A Practice with Lists

CIS2520 - Data Structures

LAB 3

Tao Xu

Real-World Objects to Data Types

Abstraction

What are the relevant characteristics of these real-world objects?
How to organize the data?
What are the operations that naturally fit with the data structure?

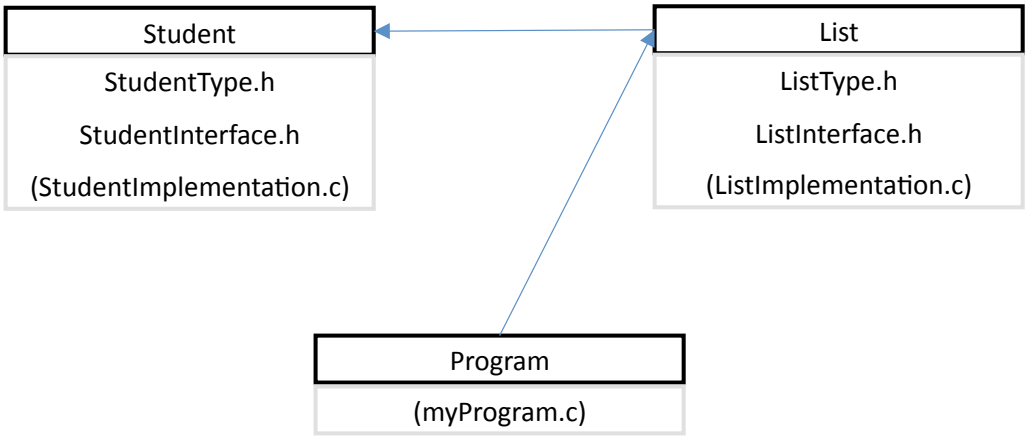
Implementation

How to store each object's characteristics in the computer's memory?
How to store the organized data in the computer's memory?
How to implement these operations?

Abstract Data Type (ADT)

- Abstraction of data types of similar behaviors
  - e.g, list, stack, queue,
- Encapsulation for unified access through interfaces

Dependencies between Modules



Student

StudentType.h

```
#define MAXNAMESIZE 20
typedef struct {
        char name[MAXNAMESIZE];  // pointer or array?
        int grade;
} Student;
```

StudentInterface.h

```
#include "StudentType.h"
extern void InitializeStudent (char *name, int grade, Student *S);
extern char *NameOfStudent (Student S);
extern int GradeOfStudent (Student S);
extern void FreeStudent (Student *S);
```

StudentImplementation.c

```
#include "StudentInterface.h"
void InitializeStudent (char *name, int grade, Student *S) {
//memory allocation for S by caller or supplier?
    …; //preconditions?
    strcpy(S->name, name); S->grade = grade;
    …; //postconditions?
}
char *NameOfStudent (Student S) {
    return S.name;  //does this work?
}
```


List (Sequential Implementation) 1/4

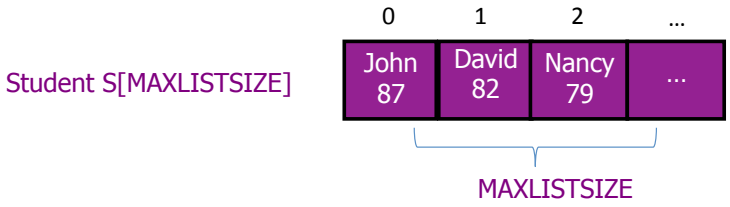
ListType.h

```
#include "StudentInterface.h"
typedef Student Item; //what for?
#define MAXLISTSIZE 4
typedef struct {
        Item items[MAXLISTSIZE];
        int count;
} List;
```

ListInterface.h

```
#include "ListType.h"
extern void Initialize (List *L); //what does it mean?
extern void Insert (Item X, int position, List *L); //how?
extern void Delete (int position, List *L); //how?
extern int Full (List *L);
extern int Empty (List *L); //Boolean or integer?
extern int Length (List *L);
extern void Peek (int position, List *L, Item *X); //who allocates
memory for X?
extern void Destroy (List *L); //what does it mean?
```

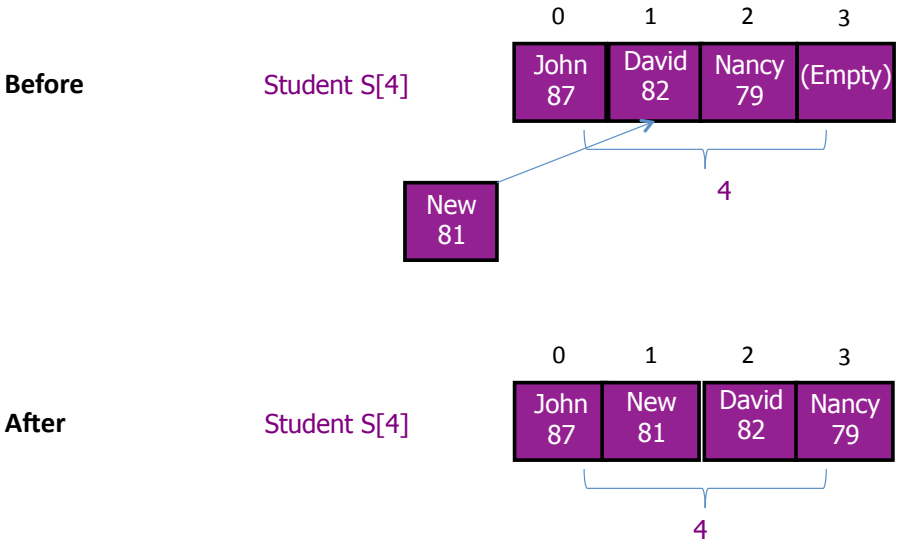
List (Sequential Implementation) 2/4



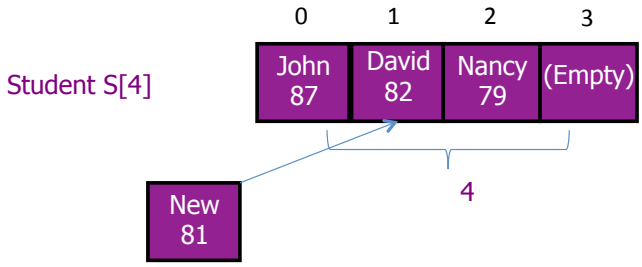
- Capacity is fixed
- Inefficient mutators
  - Insertion, e.g, at position 0 needs to push down all entries
  - Deletion, e.g, at position 0 needs to pull up all entries

List (Sequential Implementation) 3/4

```
void Insert (Item X, int position, List *L);
```



List (Sequential Implementation) 4/4




```
void Insert (Item X, int position, List *L) {
  #ifdef DEBUG
     if (Full(L)) ...; //exit? other preconditions?
  #endif
     for (int i = Length(L)-1; i>=position; i--) { //order matters
          InitializeStudent(NameOfStudent(L->items[i]), GradeOfStudent(L->items[i]),
                                &(L->items[i+1]));
     }
     InitializeStudent(NameOfStudent(X), GradeOfStudent(X), &L->items[position]);
     L->count++;
     assert(!Empty(L)) ; //do you know this macro? what it does? other postconditions?
}
```


List (Linked Implementation) 1/4

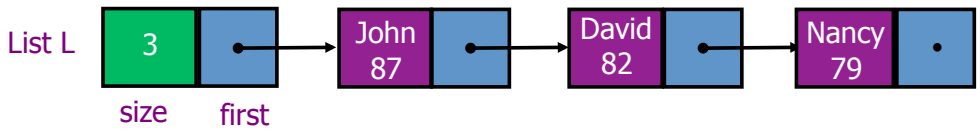
ListType.h

```
#include "StudentInterface.h"
typedef Student Item; //what for?
#define MAXLISTSIZE 4
typedef struct ListNodeTag {
          Item item;
          struct ListNodeTag *next;
} ListNode;
typedef struct {
          int size;
          ListNode *first;
} List;
```

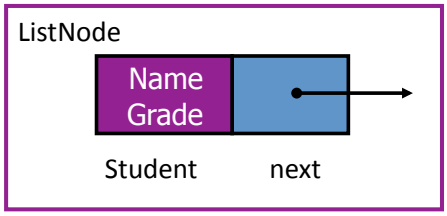
ListInterface.h

```
#include "ListType.h"
extern void Initialize (List *L);
...
//interface kept same, but implementation might differ
//where are changes required?
...
extern void Destroy (List *L);
```

List (Linked Implementation) 2/4

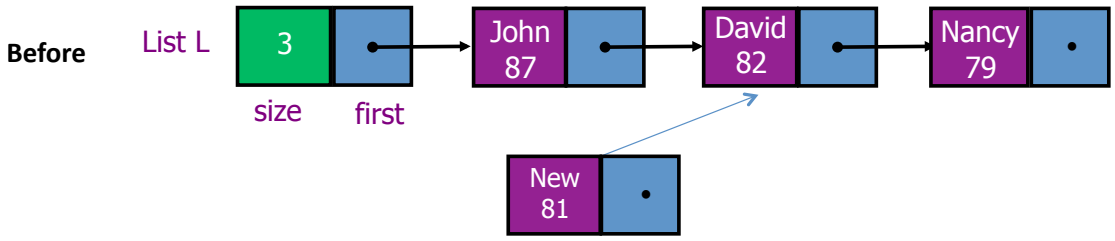


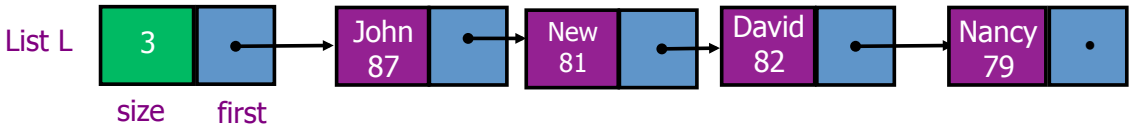
- Capacity is easy to grow
- Efficient insert & delete operations.



List (Linked Implementation) 3/4

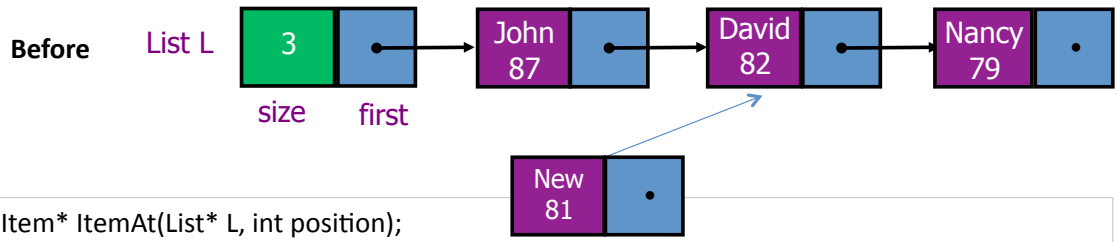
void Insert (Item X, int position, List *L);





List (Linked Implementation) 4/4

void Insert (Item X, int position, List *L);




```
Item* ItemAt(List* L, int position);

void Insert (Item X, int position, List *L) {
    …; //preconditions
    Item *itemNew = malloc(sizeof(Item)); //create a new student instance
    InitializeStudent(X.name, X.grade, itemNew); //why not just insert X?
    If(position != 0) {
        itemNew->next = itemAt(L, position);
        ItemAt(L, position-1)->next =  itemNew; //order matters
    } else ……
    L->size++;
    … //postconditions
}
```


Student (Revised)

Studentype.h

```
#define MAXNAMESIZE 20
typedef struct {
        char *name;
        int grade;
} Student;
```

StudentInterface.h

```
//interface the same, but implementation might differ, e.g.,
void InitializeStudent(char* name, int grade, Student* S) {
        …
        S->name = malloc(strlen(name)+1); //why plus one?
        strcpy(S->name, name);
        …
}
void FreeStudent(Student *S) {
        …
        free(S->name);
        …
}
```

Submission of Assignment 1

1. Create a root folder **CIS2520_LastNameFirstName_A1**

2. Create two subfolders **array** (contains all source code for the sequential implementation) and **list** (contains all source code for the linked implementation)

3. Each set of source code must come with a **makefile**, compilable with **GCC 4.4.0**. (which is ANSI C compliant) or later versions. The two sets of source code must be independent of one another (compilable independently). No change to naming (folders, files, data types, functions...) is allowed.

4. Zip the root folder (**CIS2520_LastNameFirstName_A1**) and upload it to **Moodle**. The enrollment key is **SOCS2520**

NOTE: If your computer is not equipped with **gcc**, follow the guidelines below.

- Windows : http://www.mingw.org/
- Mac: http://developer.apple.com/technologies/tools/
- Linux/Unix: http://gcc.gnu.org/

APPENDIX: Linked List Variation