# VG101 Recitation Class - Week 8

Zhang Yichi

UM-SJTU Joint Institute

July 5, 2018

# Dynamic Memory Allocation

You can use C functions to dynamically allocate memory for pointers you create.

- To allocate new memory: malloc(n),calloc(n,s)
- To change the size of memory allocated to the pointer: realloc(ptr,s)
- To free the space: free(ptr)

In this way the pointer will be allocated with a few chunks of space and you can also use pointers like (ptr+n) if this pointer is within the space allocated.

Note

- Be careful not to use pointers that is out of the space that is allocated.
- For every pointer that is allocated some space, you should free it for exactly one time.

# Pointer and Structure

As you have defined a structure, you can use pointers for your structures just like those for the normal data types.

## Access Members in the Strcutre

The calling mechanism for structure members is different between a structure variable and a pointer to that structure.
Now suppose you have a structure person with a member age

- If you declare a variable person p1, then you should use p1.age to access the age member.
- If you create a pointer person *p2 and you have assign it the address for a person variable or allocated it with some space, you should use p2->age to access its age member.

# NULL Pointer and Uninitialized Pointer

- A NULL pointer is a pointer that points to nowhere.
- An uninitialized pointer can have an address value that points to anywhere in your computer.

**Note**

Do not try to access the object the pointer points to when the pointer is NULL or uninitialized.

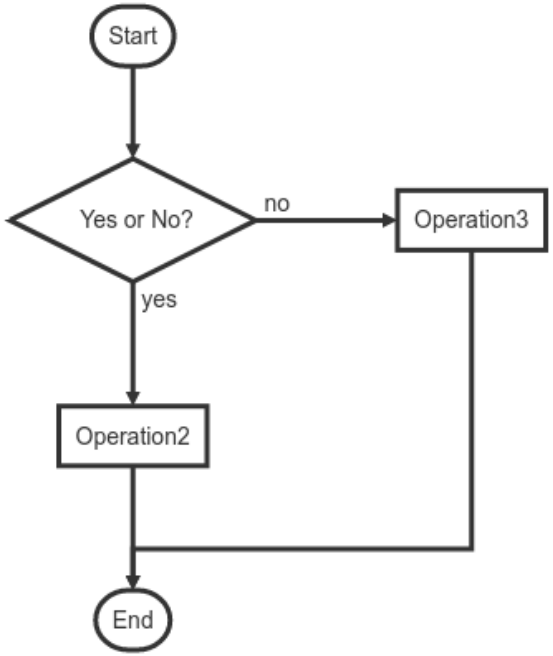
# Arrays and Pointers

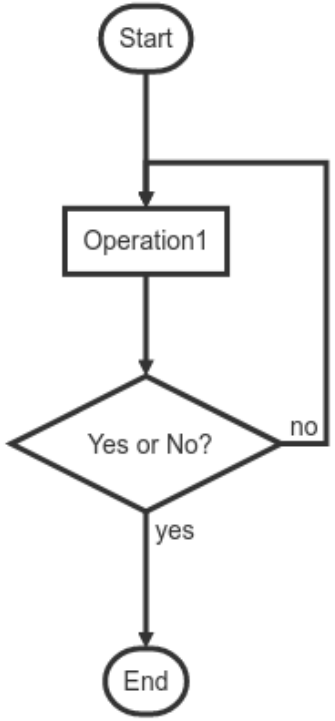
- When you define an array, for example int a[], you can use a as a pointer to the first element in the array.
- And when you declare a pointer, for example int *b and allocate some memories to it, you can also use some methods for arrays such as b[i](be careful not to cause memory problems).
- When you pass an array into a function, it is more like passing the head pointer of it.

# Algorithms

- As you have already learned previously, algorithm means the way to solve a problem.
- The three necessary elements of an algorithm are: input, output and instructions
- To think about an algorithm or optimize an algorithm, the things you should think about are:
  - Get clear about the form of the input and output.
  - Consider how easy your algorithm is to implement with your code.
  - Consider the time/space cost of your algorithm.
  - Think about how clear it is for others to understand.
  - After all, the accuracy is the most important part of your algorithm.

# Ways to Show an Algorithm: Flowchart



# Ways to Show an Algorithm: Pseudocode

**Algorithm 1** Insert Sort

**Require:** Array a, Size n

**Ensure:** Array a be sorted in descending order

```
1:  for i = 0 → n − 1 do
2:      for j = i + 1 → n − 1 do
3:          if a[i] < a[j] then
4:              temp ← a[i]
5:              a[i] ← a[j]
6:              a[j] ← temp
7:          end if
8:      end for
9:  end for
10:
```

# File I/O in C

- Open a file: FILE *fopen(const char *path,const char *mode)
  - Returns the pointer to the file you want to open.
  - Returns NULL if opening fails.
  - Every file opened should be closed using fclose()
- Change the "current position" of file operation:
  int fseek(FILE *stream,long offset, int whenence)
  - It will change the current position of file operation to the position which is offset characters away from the position whenence
  - Returns 0 if it succeeds, and return a non-zero number otherwise
- Tell the current position: int ftell(FILE *stream)
- Write to file: fprintf
- Read from file: fscanf, fgets, getc
  - Note: fgets will also stop reading if a newline character or the end-of-file is meet

## Exercise

Suppose your friend has a pile of cards with integers written on them, but you don't know the number of cards in his hand. Then your friend do the following two steps repeatedly.

- 1. Give you the card on the top of the pile.
- 2. Put the card on the top of the pile to the bottom.

So in your perspective, what you get is a sequence of cards your friend gives you.
And what you should do is to work out the original order of those cards based on the order that they are given to you.
Input: A file "order.txt", with integers which is in the order that you get the cards, and one integer is in one line. Note: You don't know the exact number of integers in this file
Output: Print those numbers in the order that they are originally in your friend's hand.

This part is for the file I/O. Since the total number in the file is unknown, the program should read until it meets the end of file. And since the size of the array is unknown, a dynamic allocation is needed.

```
1   #include<stdio.h>
2   #include<stdlib.h>
3
4      int main(){
5        FILE *stream = fopen("order.txt","r");
6        int *a = malloc(sizeof(int));
7        //Pointer a serves as an array
8        int count = 0;
9        while (fscanf(stream,"%d",a+count)!=EOF){
10         count++;
11         a = realloc(a,(count+1)*sizeof(int));
12         //Dynamic allocation as the total number is
   unknown
13       }
14
```

- Now we know the number of cards(count) and the order of them that I received from my friend(a).
- It is time to solve this problem. Many of you might think of "reversing" the whole process, and I believe you can implement this reversing process based on your knowledge.
- So I will offer an algorithm from another perspective. Let me explain my algorithm in a non-programming way.

- First I make some new cards with number $1, 2, 3, \ldots,$ *count* on them and put them in a pile just in the order of $1, 2, 3, \ldots$*count*
- Then I give this new pile of cards to my friend and ask him to do the same process for me again.
- Now I receive another sequence of cards. But this time I know exactly where each card was in the original pile (because it is just the number on it).
- And it is clear that the situation in this game should be exactly the same as the situation in the previous game, so now I can tell the position of every card in the previous game now.

```
1       int *b = malloc(count*sizeof(int));
2       for (int i=0;i<count;i++) b[i]=i;
3       //The new pile of cards is made
4       int top = 0;
5       while (top!=count-1){
6         top++;
7         //Here the card-giving process is implement by the
        shifting of the starting point
8         int temp = b[top];
9         for (int j=top;j<count-1;j++) b[j]=b[j+1];
10        b[count-1] = temp;
11        //Changing place is implemented by putting the top
        element in the bottom and shifting all other elements
        forward
12      }
13
```

```
1         int *c = malloc(count*sizeof(int));
2         for (int i=0;i<count;i++) c[b[i]] = a[i];
3         for (int i=0;i<count;i++) printf("%d ",c[i]);
4         printf("\n");
5         free(a);free(b);free(c);
6         fclose(stream);
7         //Do not forget these two steps
8         return 0;
9     }
10
```