Topic 9: Pointer to pointer

call by value

```
int main (void)
{
    int a=10;
    int b=20;
    swap (a,b);
    printf("a=%d, b=%d", a,b);
    return 0;
}

void swap (int aa, int bb)
{
    int temp;
    temp = aa;
    aa = bb;
    bb = temp;
}
```

aa and bb are the copies of a and b
→swap not work

call by address

```
int main (void)
{
    int a=10;
    int b=20;
    swap (&a, &b);
    printf("a=%d, b=%d", a,b);
    return 0;
}

void swap (int *aa, int *bb)
{
    int temp;
    temp = *aa;
    *aa = *bb;
    *bb = temp;
}
```

pass the container of a and b to swap()

aa and bb point to the memory of a and b

*aa and *bb exchange the contents of a and b

→swap works

Can allocate?

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

void Allocate (int *p)
{
    p = (int *) malloc (sizeof(int));
}

int main (void)
{
    int *p;

    p = NULL;
    Allocate(p);

    if (p == NULL)
        printf ("allocate NOT OK\n");
    else
        printf ("allocate OK\n");
    return 0;
}
```

Pointer to pointer **p

- Common use to redirect a pointer to another space

```
in main.c

int main (void)
{
    int a = 10;
    int *p;
    p = &a;
    printf(" *p=%d ", *p);

    changeP(&p);
    printf(" *p=%d ", *p);
}
```
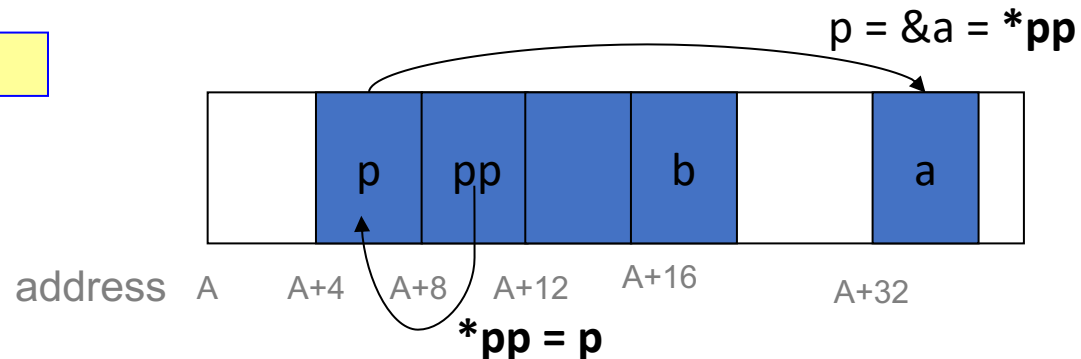
variable of **a**

variable of **b**

```
in change.c

int b = 100;

void changeP (int **pp)
{
    *pp = &b;
    **pp = 1000;
}
```

Direct to a reference → use one *

The type in the reference is a pointer →
use another *

p = &a = **pp

| | p | pp | | b | | a | |

address A   A+4   A+8   A+12   A+16   A+32

*pp = p

4

Pointer & structure & linked list & pointer to pointer

```
    pStudent = AddStudent (pStudent, 20, 40);
    pStudent = AddStudent (pStudent, 52, 100);

    pStudent = head;
    while (pStudent != NULL)
    {
        printf("ID: %d with score: %d \n",
            pStudent->ID, pStudent->score);
        pStudent = pStudent -> next;
    }
    return 0;
}
```

```
    AddStudent (&pStudent, 20, 40);
    AddStudent (&pStudent, 52, 100);

    pStudent = head;
    while (pStudent != NULL)
    {
        printf("ID: %d with score: %d \n",
            pStudent->ID, pStudent->score);
        pStudent = pStudent -> next;
    }
    return 0;
}
```

```
tReg * AddStudent (tReg *p, int ID, int score)
{
    p->next = (tReg *) malloc (sizeof(tReg));
    p->next->ID = ID;
    p->next->score = score;
    p->next->next = NULL;

    return (p->next);
}
```

OLD fashion

```
void * AddStudent (tReg **p, int ID, int score)
{
    (*p)->next = (tReg *) malloc (sizeof(tReg));
    (*p)->next->ID = ID;
    (*p)->next->score = score;
    (*p)->next->next = NULL;

    *p = (*p)->next;
}
```

NEW fashion

5

A use case of pointer to pointer

- In an embedded system
    - Most system resources are predefined
    - To dynamic allocate memory is time-consuming
- Solution
    - Take a pre-allocated space as a ring buffer
    - Use pointer to pointer to return a pre-allocated space for a process
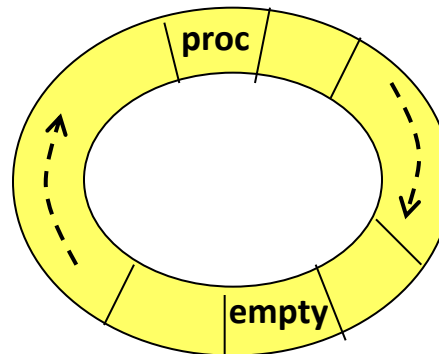
```c
in receiveSdu.c

int receiveSdu ()              Callback or ISR
{
    char *p;
    getSpaceForSdu(&p);
    copySduFromHW(p);
}
```

```c
in bufManager.h

#define SIZE 100
typedef char Buf[1024];

typedef   struct   tBufSt
{
    int empty;
    int processing;
    Buf buf[SIZE];                Ring buffer
} tSduBuf;                        implementation
```

```c
in bufManager.c

tSduBuf pMem;

void getSpaceForSdu (int **pp)
{
    *pp = &pMem.buf[empty];        Ring buffer usage
     empty = (empty + 1) % SIZE
}
```

W12-on-site assignment

- Use the provided stack.c to implement your own stack API

- Use pointer to pointer to allocate memory

- You can have only two malloc in your entire program
  - One is for allocating stack header
  - The other is for allocating stack body

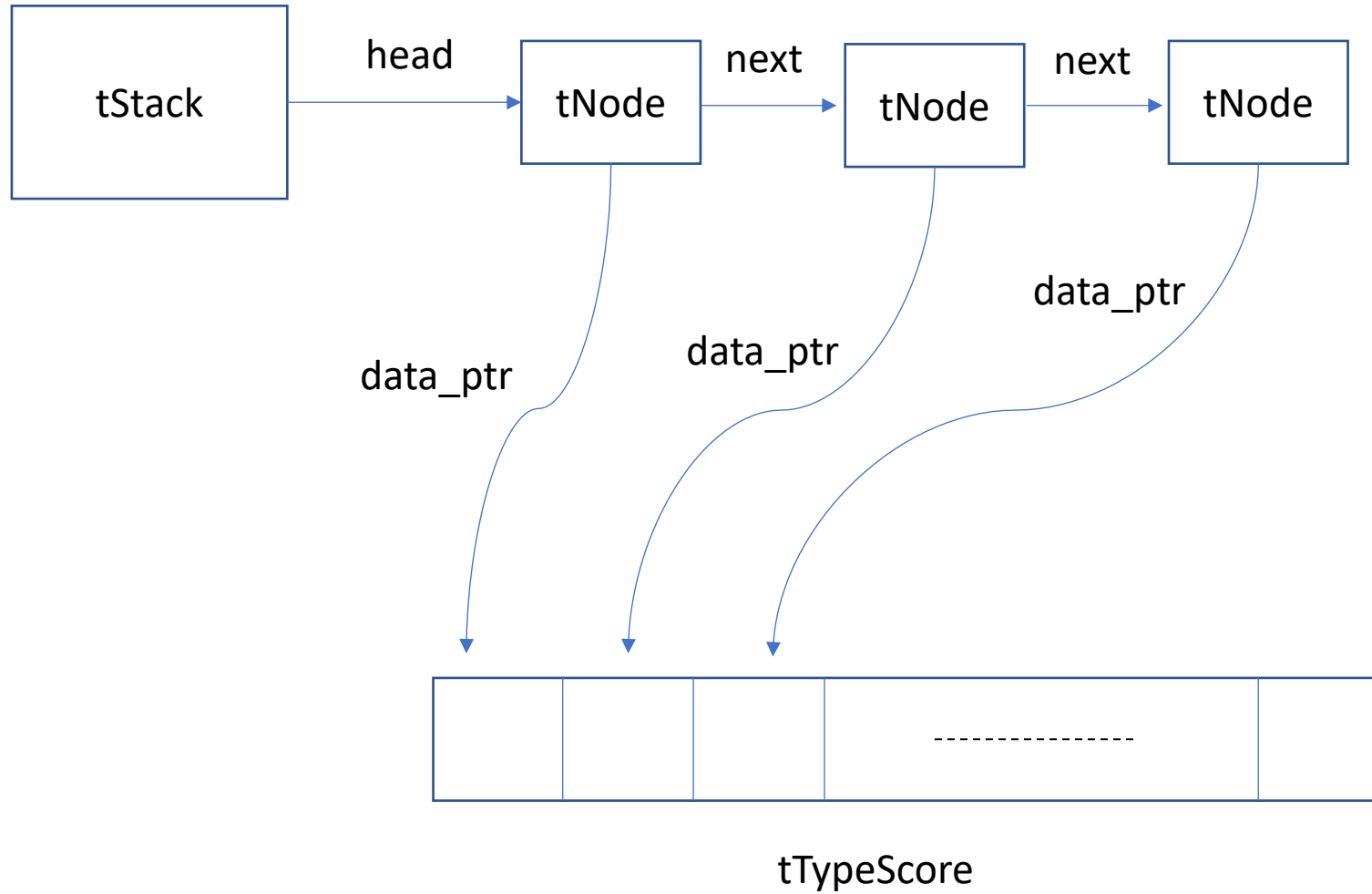- See the attachments

main.c   stack.h   stack.c   space.c   space.h

tStack → head → tNode → next → tNode → next → tNode

data_ptr

data_ptr

data_ptr

tTypeScore

```
1   (1) push or (2) pop a item to/from stack: 2
2     [Error] handlePopOperation(): nothing in stack
3
4      printStackContent(): stack items ->
5
6   (1) push or (2) pop a item to/from stack: 1
7     handlePushOperation(): enter score value: 1
8        getScoreSpace(): giving space numbered 0
9
10     printStackContent(): stack items -> 1(0)
11
12  (1) push or (2) pop a item to/from stack: 1
13    handlePushOperation(): enter score value: 2
14       getScoreSpace(): giving space numbered 1
15
16     printStackContent(): stack items -> 2(1) 1(0)
17
18  (1) push or (2) pop a item to/from stack: 1
19    handlePushOperation(): enter score value: 3
20       getScoreSpace(): giving space numbered 2
21
22     printStackContent(): stack items -> 3(2) 2(1) 1(0)
23
24  (1) push or (2) pop a item to/from stack: 1
25    handlePushOperation(): enter score value: 4
26       getScoreSpace(): giving space numbered 3
27
28     printStackContent(): stack items -> 4(3) 3(2) 2(1) 1(0)


30  (1) push or (2) pop a item to/from stack: 1
31    handlePushOperation(): enter score value: 5
32       getScoreSpace(): giving space numbered 4
33
34     printStackContent(): stack items -> 5(4) 4(3) 3(2) 2(1) 1(0)
35
36  (1) push or (2) pop a item to/from stack: 1
37    [Error] handlePushOperation(): space full
38
39     printStackContent(): stack items -> 5(4) 4(3) 3(2) 2(1) 1(0)
40
41  (1) push or (2) pop a item to/from stack: 6
42
43     printStackContent(): stack items -> 5(4) 4(3) 3(2) 2(1) 1(0)
44
45  (1) push or (2) pop a item to/from stack: 2
46    handlePopOperation(): poped value: 5
47       returnScoreSpace(): return space numbered 4
48
49     printStackContent(): stack items -> 4(3) 3(2) 2(1) 1(0)
50
51  (1) push or (2) pop a item to/from stack: 2
52    handlePopOperation(): poped value: 4
53       returnScoreSpace(): return space numbered 3
54
55     printStackContent(): stack items -> 3(2) 2(1) 1(0)
```

```
57   (1) push or (2) pop a item to/from stack: 2
58     handlePopOperation(): poped value: 3
59        returnScoreSpace(): return space numbered 2
60
61     printStackContent(): stack items -> 2(1) 1(0)
62
63   (1) push or (2) pop a item to/from stack: 1
64     handlePushOperation(): enter score value: 2
65        getScoreSpace(): giving space numbered 2
66
67     printStackContent(): stack items -> 2(2) 2(1) 1(0)
68   ^C
69   (1) push or (2) pop a item to/from stack: 2
70     handlePopOperation(): poped value: 2
71        returnScoreSpace(): return space numbered 2
72
73     printStackContent(): stack items -> 2(1) 1(0)
74
75   (1) push or (2) pop a item to/from stack: 2
76     handlePopOperation(): poped value: 2
77        returnScoreSpace(): return space numbered 1
78
79     printStackContent(): stack items -> 1(0)
80
81   (1) push or (2) pop a item to/from stack: 2
82     handlePopOperation(): poped value: 1
83        returnScoreSpace(): return space numbered 0
84
85     printStackContent(): stack items ->
```

```
87   (1) push or (2) pop a item to/from stack: 2
88     [Error] handlePopOperation(): nothing in stack
89
90     printStackContent(): stack items ->
91
92   (1) push or (2) pop a item to/from stack: ^C
```

11