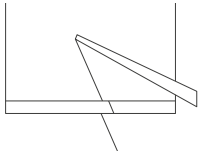
Bring ideas to life
VIA University College



C Memory Management and Pointers

ESW1

At the end of this session, you should

- Understand memory management
- Be able to explain the purpose of a `struct` and how to declare and use it
- Understand the purpose of `typedef`
- Use structs in a Linked List example

Exercises

- Exercise 4.1 – Wait with `my_strdup(...)`

Structures (`struct`) in C

- Structures in C group related data of different types (objects in Java)
- `struct` defines a type that can be used to create variables
- Member access '.' (Like Java) and '->' if pointer access

- A kind of simple class with only public fields and without methods

```
struct student {
        int student_number;
        char *student_name;
};

int main () {
        struct student me;
        struct student you;

        me.student_number = 123;
        me.student_name = strdup("Julia");

        you.student_number = 247;
        you.student_name = strdup("John");
        …
```

Dynamic memory allocation (stdlib.h)

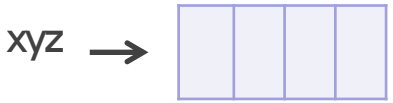
General/simple form of memory allocation:

```
void* malloc (size_t noOfBytes)
```

```
char* p = malloc(sizeof(char) * 8);



struct my_struct {
   int x, y;
};

struct my_struct* xyz;
xyz = malloc(sizeof(struct my_struct));
```

Allocates an array of 8 chars in memory/heap and makes p point to the first char

Allocates a block of two integers (2 bytes each) as a struct and makes xyz point to it



Freeing Dynamic Memory

In C dynamic memory must be deallocated after use
Remember there is No Garbage Collector

```
void free(void* ptr)
```

```
struct my_struct {
   int x, y;
};

struct my_struct *xyz;
xyz = malloc(sizeof(struct my_struct));

// Use the dynamic struct here

free(xyz);  // Dealocate the memory again after use!!
xyz = NULL; // Just a good habit
```

Dynamic memory allocation (stdlib.h)

All functions for memory allocation/freeing:

```
// allocated memory not cleared
void* malloc (size_t noOfBytes)

// allocated memory set to 0
void* calloc(size_t noOfItems, size_t itemSize)

// try to reallocate memory to new size
void* realloc(void* oldPointer, size_t newSize)

// free the memory ptr points to
Void free(void* ptr)
```

Note: FreeRTOS has it's own functions for memory management

Pointers to structs

We *rarely* transfer structs to functions – prefer pointers to structs instead (call by reference/more efficient)

Functions need the address (&) of a struct variable to be able to change it's elements

Member access through pointer '->' is used when the struct is referenced by a pointer

```
void inputNextStudent (struct student* s)
{
    …
    s->studentNumber = 25;
    s->studentName = strdup(”Roman”);
    …
}

int main(void)
{
    struct student me;

    inputNextStudent(&me);
    …
```

typedef

Makes the code easier to read

```
typedef real_type defined_type
```

```
typedef int int_t; // Declare an integer type

typedef struct linkedListNode {
      int id;
      pLinkedListNode_t next;
} linkedListNode_t;

// Create a variables of the types
int_t myInt;
linkedListNode_t myList;
```

typedef

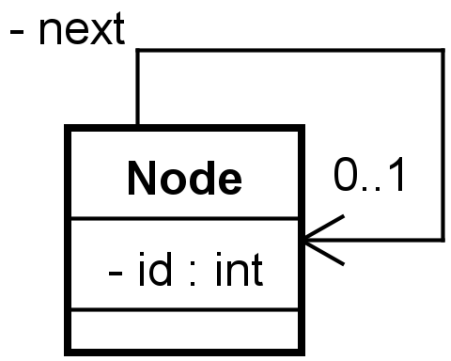
Without `typedef`

```
struct linkedListNode {
        int id;
        struct linkedListNode*  next;
};

int main (void) {
        struct linkedListNode* pTheList;
        pTheList = malloc(sizeof(struct linkedListNode));
        pTheList->id = 12;
        pTheList->next = NULL;
```

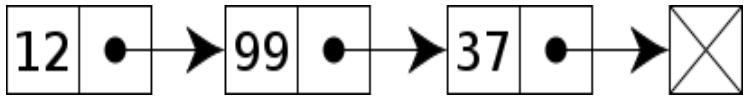
With `typedef`

```
typedef struct linkedListNode* pLinkedListNode_t;

typedef struct linkedListNode {
        int id;
        pLinkedListNode_t next;
} linkedListNode_t;

int main (void) {
        pLinkedListNode_t pTheList;
        pTheList = malloc(sizeof(linkedListNode_t));
        pTheList->id = 12;
        pTheList->next = NULL;
```

Self-referential structures



Linked lists




```
typedef struct linkedListNode* pLinkedListNode_t;
typedef struct linkedListNode {
        int id;
        pLinkedListNode_t next;
} linkedListNode_t;

int main () {
  pLinkedListNode_t pTheList;

  // Create first node
  pTheList = malloc(sizeof(linkedListNode_t));

  // Add data to first node
  pTheList->ID = 12;
  // set pointer to next node to NULL
  pTheList->next = NULL;
```

Exercises

- Design and implement the my_strdup(...) from Exercise 4.1