

Review on pointer to pointer

```
#include<stdio.h>
```

```
void main()
```

```
{
```

```
    int i =50;
```

```
    int **ptr1;
```

```
    int *ptr2;
```

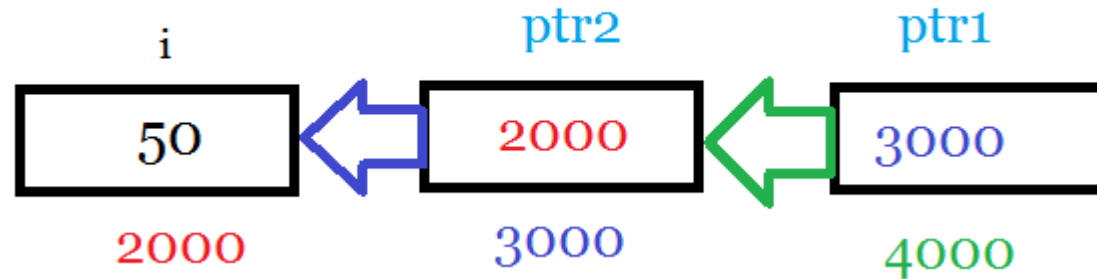
```
    ptr2 = &i;
```

```
    ptr1 = &ptr2;
```

```
    printf("\nThe value of **ptr1 : %d",**ptr1);
```

```
    printf("\nThe value of *ptr2 : %d",*ptr2);
```

```
}
```



void *

- void is an universal container
- Save first, and then process based on type

```
int main (void)
{
    int i;
    float f;
    void *ptr;

    i=30;
    f=20.0;

    ptr = (void *) &i;
    print_content(ptr, 0);

    ptr = (void *) &f;
    print_content(ptr, 1);
}
```

```
print_content(void *ptr, int type)
{
    if (type == 0)
    {
        printf ("content:%d\n", *((int*)ptr));
    }
    else if (type == 1)
    {
        printf("content :%f\n", *((float *)ptr));
    }
}
```

W14-on site assignment

- Follow the assignment last week
- Modify the original program to have one universal queue (which contains a small queue and a large queue)
- Use the **void** technique to maintain the queue

C queue.h > node_info > content

```

1  #ifndef __QUEUE__
2  #define __QUEUE__
3
4  typedef struct type_small {
5      int id;
6      int location;
7      int score;
8  }tQueueSmall;
9
10 typedef struct type_large {
11     int id;
12     int location;
13     int score[8];
14 }tQueueLarge;
15
16 typedef struct node_info {
17     int type;
18     void *content;
19     struct node_info *next;
20     struct node_info *prev;
21 }tQueueNode;
22
23 typedef struct {
24     tQueueNode *front;
25     tQueueNode *rear;
26     int count;
27 }tQueue;

```

queue.c

```

19 int tqueue_enqueue(tQueue *queue, int id, int score, int type)
20 {
21     tQueueNode *queue_node;
22     void *newptr = NULL;
23     int mem_location;
24
25     queue_node = (tQueueNode *)malloc(sizeof(tQueueNode));
26     our_malloc (type, (void *)&newptr, &mem_location);
27
28     if (newptr == NULL)
29     {
30         printf("    Enqueue Failed !!! \n\n");
31         return 0;
32     }
33

```

queue.h C space.h X

C space.h > ...

```
1  #ifndef __SPACE__
2  #define __SPACE__
3
4  #include "main.h"
5
6  #define NUM_SMALL_BYTE_BUF      8
7  #define NUM_LARGE_BYTE_BUF     8
8
9  #define SMALL_ELEMENT_SIZE      32
10 #define LARGE_ELEMENT_SIZE      64
11 #define LARGE_START              (SMALL_ELEMENT_SIZE*NUM_SMALL_BYTE_BUF)
```

space.c X

C space.c > our_free(int, int)

```
1  #include "space.h"
2
3
4  unsigned char buffer[SMALL_ELEMENT_SIZE*NUM_SMALL_BYTE_BUF + NUM_LARGE_BYTE_BUF*LARGE_ELEMENT_SIZE];
5
6  unsigned char byte_large_buf_mask = 0;
7  unsigned char byte_small_buf_mask = 0;
```