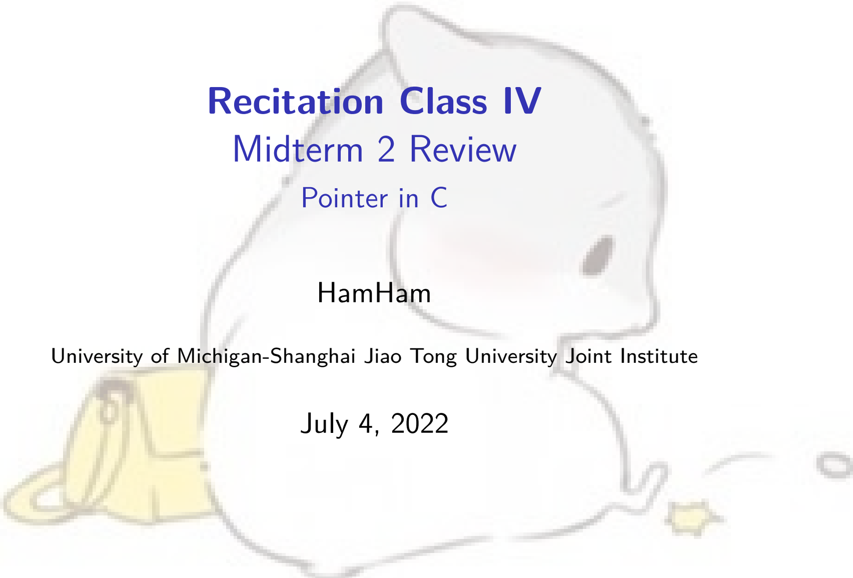


Recitation Class IV

Midterm 2 Review

Pointer in C

HamHam

University of Michigan-Shanghai Jiao Tong University Joint Institute

July 4, 2022

VG101 - Intro to Computers & Programming

Pointer

What is a pointer?

A pointer is a **variable** whose value is the **address** of another variable, i.e., direct address of the memory location.

What is the size of pointer?

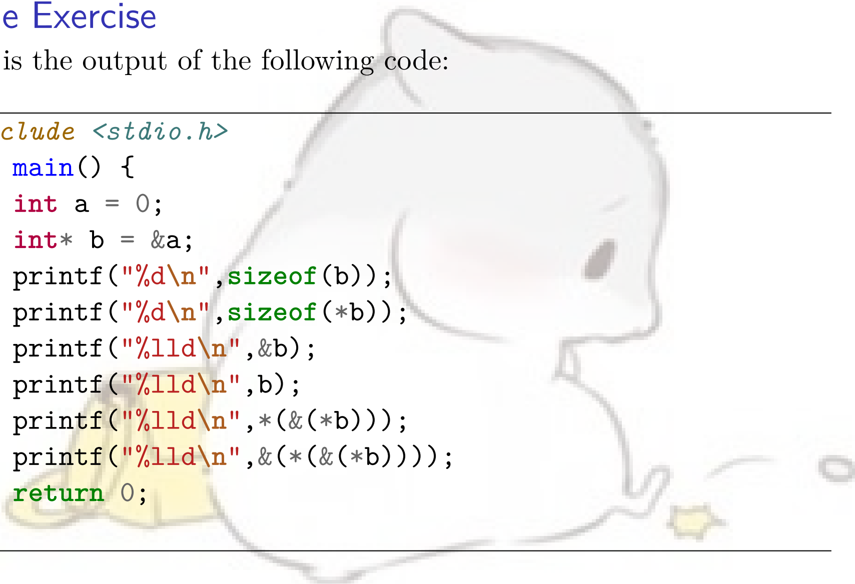
- In 32-bit system, the size of pointer is 4 Bytes.
- In 64-bit system, the size of pointer is 8 Bytes.
- No matter what type of data the pointer points to.

Handling pointers:

- The address of a variable x is &x
- The value stored at address y is *y

Simple Exercise

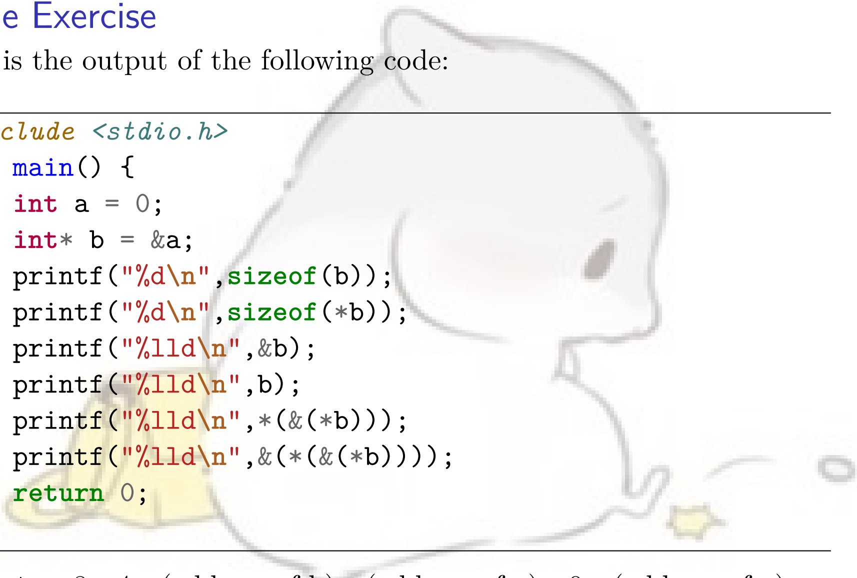
What is the output of the following code:



```c
#include <stdio.h>
int main() {
    int a = 0;
    int* b = &a;
    printf("%d\n",sizeof(b));
    printf("%d\n",sizeof(*b));
    printf("%lld\n",&b);
    printf("%lld\n",b);
    printf("%lld\n",*(&(*b)));
    printf("%lld\n",&(*(&(*b))));
    return 0;
}
```

Simple Exercise

What is the output of the following code:



```c
#include <stdio.h>
int main() {
    int a = 0;
    int* b = &a;
    printf("%d\n",sizeof(b));
    printf("%d\n",sizeof(*b));
    printf("%lld\n",&b);
    printf("%lld\n",b);
    printf("%lld\n",*(&(*b)));
    printf("%lld\n",&(*(&(*b))));
    return 0;
}
```

Output: 8 4 (address of b) (address of a) 0 (address of a)

Pointer and function

```c
#include <stdio.h>
void swap(int a,int b);
int main() {
    int a=2, b=5;
    swap(a,b);
    printf("a = %d, ",a);
    printf("b = %d\n",b);
    return 0;
}
void swap(int a,int b) {
    int temp=a;
    a=b;
    b=temp;
}
```

```c
#include <stdio.h>
void swap(int* a,int* b);
int main() {
    int a=2, b=5;
    swap(&a,&b);
    printf("a = %d, ",a);
    printf("b = %d\n",b);
    return 0;
}
void swap(int* a,int* b) {
    int temp=*a;
    *a=*b;
    *b=temp;
}
```

Dynamic Memory

- Allocate n bytes of memory, and get a pointer on the first chunk

```c
p = malloc(n);
int* q = (int*) malloc(n*sizeof(int));
```

- Allocate n blocks of size s each, set the memory to 0, and get a pointer on the first chunk

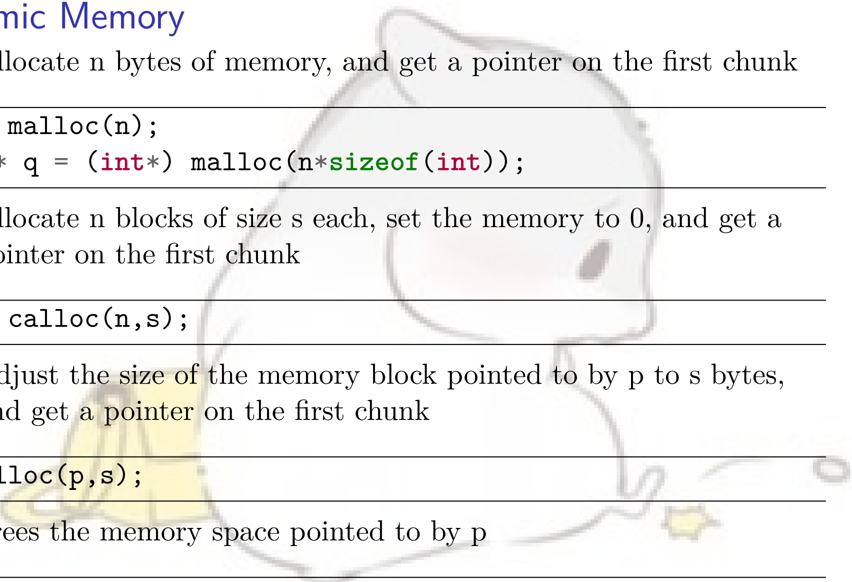
```c
p = calloc(n,s);
```

- Adjust the size of the memory block pointed to by p to s bytes, and get a pointer on the first chunk

```c
realloc(p,s);
```

- Frees the memory space pointed to by p

```c
free(p);
```



Remarks

- Not possible to choose the address, e.g. `int *p; p=12345;`
- The `NULL` pointer "points nowhere"
- An uninitialized pointer "points anywhere", e.g. `float *a;`

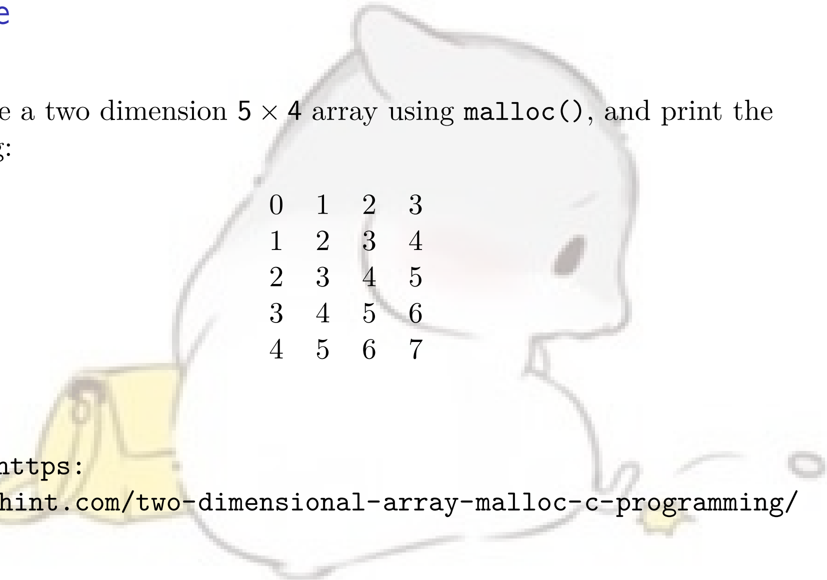
What is the difference?

```
      Vector *vec = (Vector *) malloc(sizeof(Vector));
      Vector vec2;    Vector *vec3 = &vec2;
```

Answer:

For the variable allocated by `malloc()`, it will keep alive until you manually call `free()` or the entire program terminates (cause memory leak). However, for `vec2`, it's just a local variable, and will be cleared as long as the variable scope ends. You must not `free()` vec3!!!

Exercise



1. Create a two dimension 5×4 array using `malloc()`, and print the following:

$$\begin{matrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \\ 3 & 4 & 5 & 6 \\ 4 & 5 & 6 & 7 \end{matrix}$$

credit : `https://linuxhint.com/two-dimensional-array-malloc-c-programming/`

Open a file

Syntax

```
FILE * fp = fopen(filename, mode);
...
fclose(fp);
```

- mode:r,w,a, r+, w+, a+
- judge if it opens successfully: `if(!fp) { deal with failure }`
- If you just use the filename, then the file and your program should be in the same folder.
- Closing the file is really important!!!

Read from a file

Syntax:

```
fscanf(fp,"%d",x);
fgets(str,countMax,fp);
```

Read integers until End

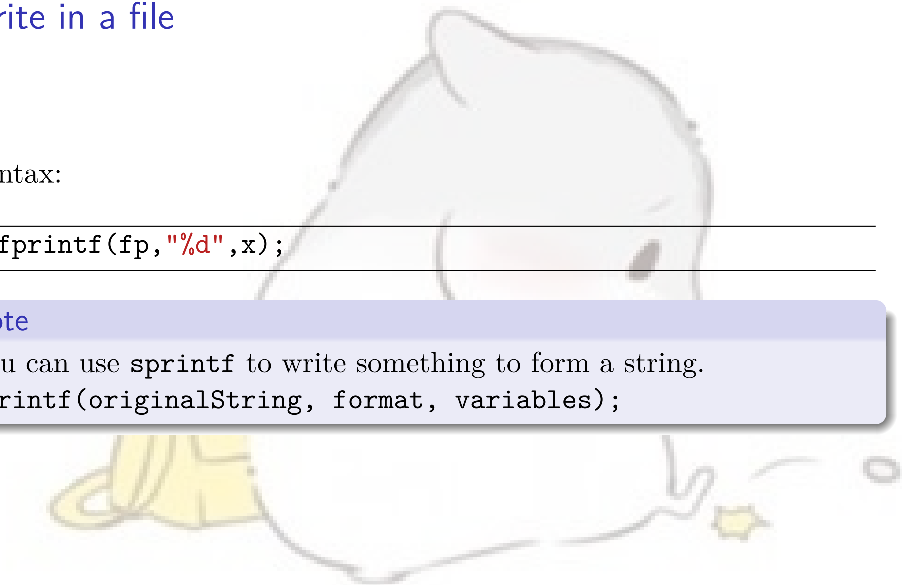
```
while (fscanf(f,"%d",&x)!=EOF) {printf("%d ",x);}
```

Note

You can use `sscanf` to read something from a string.

```
sscanf(originalString, format, variables);
```

Write in a file



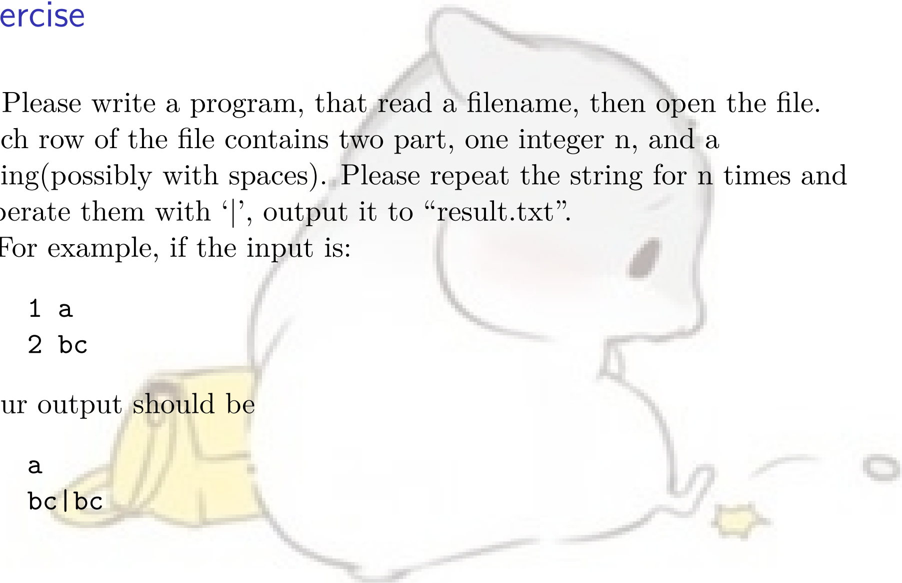
Syntax:

```
fprintf(fp,"%d",x);
```

Note

You can use `sprintf` to write something to form a string.
`sprintf(originalString, format, variables);`

Exercise



2. Please write a program, that read a filename, then open the file. Each row of the file contains two part, one integer n, and a string(possibly with spaces). Please repeat the string for n times and seperate them with '|', output it to "result.txt".

For example, if the input is:

```
1 a
2 bc
```

Your output should be

```
a
bc|bc
```

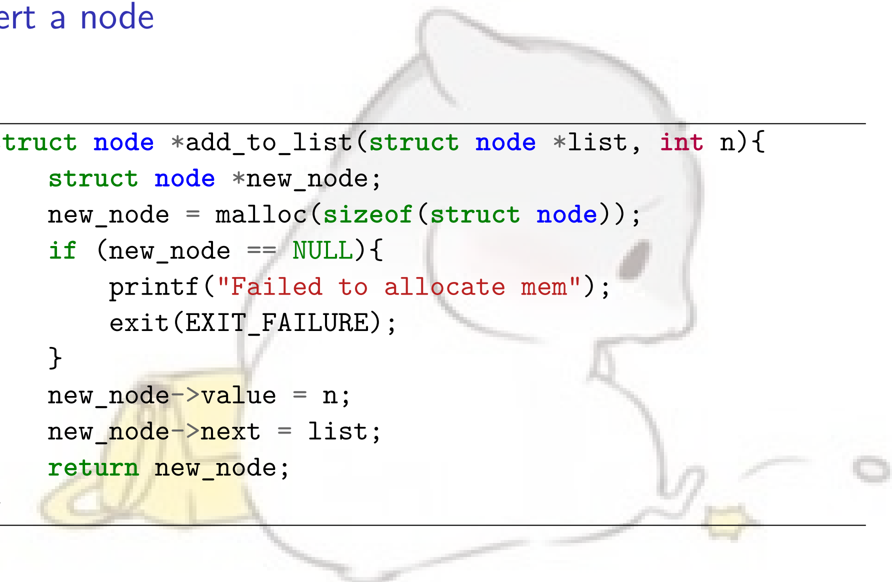
Linked List

```
// Definition for singly-linked list
struct node {
    int value;
    struct node *next;
};
```

Operations:

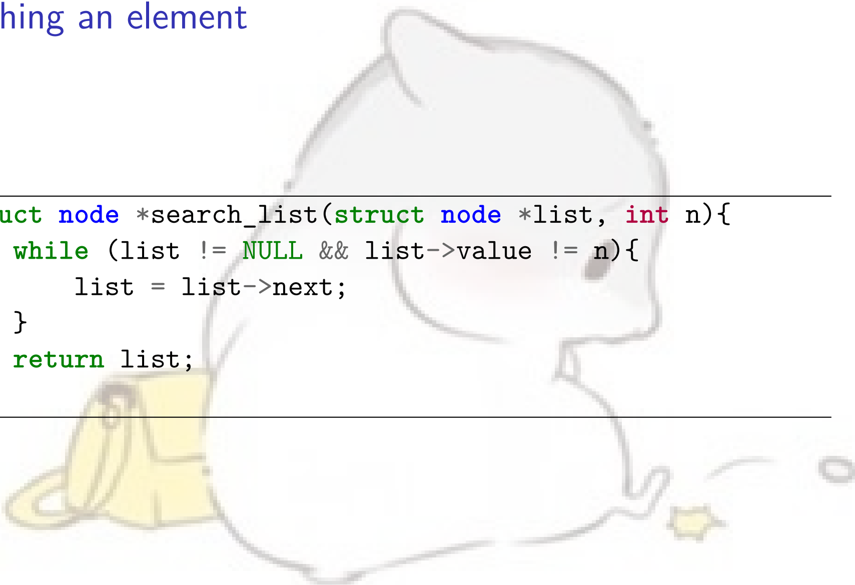
- traverse
- create
- copy
- find
- delete (front, end, middle)
- insert (front, end, middle)
- connect
- free

Insert a node



```c
struct node *add_to_list(struct node *list, int n){
    struct node *new_node;
    new_node = malloc(sizeof(struct node));
    if (new_node == NULL){
        printf("Failed to allocate mem");
        exit(EXIT_FAILURE);
    }
    new_node->value = n;
    new_node->next = list;
    return new_node;
}
```

Searching an element



```c
struct node *search_list(struct node *list, int n){
    while (list != NULL && list->value != n){
        list = list->next;
    }
    return list;
}
```

Node deletion

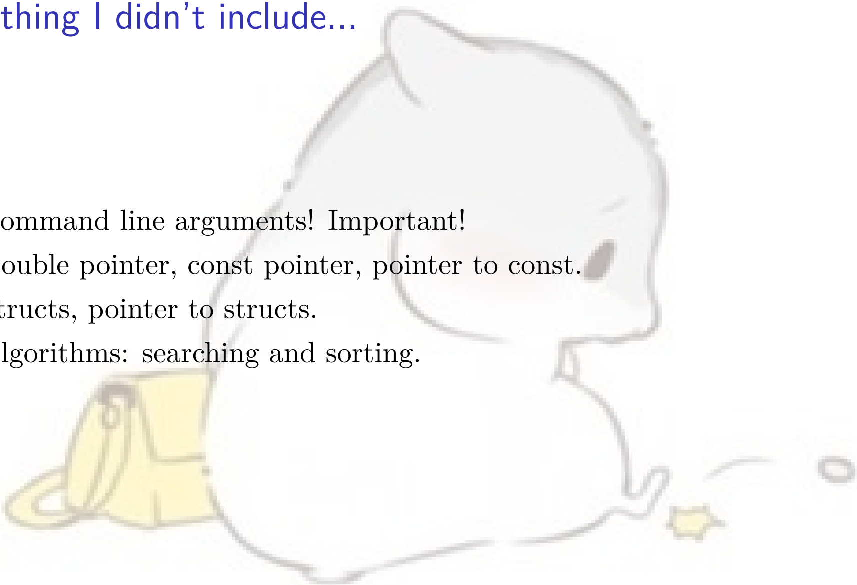
```
struct node *delete_from_list(struct node *list, int n){
    struct node *cur, *prev;
    for (cur = list, prev = NULL; cur != NULL && cur->value
    n; prev = cur, cur = cur->next);
    if (cur == NULL)
        return list;// n was not found
    if (prev == NULL)
        list = list->next; // n is in the first node
    else
        prev->next = cur->next; // n is in middle of list

    free(cur);
    return list;
}
```

Exercise

3. Implement the following functions (in `list.c`):

```c
node_t * merge_list(node_t * l1,node_t * l2){
    //return a pointer point to the first node of
    //the merged list
    //TO DO: implement
}
node_t* split(node_t ** list1, int n){
    //drop all the nodes after the n-th node to be a
    //splitted list and return a pointer point to
    //the first node of the splitted list
    //TO DO: implement
}
```

Something I didn't include...



- Command line arguments! Important!
- Double pointer, const pointer, pointer to const.
- Structs, pointer to structs.
- Algorithms: searching and sorting.

Reference

- Dr. Charlemagne, Lecture Slides.
- Dr. Zhu, Yifei. VG101-2022SU Lecture Slides.
- Zhu, Kan. VG101-2021SU-RC6&7&8 Slides
- Zhang, Boming. VG101-2020FA-Mid2 Slides.
- Yu, Zesheng. VG101-2020FA-Mid2 Slides.

End

Thanks!