到底什么是Object Value

object

region of data storage in the execution environment, the contents of which can represent values

value

precise meaning of the contents of an object when interpreted as having a specific type

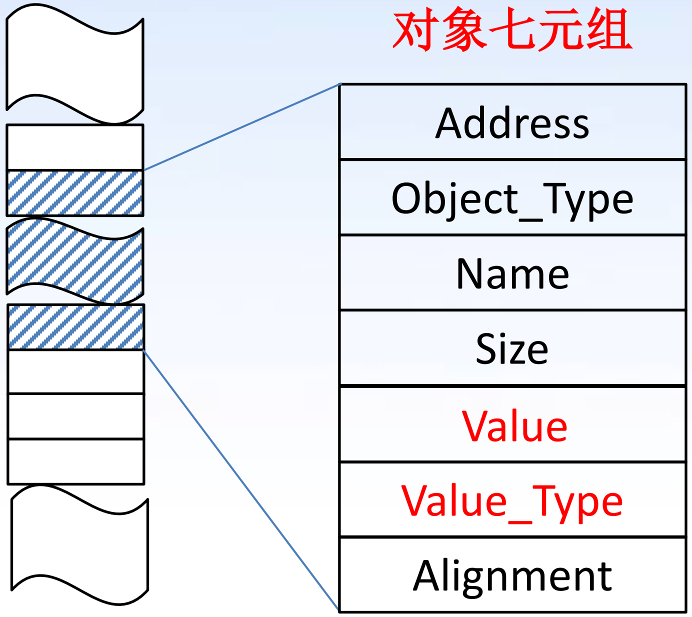
Object Representation vs. Object Value

给定同样对象类型的两个对象

1、Two values (other than NaNs) with the same object representation compare equal

2、Values that compare equal may have different object representations.

Revisit Object Value

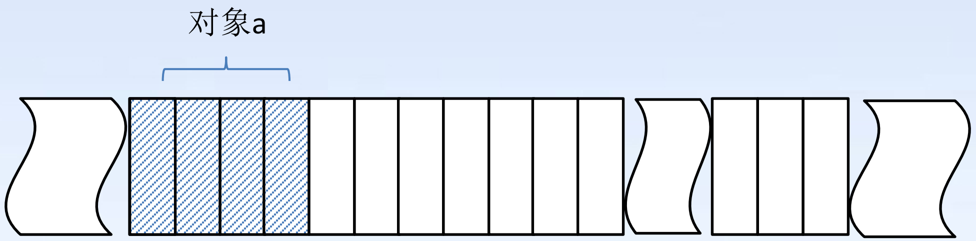


<Value, Value_Type>

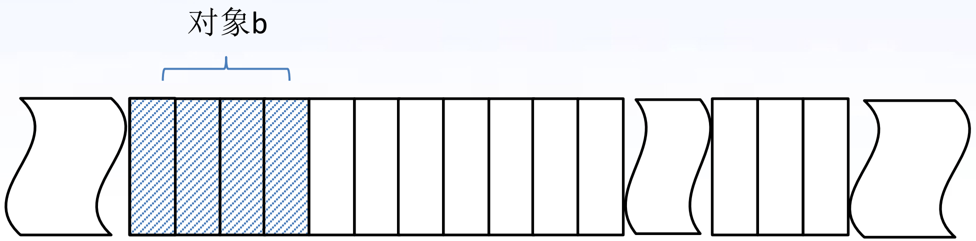
真的是Object的value么？

Object Representation vs. Object Value

```
int a = 1;
int b = 1;
```



1、对象a和b的object representation一样吗？



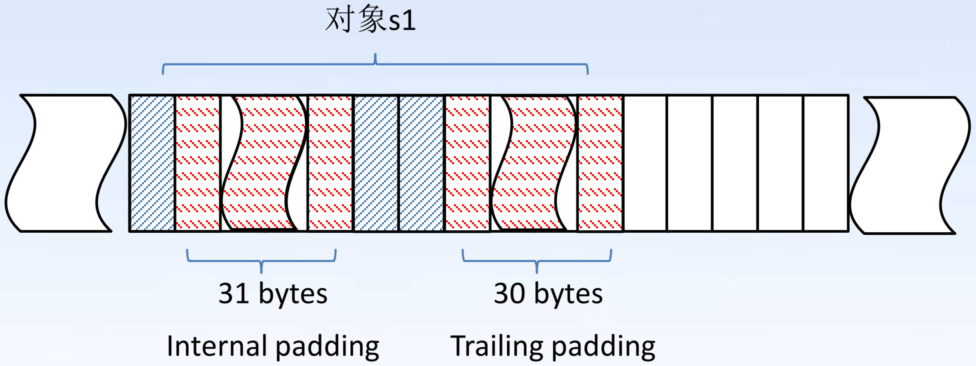
2、对象a和b的value一样吗？

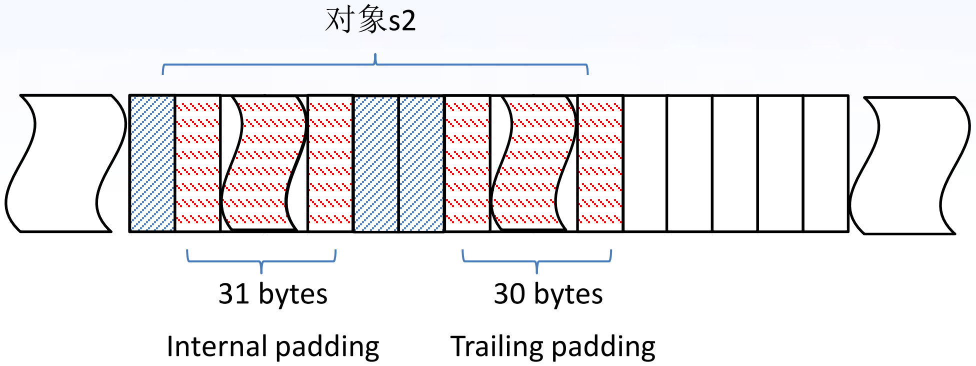
Object Representation vs. Object Value

```c
typedef struct stru {
      char a;
     _Alignas(32) short b;
} STRU;

int main(int argc, char* argv[])
{

    _Alignas(64) STRU s1 = {'a', 2};
    STRU s2 = {'a', 2};

    return 0;
}
```

1、对象s1和s2的object representation一样吗？

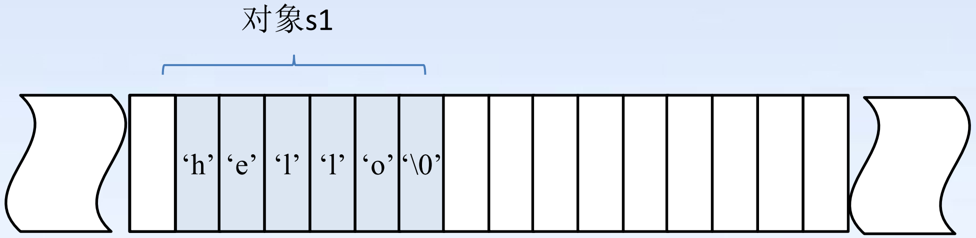
2、对象s1和s2的value一样吗？

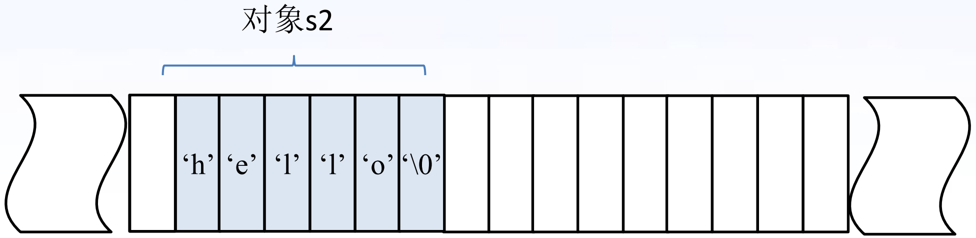




Object Representation vs. Object Value

```
char s1[6] = "hello";
char s2[6] = "hello";
```





1、对象s1和s2的object representation一样吗？

2、对象s1和s2的value一样吗？

回想一下数组对象的七元组表示

```
char s1[6] = "hello";
```

```
char s2[6] = "hello";
```

A: 0x0065FE10
Obj_T: char[6]
N: s1
S: 6
V: 0x0065FE10
V_T: char*
Align: 1

对象s1

Vs.

A: 0x0066FE10
Obj_T: char[6]
N: s2
S: 6
V: 0x0066FE10
V_T: char*
Align: 1

对象s2

七元组中两个V

是不相等的

和标准有冲突了

到底什么是对象的value呢？

T O;

对象声明的形式化方法，这时候Object Representation确定么？

未初始化有什么问题？如果要初始化怎么办呢？

T O = Initializer;

等号右边进行初始化的这个值，就是这个对象的

Object Value

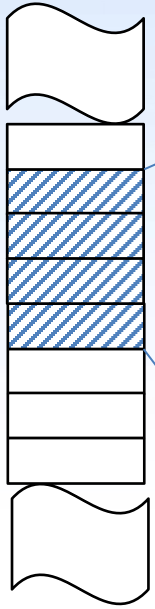
自行总结

对象o的object value可以形式化定义成<Initializer, T>

非数组对象的value呢？

int a = 1;

对象a七元组



A: 0x0061FE10
Obj_T: int
N: a
S: 4
V: 1
V_T: int
Align: 4

这个等号右边的1就是

Object Value

这个V的1，其实是能定位这个对象的lvalue进行evaluate的时候的

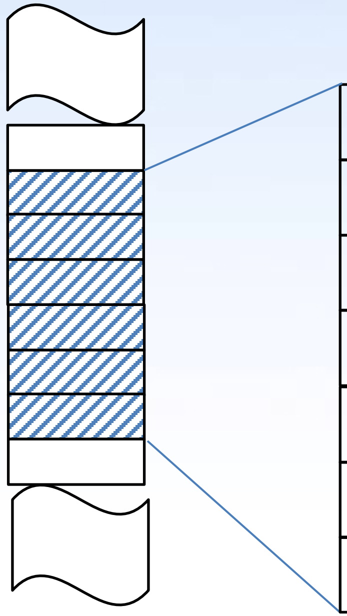
rvalue

非数组对象lvalue进行evaluate的时候，rvalue等于object value的过程称为左值转换，lvalue conversion

rvalue和object value紧密相关

数组对象的value呢？

```
char s1[6] = {'h','e','l','l','o','\0'};
```



对象s1七元组

A: 0x0065FE10
Obj_T: char[6]
N: s1
S: 6
V: 0x0065FE10
V_T: char*
Align: 1

```
{'h','e','l','l','o','\0'}
```

就是Object Value

这个V的0x0065FE10，其实是能定位这个对象的lvalue进行evaluate的时候的rvalue

数组对象lvalue进行evaluate的时候，rvalue等于首元素的地址，类型是元素类型的指针，这个过程被广泛称为decay

rvalue和object value 无关

再看七元组中的<V, V_T>

```
int a = 1;
char s1[6] = {'h','e','l','l','o','\0'};
```

1、Object Value是对象实际的值
2、只能在Object Declaration使用

七元组的<V, V_T>其实并不是对象的值，而是定位该对象lvalue进行evaluate时候的rvalue

```
a = 1;
s1 = ?
```

lvalue进行赋值的时候，等号右边表达式都是evaluate之后的rvalue，对于数组，无法赋值，这是数组对象lvalue是unmodifiable lvalue的原因

应该称为对象的表示值，即外部可以观察到的值

关于字符数组

```
char s1[6] = "hello";        char s2[6] = "hello";
```

A: 0x0065FE10
Obj_T: char[6]
N: s1
S: 6
V: 0x0065FE10
V_T: char*
Align: 1

对象s1

Vs.

A: 0x0066FE10
Obj_T: char[6]
N: s2
S: 6
V: 0x0066FE10
V_T: char*
Align: 1

对象s2

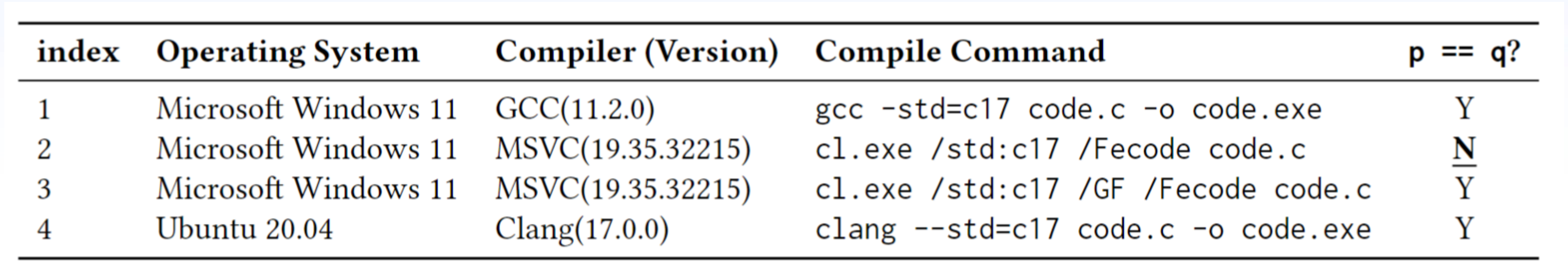
```
if(s1 == s2)
    printf("success");
```

1、s1和s2都是lvalue
2、各自需要做evaluate
3、两个rvalue不相等

s1==s2恒不为真

关于字符串

```c
char* p = "hello";
char* q = "hello";

if (p == q)
    printf("success\n");
```



index	Operating System	Compiler (Version)	Compile Command	p == q?
1	Microsoft Windows 11	GCC(11.2.0)	`gcc -std=c17 code.c -o code.exe`	Y
2	Microsoft Windows 11	MSVC(19.35.32215)	`cl.exe /std:c17 /Fecode code.c`	**N**
3	Microsoft Windows 11	MSVC(19.35.32215)	`cl.exe /std:c17 /GF /Fecode code.c`	Y
4	Ubuntu 20.04	Clang(17.0.0)	`clang --std=c17 code.c -o code.exe`	Y

字符串是不是常量

```
"hello"[0] = 'a';
```

修改字符串的内容是未定义行为

```
"hello" == "hello"
```

这个判断相等表达式不恒为真

两个"hello"表达式的rvalue可能不同

字符串不是常量、不是地址常量、不是指针、不是常量指针。。。
它是一个lvalue，按照语法做evaluate，evaluated之后rvalue类型为char*

数组名是不是常量

```
int b[10] = {0};
```

数组名不是常量、不是地址、不是指针、不是常量指针。。。
它是一个lvalue，按照语法做evaluate

evaluated之后rvalue类型为元素类型的指针类型

此外，数组名还是一个unmodifiable lvalue

什么是地址常量（Address Constant）

An address constant is

1、 a null pointer

2、 a pointer to an lvalue designating an object of static storage duration

3、 a pointer to a function designator

"hello"的rvalue不是地址常量

int b[10]; b的rvalue也不是地址常量

思考题

int b[10] = {1, 2, 3};

1、Object value确定吗？

2、对象b的object representation确定吗？

3、定位对象b的lvalue做evaluate之后rvalue是多少？

按语法补齐

1、<{1, 2, 3, 0, 0, 0, 0, 0, 0, 0}, int[10]>

2、{1, 2, 3, 0, 0, 0, 0, 0, 0, 0}十个int值依次转成32位bit顺序放入

3、<Address, int*>

思考题

给定同样对象类型的两个对象

1、Two values (other than NaNs) with the same object representation compare equal

2、Values that compare equal may have different object representations.

两个同样类型的对象的object value能进行比较么？

非数组对象类型 vs. 数组对象类型

思考题

```c
char* p = "hello\0world";
char* q = "hello\0word";

if(0 == strcmp(p, q));
    printf("success");
```

strcmp精确比较了是两个数组对象的value么？

思考题

```c
typedef struct stru {
    char a;
    _Alignas(32) short b;
} STRU;

int main(int argc, char* argv[])
{
    STRU a = {'a', 1};
    STRU b = {'a', 1};

    if(0 == memcmp(&a, &b, sizeof(STRU)))
        printf("success");

    return 0;
}
```

memcmp能精确比较两个对象的value么？

padding bits的问题

a == b能行吗？

a = b能行吗？

对象存储规则(Storage Duration)

任何一个对象都有生命周期（lifetime)，决定生命周期的就是对象的 **Storage Duration**

在生命周期内
1、系统确保对象的内存有效
2、在对象声明周期内，地址不变
3、对象保有最后赋值(last-stored value)不变

超出对象生命周期之外对对象的访问是未定义行为

对象的4种存储周期

C语言定义了4种存储周期： **static**, **thread**, **automatic**, **allocated**

Static存储周期的对象

Its lifetime is the entire execution of the program and its stored value is initialized only once, prior to program startup.

Automatic存储周期的对象

An object whose identifier is declared $\boxed{\text{with no linkage}}$ and without the $\boxed{\text{storage-class specifier}}$ static has automatic storage duration

allocated存储周期的对象

依靠malloc这些内存管理函数分配的空间，需要调用free释放

Storage-class specifiers

C语言提供了一系列Storage Specifier
来规定对象可以存在内存的什么区域

auto

constexpr

extern

register

static

thread_local

typedef

Storage-Class Specifiers的限制

一般来说，对象声明的时候，Storage-Class Specifier只能有一个，除了：

1、thread_local可以和static或extern同时出现

2、constexpr可以跟auto，register或static同时出现

Storage-Class Specifiers的作用

Storage-Class Specifier规定了标识符（Identifier）的不同属性

1、storage duration：thread_local，auto，register，以及block scope中的static

2、linkage： extern，file scope中的static和constexpr，typedef

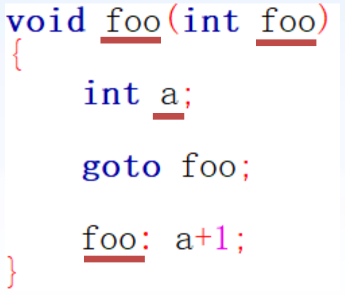
3、value： constexpr

4、type： typedef

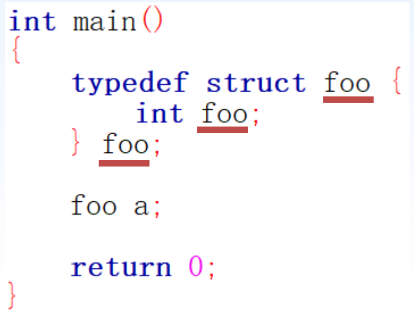
首先需要了解Identifier、Scope，Name Space, Linkage的基本概念

什么是Identifier

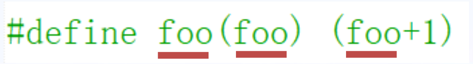
标识符是“a sequence of nondigit characters (including the underscore _, the lowercase and uppercase Latin letters, and other characters) and digits, which designates one or more entities”



```
void foo(int foo)
{
    int a;

    goto foo;

    foo: a+1;
}
```



```
int main()
{
    typedef struct foo {
        int foo;
    } foo;

    foo a;

    return 0;
}
```



```
#define foo(foo)  (foo+1)
```

Object Function Label Name Tag Member of Structure Typedef Name Macro Name Macro Parameter

C23增加了a standard attribute, an attribute prefix, or an attribute name;

什么时候Identifier能重名

```
int main()
{
    typedef struct foo {
        int foo;
    } foo;

    foo foo;

    return 0;
}
```

error: redefinition of 'foo' as different kind of symbol

为什么会编译错误？

什么时候Identifier能重名

```
typedef struct foo {
    int foo;
} foo;

int main()
{
    foo foo;

    return 0;
}
```

为什么没有编译错误了？

同一个标识符，如果指代不同的实体时

这些实体要么处于不同的Scope，或者不同的Name Space

什么是Scope？什么是Name Space？

标识符的Scope

Identifier（标识符）是C语言中最基础的一种Symbol

For each different entity that an identifier designates, the identifier is visible (i.e., can be used) only within a region of program text called its **scope**.

Scope的分类:Function Scope

C语言一共有4种Scope

function, function prototype, file, block

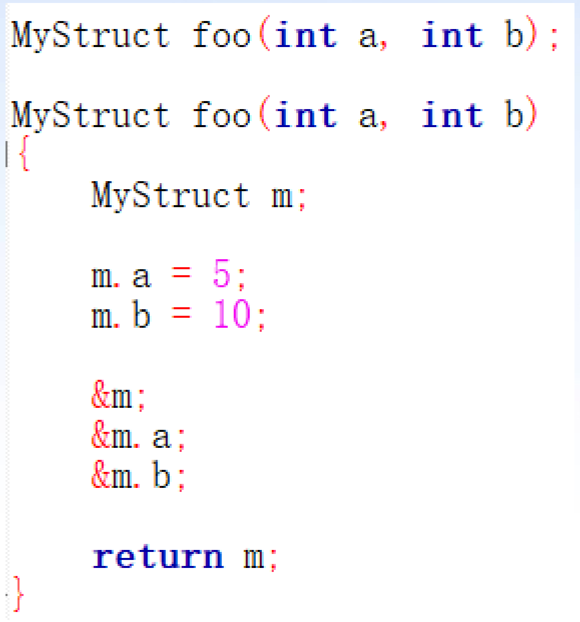
A label name is the only kind of identifier that has function scope.

```
typedef struct foo {
    int foo;
} foo;
```

```
int main(int argc, char* argv[])
{
    foo foo = {5};
    goto foo;
    foo: foo.foo++;
    printf("%d\n", foo.foo);
    return 0;
}
```

这个foo是function scope

Scope的分类：Function Prototype Scope



```
MyStruct foo(int a, int b);

MyStruct foo(int a, int b)
{
    MyStruct m;

    m.a = 5;
    m.b = 10;

    &m;
    &m.a;
    &m.b;

    return m;
}
```

the identifier appears within the list of parameter declarations in a function prototype (not part of a function definition), the identifier has function prototype scope, which terminates at the end of the function declarator.

注意区分函数原型说明和函数定义

Scope的分类：Block Scope vs. File Scope

If the declarator or type specifier that declares the identifier appears outside of any block or list of parameters, the identifier has file scope

If the declarator or type specifier that declares the identifier appears inside a block or within the list of parameter declarations in a function definition, the identifier has block scope, which terminates at the end of the associated block.

Identifier是否位于一个**block**或者作为函数实参

Name Space

在一个文件中，同样的identifier可以属于不同的name space，包括：

1、label name

2、tag

3、member of structures or unions

4、standard attributes and attribute prefixes

5、trailing identifier in an attribute prefixed token

6、ordinary identifiers

Name Space

```
                     ①
typedef struct foo {
    int foo; ②
} foo; ③

int main(int argc, char* argv[])
{         ④
    foo foo = {5};

    goto foo;
   ⑤
    foo: foo.foo++;

    printf("%d\n", foo.foo);

    return 0;
}
```

	Scope	Name space
①	File Scope	Tag
②	Block Scope	Structure Member
③	File Scope	Ordinary
④	Block Scope	Ordinary
⑤	Function Scope	Label

思考题

```
                        ①
typedef struct foo {
    int foo;
} foo;

int main(int argc, char* argv[])
{                           ②
    typedef struct foo{
        int foo;
    } foo;

    foo a = {5};

    return 0;
}
```

typedef两个结构体类型foo，为什么可以？

①和②都属于name space中的tag，
但分别属于不同的scope

思考题

①
```
typedef struct foo {
      int foo;
} foo1;
```
②
```
typedef struct foo {
      int foo;
} foo;
```

为什么不可以？

①和②都属于name space中的Tag，都属于相同的scope

```
typedef struct foo {
      int foo;
} foo; ③

typedef struct foo1 {
      int foo;
} foo; ④
```

为什么不可以？

③和④都属于name space中的Ordinary，都属于相同的scope

了解Linkage

相同的标识符出现在不同的地方，他们是否表示同一个实体呢？

C语言依靠一个叫做linkage的机制来进行分辨

An identifier declared in different scopes or in the same scope more than once can be made to refer to the same object or function by a process called linkage.

C语言有三种Linkage：internal、none、external

Internal Linkage

如果一个对象标识符拥有file scope，且被static或constexpr修饰，则该标识符internal linkage

```
static int a;

int main(int argc, char* argv[])
{
    a++;
    printf("%d", a);
    return 0;
}
```

printf出来的值一定是1吗？

All objects with static storage duration shall be initialized (set to their initial values) before program startup. The manner and timing of such initialization are otherwise unspecified

思考题

```
int main(int argc, char* argv[])
{
    static int a;

    return 0;
}
```

对象标识符a是interal linkage吗？

标识符a是什么scope？

Internal Linkage

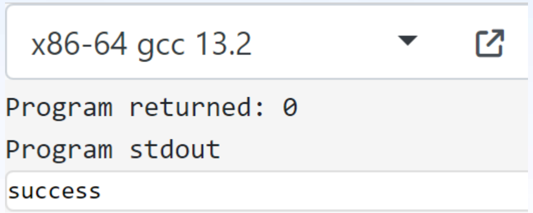
如果一个函数标识符拥有file scope， 且被static修饰，则该标识符internal linkage

```
static void func()
{
    // do nothing
}
```

函数嵌套定义

```c
int foo()
{
    printf("success");

    return 0;
}

int main(int argc, char* argv[])
{
    int foo()
    {
        printf("success");
    }

    foo(&a);

    return 0;
}
```

C语言不支持函数嵌套定义



```
Program returned: 0
Program stdout
success
```

GCC扩展支持，注意这不符合标准

Internal Linkage的特点

如果一个标识符具有Internal Linkage，则其他编译单元无法使用这个标识符，有助于程序的模块化

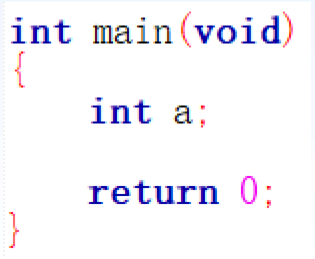
no linkage

The following identifiers have no linkage:

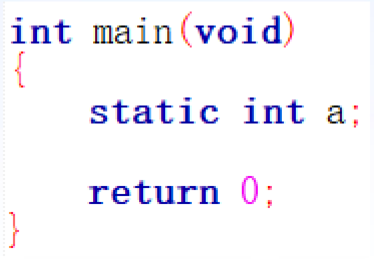
1、 an identifier declared to be anything other than an object or a function;

2、 an identifier declared to be a function parameter;

3、 a block scope identifier for an object declared without the storage-class specifier extern.



```
int main(void)
{
    int a;

    return 0;
}
```



```
int main(void)
{
    static int a;

    return 0;
}
```

no linkage的标识符都指向不同的对象

extern修饰对象标识符

```
static int a =  10;

int main()
{
    int* p = &a;

    extern int a;

    a++;

    printf("%d", *p);
    return 0;
}
```

For an identifier declared with the storage-class specifier extern in a scope in which a prior declaration of that identifier is visible, if the prior declaration specifies internal or external linkage, the linkage of the identifier at the later declaration is the same as the linkage specified at the prior declaration.

`extern int a;`

这个a具有internal linkage

extern修饰函数标识符

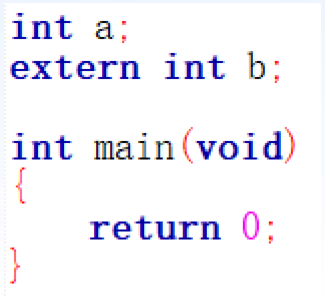
```c
int main(int argc, char* argv[])
{
    extern int foo();

    foo();

    return 0;
}

int foo()
{
    printf("success");

    return 0;
}
```

The declaration of an identifier for a function that has block scope shall have no explicit storage-class specifier other than extern.

foo是函数标识符，目前是block scope

external linkage

If no prior declaration is visible, or if the prior declaration specifies no linkage, then the identifier has external linkage.



```
int a;
extern int b;

int main(void)
{
    return 0;
}
```

思考题

```
static int a;
extern int a;

int main()
{
    return 0;
}
```

```
extern int a;
static int a;

int main()
{
    return 0;
}
```

左右两种写法，哪个对？为什么？

思考题

```
int main()
{
    extern int a;

    printf("%d", a);
    return 0;
}

int a = 30;
```

这样可以吗？

extern int a
和
int a = 30
这两个a都是external linkage，指向同一个对象

思考题

```
static int foo()
{
    printf("success");

    return 0;
}
extern int foo();

int main()
{
    return 0;
}
```

```
extern int foo();
static int foo()
{
    printf("success");

    return 0;
}
```

这样可以吗？

思考题

```c
int a = 1;
static int b = 2;
extern int c;
int d;

int a;
static int b;
int c;
int d;

int main() {

  return 0;
}
```

Vs.

```c
int main()
{
    int a = 1;
    static int b = 2;
    extern int c;
    int d;

    int a;
    static int b;
    int c;
    int d;

    return 0;
}
```

register

register T O

告诉编译器越快越好，但编译器可以不理会

register修饰的对象不能取地址

```
register int a;
&a
```

✕

register修饰数组对象呢？

```
register int a[10];

int* p = a;
int c = a[2];
```

未定义行为

对象标识符a进行evaluate之后，应该是第一个元素的首地址，但是register的对象没有地址，因此有问题

If the array object has register storage class, the behavior is undefined

constexpr

constexpr是C23新增加的storage-class specifier，

主要是为了将标识符声明成常量表达式

```
constexpr int m = 10;
int a[m];
```

对象a不再是VLA

考试要求

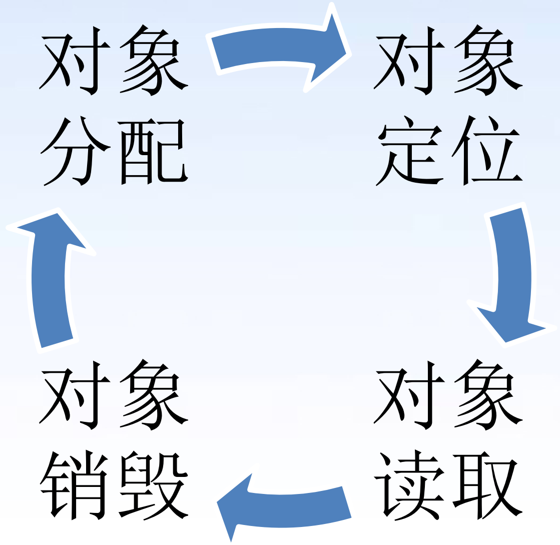
考试时间/地点：暂定

考试时长：2小时

考试形式：开卷考试

考试题型：选择、简答题、问答

简单复习一下



表达式和lvalue

evaluate和rvalue

function designator和function pointer