PROGRAMMING TECHINQUES

FINAL REVIEW

TOPICS

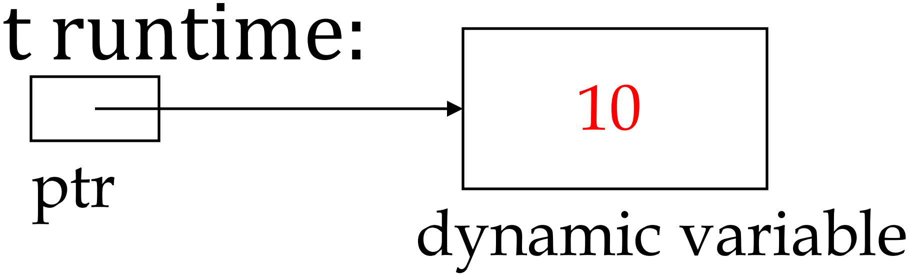
1. Pointers & Dynamically Allocated Arrays
2. Linked List
3. Stack, Queue
4. Recursion
5. Sorting
6. Binary File

POINTER

- In C++, a pointer is just a different kind of variable
  - Size: 4 bytes
  - Value: memory address of another variable/object
  - Pointers must be defined and then initialized
    - int* p;
    - int* p = NULL; int* p = 0; int* p(0);
    - int* p = &a;
- Usage:
  - Use data structures that grow and shrink as the program is running:
    - Dynamic array
    - Linked list

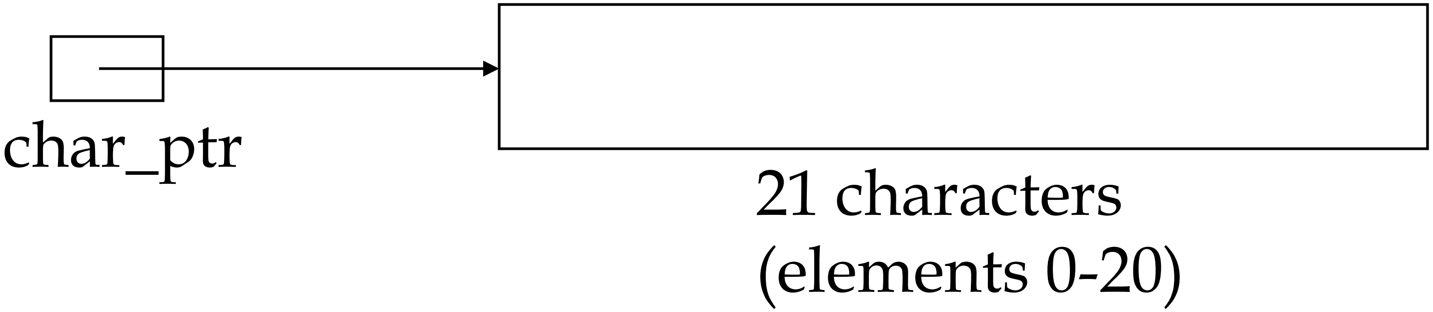
ALLOCATING – DEREFERENCE – DEALLOCATING

- Allocating memory at runtime:
  - int* ptr = new int;



- Dereference:
  - *ptr = 10;
- Deallocating the dynamic memory:
  - delete ptr; //This does not delete ptr
  - ptr = NULL;

DYNAMIC ARRAY



- int size = 21;
- char* char_ptr = new char[size];
- cin.get(char_ptr,21,’\n’);
- delete [] char_ptr;
- char_ptr = NULL;

Segmentation Fault!

- Dereferenced NULL pointer.
- Step outside of array.
- Access memory that has already been deallocated.

POINTER ARITHMATIC

- Pointer is prefered instead of an array because the variable pointer can be incremented/decremented, unlike the array name which cannot be changed because it is a constant pointer.
  - int arr[4] = {1, 3, 5, 8};
  - int* ptr = arr;
  - ptr++; //arr[1]
  - ++ptr; //arr[2]
  - ptr--; //arr[1]
  - --ptr; //arr[0]
  - ptr=ptr + 4; //pointer points to arr[4]
  - int* q = &arr[1];
  - q – ptr; //number 1

(pointer) + (int) = (pointer)

(pointer) – (pointer) = (int)

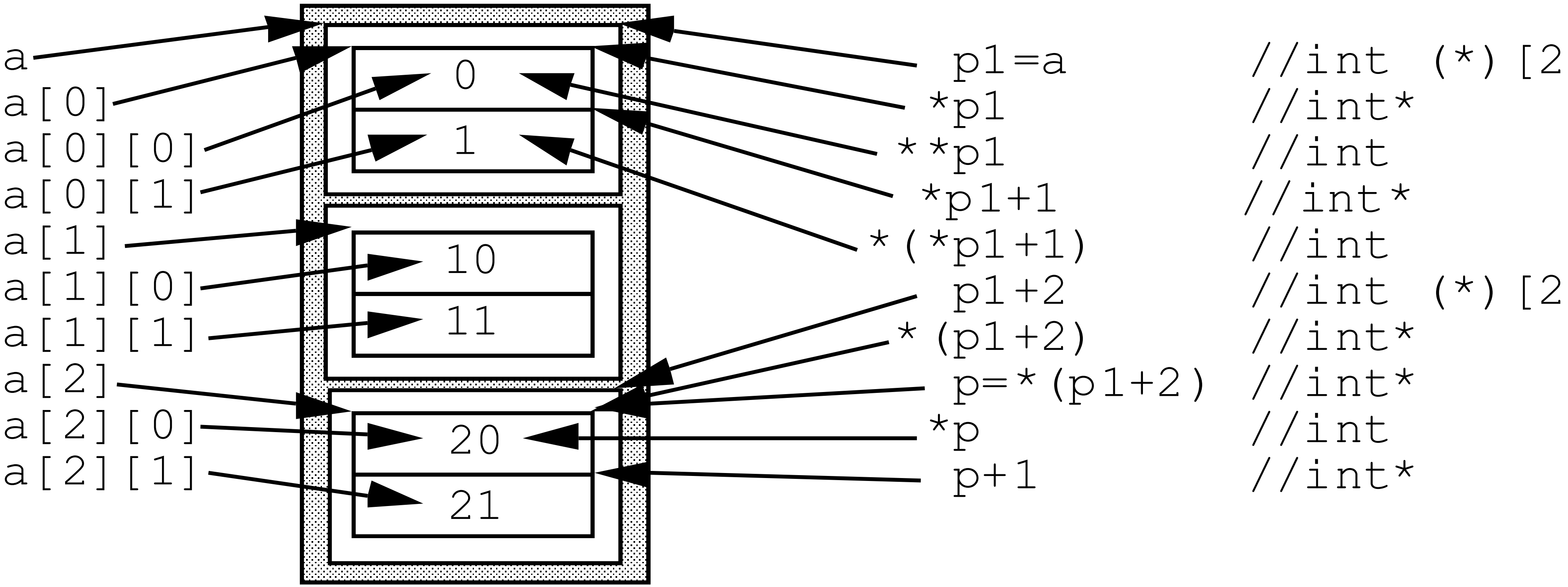
ARRAY OF ARRAY (MULTIDIMENSIONAL ARRAY)

```
int array[6][2];
int (*p1)[2];   //define pointer of same type as array
p1 = array;     //assign pointer to point to array

int *p;         //define ptr of same type as subarray
p = *p1;        //assign ptr to point to 1st subarray
p = array[0];   //this also points to 1st subarray and
p = *(array+0); //so does this because of our identity
p = *array;     //and so does this
```

ARRAY OF ARRAY (MULTIDIMENSIONAL ARRAY)

```
int a[3][2]={{0,1},{10,11},{20,21}};
```



a
a[0]
a[0][0]
a[0][1]
a[1]
a[1][0]
a[1][1]
a[2]
a[2][0]
a[2][1]

0
1
10
11
20
21

```
 p1=a         //int (*)[2
*p1           //int*
**p1          //int
*p1+1         //int*
*(*p1+1)      //int
 p1+2         //int (*)[2
*(p1+2)       //int*
 p=*(p1+2)    //int*
*p            //int
 p+1          //int*
```

POINTER AND STRING

```
char animal[20] = "bear";      //animal holds bear
const char* bird = “pigeon";//initialize a pointer-to-char
                     //to a string -> assign the address
                     //of pigeon to pointer bird
char* ps;// uninitialized
ps = animal;// set ps to point to string
cout << animal << " is at " << (int *) animal << endl;
cout << ps << " is at " << (int *) ps << endl;
```

- animal and ps are pointers of type char → cout displays the pointed-to-string.
- If you want to see the address of the string, you have to type cast the pointer to another pointer type, such as (int *)

DYNAMIC STRUCTURE

```
struct Student{
    int ID;
    char name[20];
    float gpa;
};
```

- Allocate a student dynamically:

```
Student* st = new Student;
st->ID = 9.5;
```

st is a pointer to a student

- Allocate a list of students dynamically:

```
Student* st_lst = new Student[40];
st_lst[0].ID = 8.5;
```

st_lst is a pointer to the first student of a list 40 students

st[0] is not a pointer, it is a Student

…

PASS BY VALUE & PASS BY REFERENCE

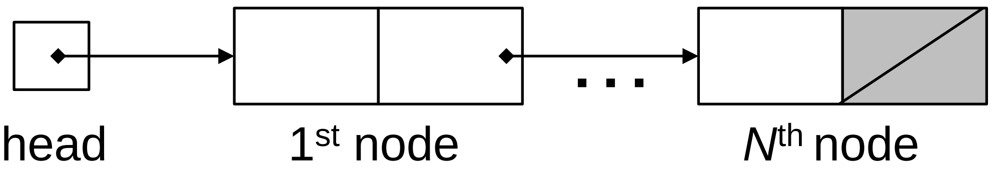
```
void Input(Student* & p);
void Display(Student* p);
int main()
{
    Student* st_ptr = new Student;
    Input(st_ptr);          //Call by reference
    Display(st_ptr);        //Call by value
    return 0;
}
```

DOUBLE POINTERS (POINTER TO POINTER)

- Double pointers are pointers that ***point to other pointers***. In other words, they are pointers that store the memory address of a pointer variable.
- Usage:
  1. Dynamic memory allocation
  2. Arrays of strings: `const char* strs[] = {"Hello", "world", "!"};`
  3. Multi-dimensional arrays
     ```
     int* arr[3];
     for (int i = 0; i < 3; i++)
        arr[i] = new int[4];
     ```
  4. Pointers to functions: `double (*pf) (int);`
  5. Data structures: Arrays of pointers are often used to store pointers to data structures.
     ```
     video * arr[10];
     ```
  6. Dynamic memory allocation: for example, create a 2D array.
  7. Pointer parameters to functions that need to modify a pointer variable.
     ```
     void createList(ListNode** head);
     ```
  8. Creating an array of pointers
     ```
     int* arr[3];
     int** ptr = arr;
     ```

SINGLY LINKED LIST

- Singly Linked List (Linear Linked List)



- Define a node structure

```
struct node {
  video data;
  node * pNext;  //a pointer to the next
};
```

- Define a linked list if needed

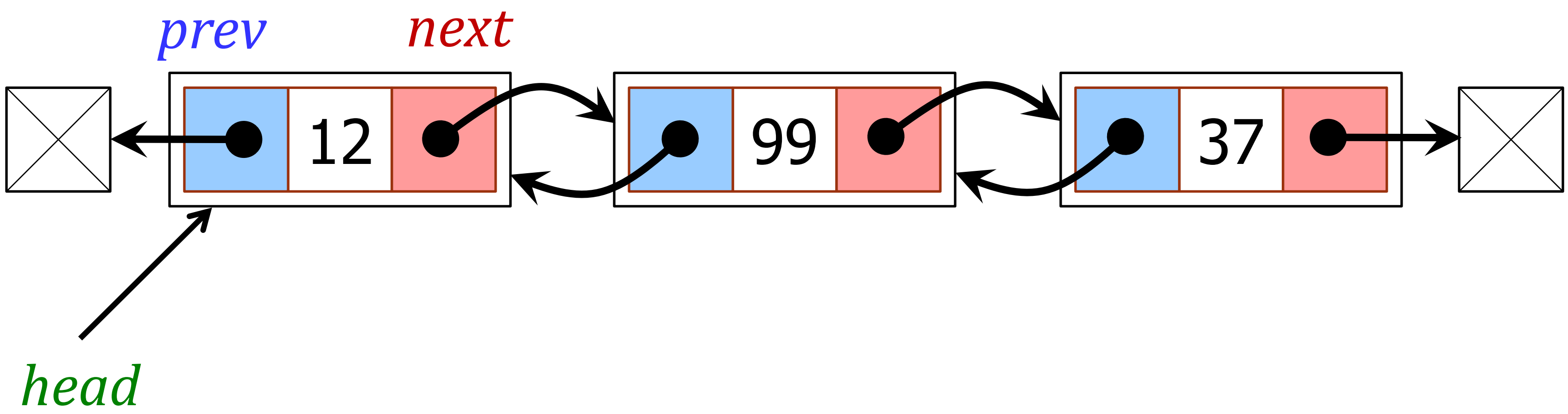
```
struct list {
  node * pHead;  //a pointer to the head
};
```

SINGLY LINKED LIST

- Operations:
  - ADD (Insert) at the beginning/middle/end
  - DELETE (Remove) at the beginning/middle/end
  - TRAVERSE

```
node* current = head;
while(current != NULL) //while(current)
{
        ....
        current = current->next;
}
```

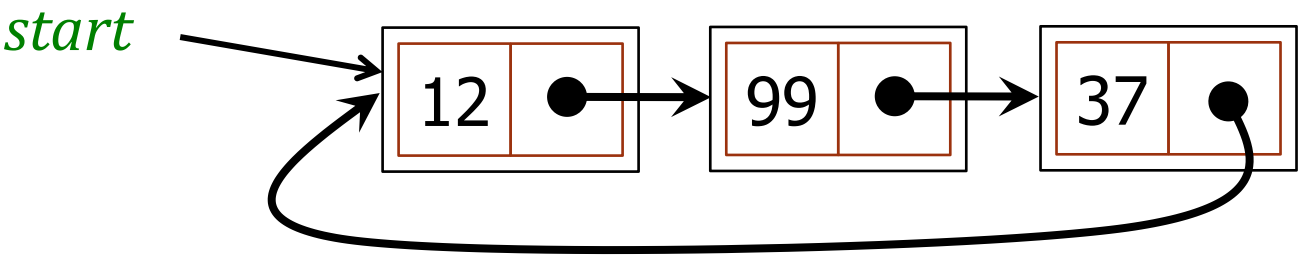
DOUBLY LINKED LIST



- Define a node structure

```
struct node {
  student data;
  node* pPrev; //a pointer to the previous
  node* pNext; //a pointer to the next
};
```

CIRCULAR LINKED LIST



- Define a node structure: singly circular linked list

```
struct node {
  POINT data;
  node* pNext; //a pointer to the next
};
```

- Define a node structure: doubly circular linked list

```
struct node {
  POINT data;
  node* pPrev; //a pointer to previous
  node* pNext; //a pointer to the next
};
```

STACK & QUEUE

- Stack: LIFO
  - Operation: PUSH, POP, TOP,
- Queue: FIFO
  - Operation: ENQUEUE, DEQUEUE, FIRST
  - Priority Queue

RECURSION

- 3 steps to write a recursive algorithm:
  1. Determine the base case
     - The simplest case for which you know the answer
     - The function returns when this condition meets
  2. Determine the general case
     - The one where the problem is expressed as a smaller version of itself
  3. Verify the algorithm
     - Your code must have a case for all valid inputs

SORTING

- Selection Sort
- Bubble Sort
  - Enhancement: Interchange Sort, Comb Sort, Cocktail Sort
- Insertion Sort
  - Enhancement: Binary Insertion Sort, Shell Sort

BINARY FILE

#include <fstream>

- Open a binary file:
  - ifstream fin(“test.dat”, ios::in | ios::binary);
  - ofstream fout(“test.dat”, ios::out | ios::app | ios::binary);
- Read a block of memory from a binary file:

  fin.read((char *)& s, sizeof(s));
- Write a block of memory to a binary file:

  fout.write((char *)& s, sizeof(s));
- The get and put cursors of a stream:
  - tellg(); tellp();
  - seekg(position); seekp(position);