ADT example: string table

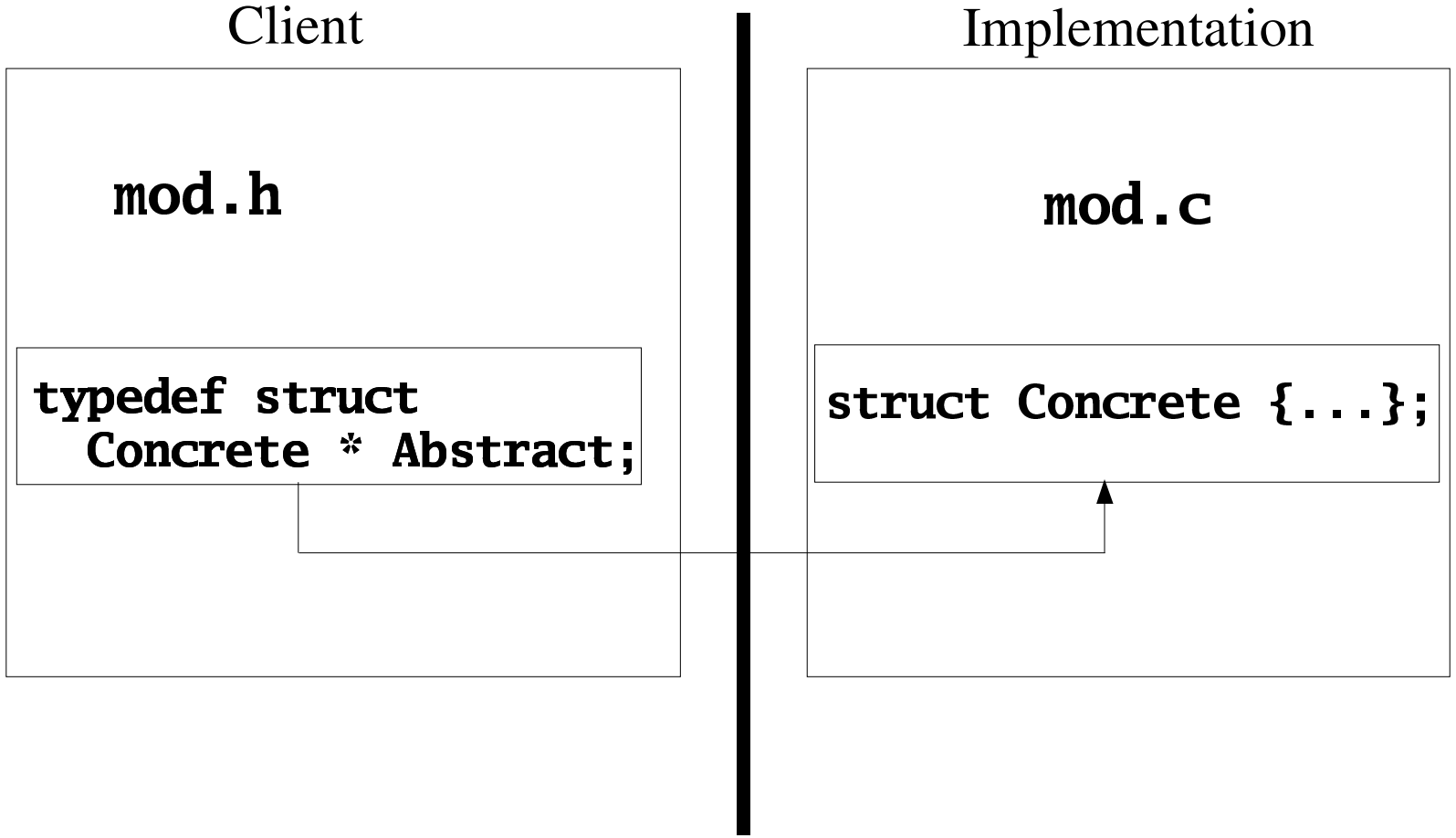
CS2023 Winter 2004

Recall: Opaque Data Type

- Incomplete structure definition:
  - can define the type of a structure that hasn't been defined yet:

```
typedef struct Concrete *Abstract;
```

  - allows one to use the type **`Abstract`** as a synonym for **`struct Concrete *`**.


Client
Implementation
mod.h
mod.c
typedef struct
Concrete * Abstract;
struct Concrete {...};

Recall: String Table Module

```
typedef struct{
  char *string;
  int count;
}StringTabT;

#define SIZE 5000

StringTabT *initTable(void);

int addStringTable(char *string, StringTabT
*table);

void printTable(StringTabT *table);

void clearTable(StringTabT **ptable);
```

String Table ADT

- Turn this module into an ADT
  - Define opaque type:

```
typedef struct StringTabCDT *StringTabT
```

  - Note that in previous module, **`StringTabT`** was a struct type. Here **`StringTabT`** is a *pointer* to struct **`StringCDT`**.

String Table ADT

- Turn this module into an ADT (cont'd)
  - Create function prototypes for interface (can now specify initial size of each table)

```
StringTabT initTable(int size);

int addStringTable(char *string,
                   StringTabT table);

void printTable(StringTabT table);

void clearTable(StringTabT *ptable);
```

String Table CDT

- Implement table as an array
  - Define type for elements of array:

```
typedef struct {
  char *string;
  int count;
} StringArrayT;
```

  - Define CDT as a struct containing pointer to element of type **StringArrayT**

```
typedef struct StringTabCDT{
  StringArrayT *sArray;
  int size;
} StringTabCDT;
```

String Table CDT as linked list

- Could also implement as linked list
  - Define type for elements of list:

```
typedef struct node{
   char *string;
   int count;
   struct node *next;
} StringNodeT;
```

  - Define CDT as a struct containing pointer to node of linked list of type **StringNodeT**

```
typedef struct StringTabCDT{
   StringNodeT *sList;
   int size;
} StringTabCDT;
```

Implementation of String Table ADT as an array

- Use previous implementation, but now referring indirectly to table through **`table->sArray`**

`StringTabT initTable(int size);`

  - Allocate memory for struct containing pointer to beginning of array: **`malloc(sizeof(StringTabCDT))`**
  - Allocate memory for array, store address in **`StringTabCDT p->sArray`**
  - Store **`size`** in **`p->size`**

```
StringTabT initTable(int size)
{
  StringTabT p; /* same as StringTabCDT *p */
  StringArrayT *s;

  if((p = malloc(sizeof(StringTabCDT))) == NULL)
    return NULL;
  p->size = size;
  if((s = calloc(size,sizeof(StringArrayT)))
        == NULL){
    free(p);
    return NULL;
  }
  p->sArray = s;
  return p;
}
```

```
Int addStringTable(char *string, StringTableT
                                    table);
```

  - To refer to array, use `table->sArray`
  - If table size exceeded, reallocate memory to grow table

- Similar modifications for `printTable()` and `clearTable()`

```
int addStringTable(char *string, StringTableT
                   table)
{
  StringArrayT *p;
  int newSize;

  for(p = table->sArray;
      p < table->sArray+table->size &&
      p->string != NULL; p++)
    if(strcmp(string,p->string) == 0){
      p->count++;
      return 1;
    }
  if(p == table->sArray+table->size)

  /* ... reallocate */
```

```
  if(p == table->sArray+table->size)
    newSize = 2*table->size;
    if((p = realloc(table->sArray,
        newSize*sizeof(StringArrayT))) == NULL)
      return -1;
    table->sArray = p;
    p = table->sArray+table->size;
    table->size = newSize;
  }
  if((p->string = strdup(string)) == NULL)
    return -1;
  p->count = 1;
  return 0;
}
```

```
void printTable(StringTabT table)
{
  StringArrayT *p;
  for(p = table->sArray;
      p < table->sArray+table->size &&
      p->string != NULL; p++)
    printf("%s: %d\n", p->string, p->count);
}
```

```
void clearTable(StringTabT *ptable)
{
  StringArrayT *p;

  for(p = (*ptable)->sArray;
      p < (*ptable)->sArray+(*ptable)->size &&
      p->string != NULL; p++)
    free(p->string);
  free((*ptable)->sArray);
  *ptable = NULL;
}
```