SC1007
Data Structures and Algorithms

Dynamic Data Structure

&

Linked Lists

College of Engineering

School of Computer Science and Engineering

Dr. Loke Yuan Ren

yrloke@ntu.edu.sg

N4-02B-69A

Overview

- Definition of Pointers and Structures
- Static Data Structure and Dynamic Data Structure
- Computer Memory Layouts
- Memory Allocation
- Memory Deallocation
- Examples and Common Mistakes

- Concepts of Linked Lists

C Programming - Quick Recap

Basic C Programming:

1. **Read Input: scanf()**
2. **Write Output: printf()**
3. **Arithmetic: +,-,*,/,%**
4. **Logic: &&, ||, !, ==, !=, >,<,>=,<= etc.**
5. **Control Structure:**
    1. **Sequence Structure**
    2. **Selection Structure: if… else…, switch and break, goto, ? :**
    3. **Repetition Structure: while, do… while, for loop**

1. **Function**
2. **Pointer – * and &**
3. **Array**
4. **Character String**
5. **Structure**
6. **Recursion**

Static Data Structure and Dynamic Data Structure

Static Data Structure: the allocated memory size is fixed at compile time. You are not able to change the size while you are running it.

Built-in Data Types

Scalar variables	char, int, long, float, double
Array	char name[64]; int a[8][8];

Derived Data Type

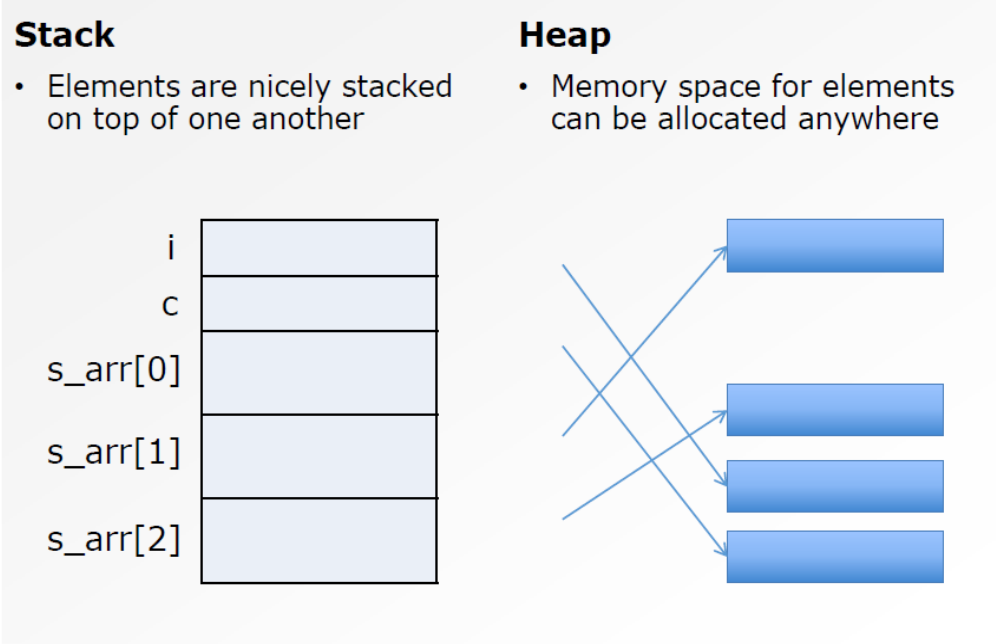
C Structs	**typedef struct node {** **int age;** **float height;** **}student_t;** **student_t s1;** **struct node s2;**
	struct node { int age; float height; }student;

Dynamic Data Structure: the memory allocation is done during execution time. You have some ways to change the size during the program is running.

Static Data Structure and Dynamic Data Structure

Static Data Structure: the allocated memory size is fixed at compile time. You are not able to change the size while you are running it.

Dynamic Data Structure: the memory allocation is done during execution time. You have some ways to change the size during the program is running.



Memory Allocation

3 scenario

1. **Known the data size before compile**
2. **Known the data size at the beginning**
3. **Unknown the data size. The size can be increased or decreased over the time while the program is running**

1. **Static Data Allocation (in stack memory)**
2. **Dynamic Data Allocation (in heap memory)**
3. **Dynamic Data with linked list structure**

Dynamic Memory Allocation and Deallocation

```
#include <stdlib.h>
malloc() and free()
```

- `malloc()` takes an argument of the number of bytes to be allocated and returns a **`void`** pointer `to the allocated memory.`
- `sizeof()` can be used to determine the size of a structure in bytes

```
Eg. int *array = (int *) malloc(sizeof(int)*5);
```

- `free() deallocates memory but freed pointer is not NULL.`

Extra Information

- malloc() does not “clear” the data in the memory.
- If you would like to initialize the elements as zero,
  1. Write a loop to initialize them to zero
  2. Use calloc()

  Eg. int *array = (int *) calloc(5,sizeof(int));
- free() is still required to deallocate memory
- realloc() allows user change the size of the allocated memory.

```c
#include <stdio.h>
#include <stdlib.h>                                     //include library for malloc() and free()

int main(void)
{
      int i;
      double* item;                                     //declare two pointers
      char* string;
      item = (double *) malloc(10*sizeof(double));      //dynamically memory allocation for 10 elements each
      string = (char *) malloc(10*sizeof(char));

      for(i=0;i<10;i++)                                 //Read 10 floating numbers
      {
            scanf("%lf",&item[i]);
      }
      scanf("%*c");                                     //skip the last '\n'

      i=0;                                              //Read 9 character + null character to stop
      char *stringP=string;
      while(i++<9)
            scanf("%c",stringP++);
      *stringP='\0';
      printf("%s\n",string);                            //Print the string

      double* itemP=item;                               //Print the numbers in item
      for(i=0;i<10;i++,itemP++)
            printf("%.2lf ",*itemP);
      printf("\n");

      free(item);                                       //free the allocated memory
      free(string);
      return 0;
}
```

Memory Allocation

3 scenario

1. **Known the data size before compile**
2. **Known the data size at the beginning**
3. **Unknown the data size. The size can be increased or decreased over the time while the program is running**

1. **Static Data Allocation (in stack memory)**
2. **Dynamic Data Allocation (in heap memory)**
3. **Dynamic Data with linked list structure**

Memory Allocation

3 scenario

1. **Known the data size before compile**
2. **Known the data size at the beginning**
3. **Unknown the data size. The size can be increased or decreased over the time while the program is running**

1. **Static Data Allocation (in stack memory)**
2. **Dynamic Data Allocation (in heap memory)**
3. **Dynamic Data with linked list structure**

Linked List

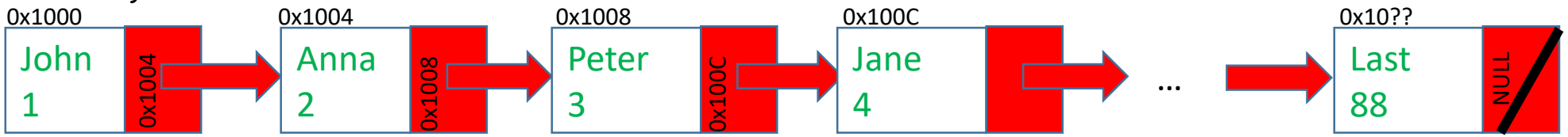
Memory Address	Name	Matric No
0x1000	John	0001
0x1004	Anna	0002
0x1008	Peter	0003
0x100C	Jane	0004
	…	…

- **Structure**: a collection of variables with different types:

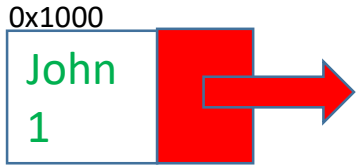
```
struct student{
        char Name[15];
        int matricNo;
}
```

- **Self-referential structure:** a pointer member that points to a structure of the same structure type

```
struct node{
        char Name[15];
        int matricNo;
        struct node *nextPtr;   //link
}
```

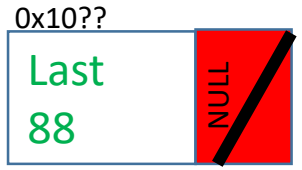


Linked List




```
struct node{
          char Name[15];
          int matricNo;
          struct node *nextPtr;  //link
}
```

1. **Each node contains data and link**
2. **The link contains the address of next node**
3. **If user knows the address of first node, the next node can be found from the link.**
4. **The link of the last node is a NULL pointer**



5. **The example is known as singly-linked list**
   - **There is only ONE link in the node**

Summary

- Static Structure Definition

```
typedef struct node {
        int age;
        float height;
   }student_t;
student_t s1;
struct node s2;
```

- Dynamic Memory Allocation/Deallocation

```
#include <stdlib.h>
item = (double *) malloc(10*sizeof(double));
free(item);
```

- Concepts of Linked Lists

```
struct node{
        char Name[15];
        int matricNo;
        struct node *nextPtr;  //link
    }
```

Overview of Next Lecture

1. **What is the linked list?**
2. **How to create a linked list?**
3. **How to use the linked list?**
4. **Why do you need a linked list?**

3 scenario

1. **Known the data size before compile**
2. **Known the data size at the beginning**
3. **Unknown the data size. The size can be increased or decreased over the time while the program is running**

1. **Static Data Allocation (in stack memory)**
2. **Dynamic Data Allocation (in heap memory)**
3. **Dynamic Data with linked list structure**