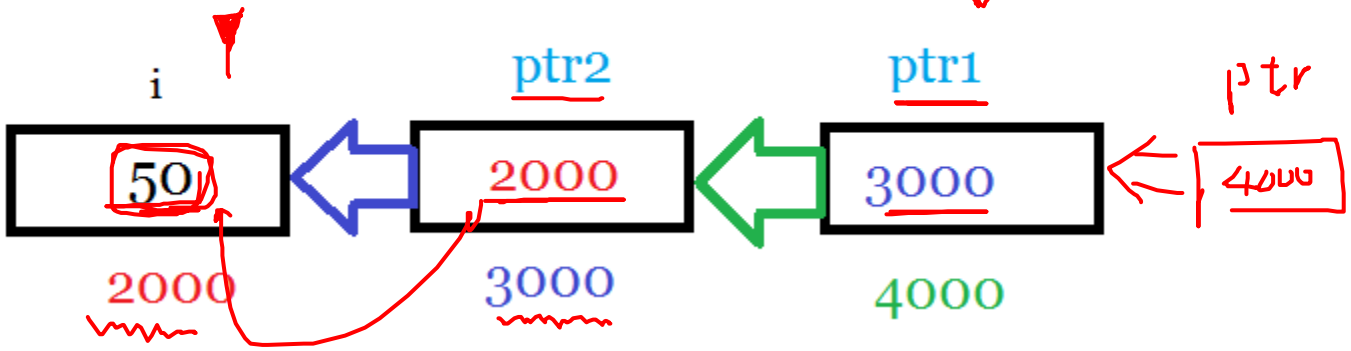
Review on pointer to pointer

```
#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 *

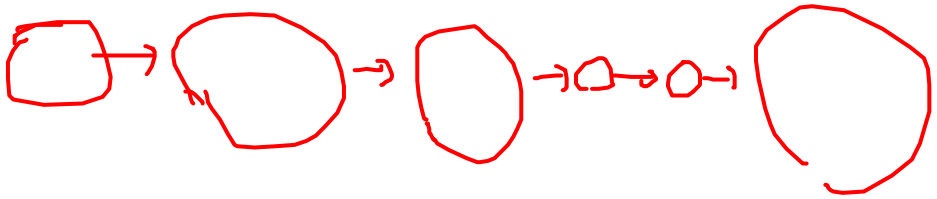
- void is an universal container
- Save first, and then process based on type

```
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));
  }
}
```

W13-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
- No warning is allowed, only 2 malloc is allowed

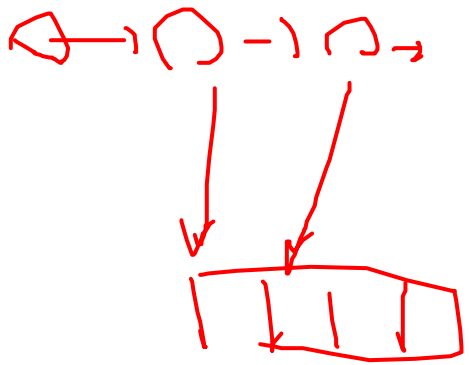
queue.c queue.h

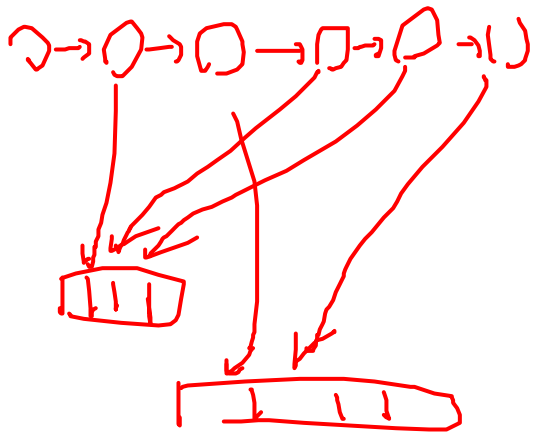
queue.h > node_info > content

```c
#ifndef __QUEUE__
#define __QUEUE__

typedef struct type_small {
    int id;
    int location;
    int score;
}tQueueSmall;

typedef struct type_large {
    int id;
    int location;
    int score[8];
}tQueueLarge;

typedef struct node_info {
    int type;
    void *content;
    struct node_info *next;
    struct node_info *prev;
}tQueueNode;

typedef struct {
    tQueueNode *front;
    tQueueNode *rear;
    int count;
}tQueue;
```

queue.c

```c
int tqueue_enqueue(tQueue *queue, int id, int score, int type)
{
    tQueueNode *queue_node;
    void *newptr = NULL;
    int mem_location;

    queue_node = (tQueueNode *)malloc(sizeof(tQueueNode));
    our_malloc (type, (void *)&newptr, &mem_location);

    if (newptr == NULL)
    {
        printf("    Enqueue Failed !!! \n\n");
        return 0;
    }
```





queue.h | space.h

space.h > ...

```c
#ifndef __SPACE__
#define __SPACE__

#include "main.h"

#define NUM_SMALL_BYTE_BUF      8
#define NUM_LARGE_BYTE_BUF      8

#define SMALL_ELEMENT_SIZE      32
#define LARGE_ELEMENT_SIZE      64
#define LARGE_START             (SMALL_ELEMENT_SIZE*NUM_SMALL_BYTE_BUF)
```

space.c

space.c > our_free(int, int)

```c
#include "space.h"


unsigned char buffer[SMALL_ELEMENT_SIZE*NUM_SMALL_BYTE_BUF + NUM_LARGE_BYTE_BUF*LARGE_ELEMENT_SIZE];

unsigned char byte_large_buf_mask = 0;
unsigned char byte_small_buf_mask = 0;
```