为内部类型，Teacher 称为外部类型；通过嵌套，内部类型的属性、方法，可以为外部类型所有，就好像是外部类型自己的一样。此外，外部类型还可以定义自己的属性和方法，甚至可以定义与内部相同的方法，这样内部类型的方法就会被“屏蔽”。这个例子中的 ShowB() 就是同名方法。