COMP26120: ALGORITHMS AND IMPERATIVE PROGRAMMING

LECTURE 4:

INFORMATION REPRESENTATION (PART II)

LECTURE OUTLINE

- Information representation (cont.)
  - Arrays;
  - Arrays and pointers;
  - Multidimensional arrays;
  - Arrays of pointers;
  - Structures;
  - Linked lists;

INFORMATION REPRESENTATION ARRAYS

- The basic types are used to build a more complex data.
- The simplest example is to group together several variables of a certain data type together and give them the same name. Such data structure is called an array.
- An array (a vector), is a collection of elements of the same type. An array has a fixed size. In memory we can allocate in advance the space needed to store an array.
- Question: Which commands would you use to allocate the space in memory for an array?

INFORMATION REPRESENTATION ARRAYS

- To declare an array, we need to define its type (int, float, char), its name (identifier) and the size of the array.
- Example:

```
int a[5];
```

- Arrays in C are zero-based, i.e. the indices start from 0. In the previous example valid elements are a[0],a[1],a[2],a[3],a[4].
- C does not provide bounds checking, i.e. it does not prevent you from accessing elements beyond the ends of your array, elements that do not actually exist – this may be the source of serious programming errors.
- Elements of an array are referenced by their index and can be used as any other variables.

INFORMATION REPRESENTATION ARRAYS

- Example:

```
a[1] = 10*a[2];
if(a[3]==0 && a[4]) a[4]=a[4]+x;
```

- It is a good programming practice to define the array size as a constant in a pre-processing command **#define.**
- Example:

```
#define SECOND_YEAR_SIZE 180;
int student_ID [SECOND_YEAR_SIZE];
float comp26120_score [SECOND_YEAR_SIZE];
```

INFORMATION REPRESENTATION INITIALISATION OF ARRAYS

- The array declaration only reserves the space of necessary size in the memory to store the array. However, initially, the values of the array elements are not set.
- This can be done at the same time the array is declared, or later in the program.
- Example:

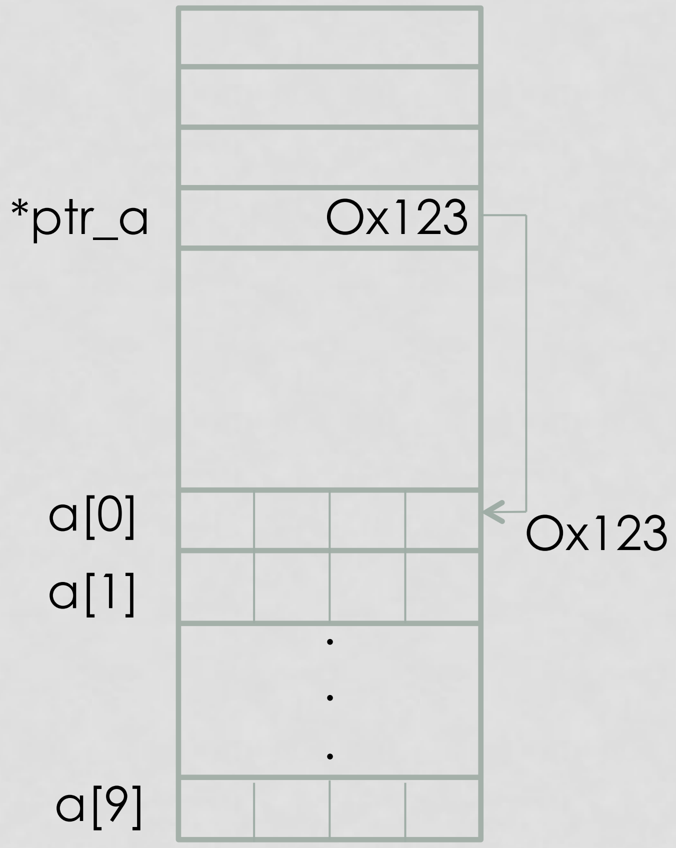
```
float comp26120_score [5]={85.3,78.2,90.4,75.7,68.9};
int student_ID[ ]={5502,3478,2234,2289,6542};
```

- Processing the arrays is associated with the loops (for,while);

INFORMATION REPRESENTATION
ARRAYS AND POINTERS

- Remember, a pointer stores the address of a memory location containing useful data.
- An array is a set of consecutive memory locations that store data of certain type.
- Each of the array's elements have an address. This allows us to set pointers to point at individual elements in the array.
- Example:

```
int a[10];
int *ptr_a;
ptr_a=&a[0];
```



INFORMATION REPRESENTATION ARRAYS AND POINTERS

- Example (cont.):

```
int a[10];
int *ptr_a;
ptr_a=a;
```

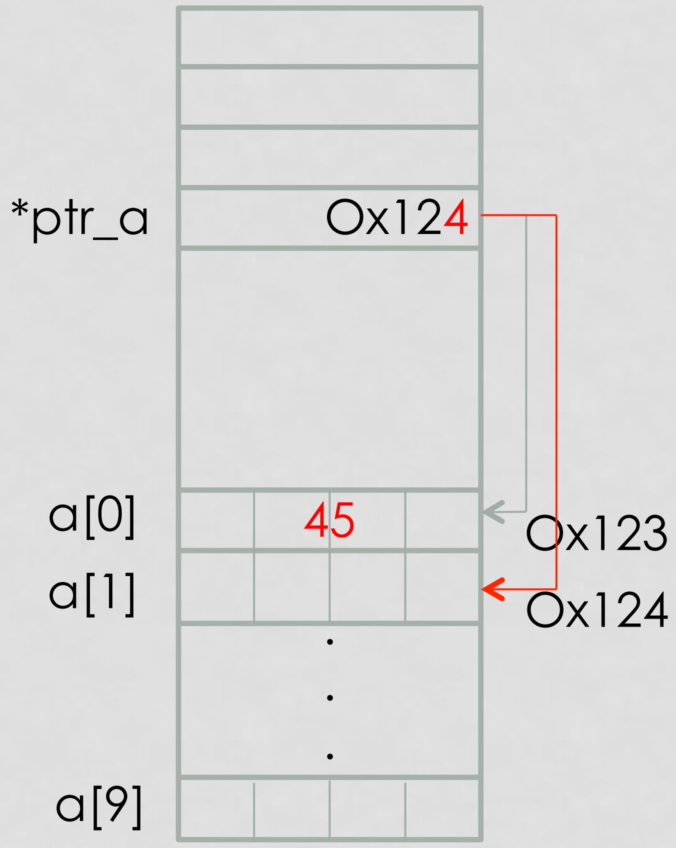
- The command ptr_a=a assigns the address of the first element of the array a to ptr_a .
- This is correct because the name of the array is equivalent to the memory location of its first element, i.e. (a is the same as &a[0]). This applies to any type of an array.

INFORMATION REPRESENTATION ARRAYS AND POINTERS

- There is a relationship between the array indices and pointers.
- <u>Example</u>:

```
int a[10];
int *ptr_a;
ptr_a=a;
*ptr_a=45;
ptr_a++;
```

- Explain what is the effect of the last two statements?



INFORMATION REPRESENTATION ARRAYS AND POINTERS

- Example:

```
int a[10];
*(a+2)=5;
```

- What is the result of the execution of the above statement?
- The array name a returns the address to the first element (a is the same as &a[0]).
- +2 is the offset, thus we want to access the element a[2].
- *(···) dereferences the address a+2 in order to access its contents.
- The same effect can be achieved by a[2]=5.
- In the early days using pointer arithmetic instead of the subscripts led to a more efficient and faster code.
- This is generally not the case with modern compilers (pointer arithmetic and subscripts tend to be equivalent now).
- In many cases using subscripts instead of pointer arithmetic can be preferable from a readability point of view.

INFORMATION REPRESENTATION MULTIDIMENSIONAL ARRAYS

- Multidimensional arrays in C are represented as arrays of arrays.
- Example:

```
int a[10]; /* one-dimensional array */
float b[3][2]; /*two-dimensional array */
```

- The elements of multi-dimensional arrays are stored continuously in the memory in a row-wise fashion (the rightmost indices change the fastest).

b[0][0]	b[0][1]	b[1][0]	b[1][1]	b[2][0]	b[2][1]

- Pointers can be used as with one-dimensional arrays to access the elements, but the expressions are more intricate.
- Write an equivalent pointer-based statement that does b[i][j]=7.

INFORMATION REPRESENTATION
ARRAYS OF POINTERS

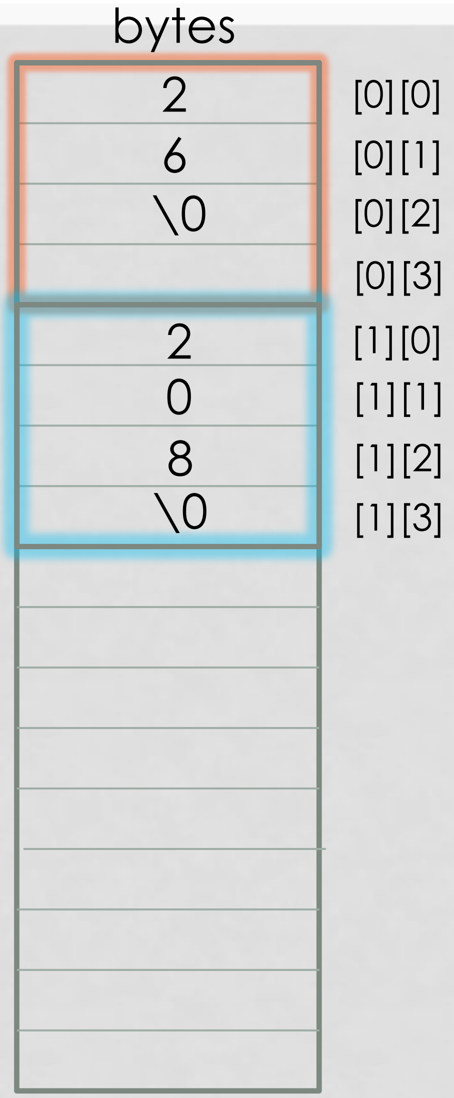
- In C it is possible to define an array of any type (so, an array of pointers is perfectly valid).
- Example:

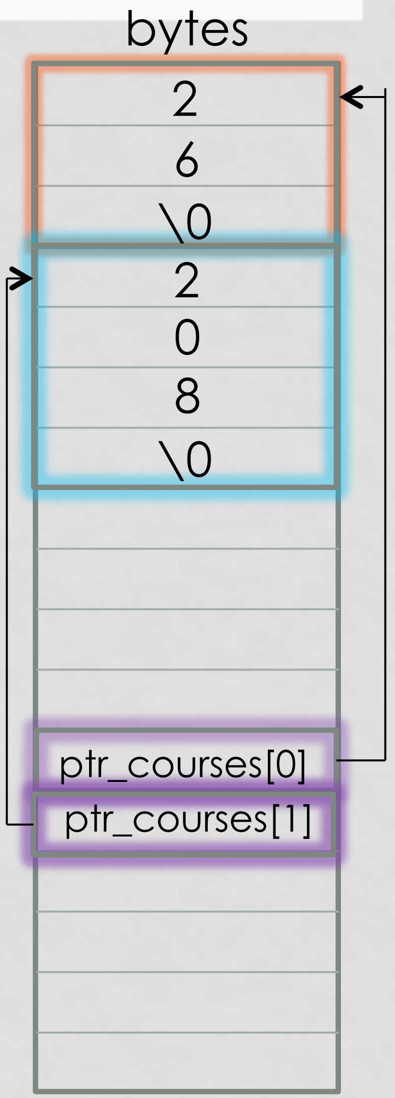
```
int *ptr_a[10];
char *ptr_c[5];
```

- In some cases multidimensional arrays can be represented as arrays of pointers.
- Example:

```
char courses[2][4];
char *ptr_courses[2];
```

- Homework: Write a C code that would read from the screen the course numbers and initialise the array of pointers.
- Homework: Write a C code that implements a simple sorting algorithm (e.g. bubble sort) that will sort the array of strings in ascending order.





INFORMATION REPRESENTATION STRUCTURES

- User-defined structures are the collections of one or more variables, possibly of different types.
- The main application is in organising complicated data where all related variables can be put under one name to form a unit. Think of an example?
- To work with a structure we need to:
  - **Define a structure**: Structure tags, member variables, structure names and initialisation.
  - **Use a structure**: The field operator "."
  - **Pointers to structures**: The "->" operator.
  - **typedef**: A facility for creating new names for data types.

INFORMATION REPRESENTATION
DEFINING A STRUCTURE

- Example: Suppose that we need to register some personal data about a tutor and his tutorial group (e.g. the name, the age, the height, and the home address).

```
char tutor_name[20];
char tutor_address[32];
unsigned int tutor_age;
float tutor_height;
char student1_name[20];
char student1_address[32];
unsigned int student1_age;
float student1_height;
…
char student5_name[20];
…
float student5_height;
```

```
struct
   {
   char name[20];
   char address[32];
   unsigned int age;
   float height;
   } tutor, student[5];
```

INFORMATION REPRESENTATION STRUCTURE TAGGING

- A tag (name) can be associated to each structure definition, which helps when multiple instances of that structure are initialised in the program.
- Example:

```
struct person
{
    char name[20];
    char address[32];
    unsigned int age;
    float height;
};
…
struct person tutor;
struct person student[5];
struct person second_year[200];
```

INFORMATION REPRESENTATION
STRUCTURE INITIALISATION

- Structures are initialised in a similar way as the arrays.
- Example:

```
struct person
{
    char name[20];
    char address[32];
    unsigned int age;
    float height;
};
struct person tutor={“John Stevens”,”5 Hazel Drive, Manchester M20 7BD”, 36, 182.0};
```

- Homework: How would you initialise the array:

```
struct person student[3]=???
```

INFORMATION REPRESENTATION ACCESSING THE VARIABLES

- Individual variables (fields) in a structure can be accessed using the structure field operator "."
- Example:

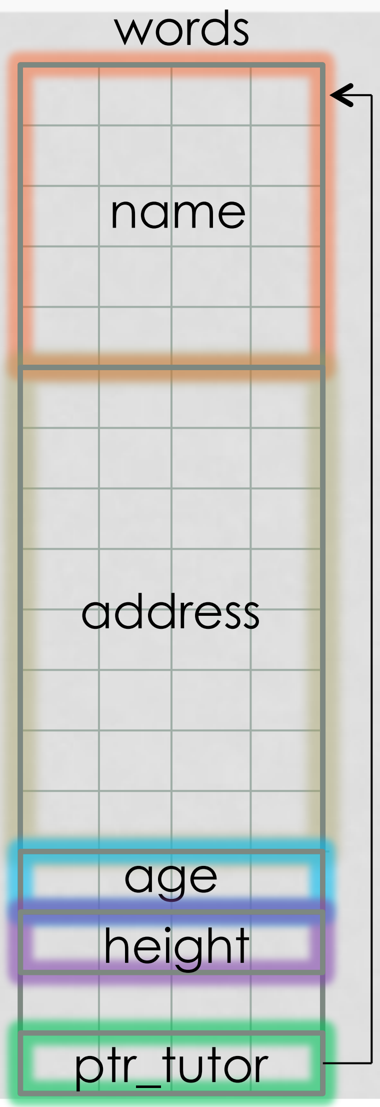
```
struct person
{
    char name[20];
    char address[32];
    unsigned int age;
    float height;
} tutor={"John Stevens","5 Hazel Drive, Manchester M207BD",
36, 182.0};
tutor.age=37;
printf("%6.1f",tutor.height);
```

INFORMATION REPRESENTATION
POINTERS AND STRUCTURES

- We already said that pointers can point to anything, so why not pointing to structures?

Example:

```
        struct person
        {
            char name[20];
            char address[32];
            unsigned int age;
            float height;
        } tutor={"John Stevens",
        "5 Hazel Drive, Manchester
        M207BD", 36, 182.0};
        struct person *ptr_tutor;
        ptr_tutor=&tutor;
```



INFORMATION REPRESENTATION POINTERS AND STRUCTURES

- How to access an individual variable in a structure using a pointer (e.g. we want to change tutor's height in the structure tutor)?
- We need to use the "->" operator.

```
ptr_tutor->height=180.6;
```

- The important thing to notice is that we do not need to dereference the pointer ptr_tutor, as -> operates automatically on the object pointed to by ptr_tutor, i.e. the structure person.
- Two or more structures can be nested.
- Example:

```
struct pay_details
{
   long bank_account_number;
   float NI_number;
   float net_pay;
};
```

```
struct person
{
   char name[20];
   char address[32];
   unsigned int age;
   float height;
   struct pay_details salary;
};
```

INFORMATION REPRESENTATION STRUCTURE NAMES

- We can assign an alias to any simple data type, using the keyword **typedef.**

```
typedef double double_precision;
typedef long double quad_precision;
double_precision a,b;
quad_precision c,d;
```

- The same can be done with structures:

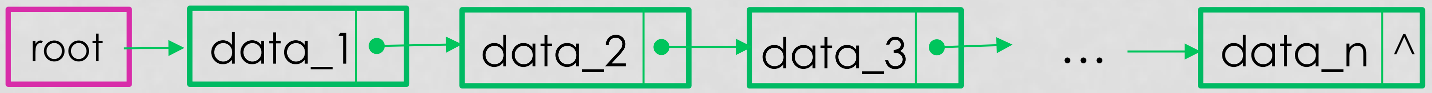
```
typedef struct
{
   char name[20];
   char address[32];
   unsigned int age;
   float height;
} person;
person tutor, student[5], second_year[200];
```

INFORMATION REPRESENTATION LINKED LISTS

- C does not provide any dynamic data structures, that can change their structure, size, or reordering the arrangement of its elements.
- Such structures are, however, essential in some problems (can you think of an example?), or make programming much easier and flexible.
- One of such structures is a linked list.
- It consists of a an array of some useful data (a simple variable, or an object), chained together. For each element in the list we know its predecessor and successor. Also, the first and the last element in the list are uniquely determined.
- Linked lists are essential in algorithms involving trees, graphs, hash-tables, which you will see in your labs towards the end of this semester and in the second semester, so pay attention.

INFORMATION REPRESENTATION LINKED LISTS

- Schematically, we can present a linked list as follows:



simple (single chained) linked list

- Example: Instead of an array struct person student[5], use the definition of a linked list to represent a (potentially variable in size) data structure for a tutorial group.

```
typedef struct person
{
  char name[20];
  char address[32];
  unsigned int age;
  float height;
  struct person *next;
} student;
```

INFORMATION REPRESENTATION LINKED LISTS

- <u>Example</u>: Create a linked list consisting of two elements.

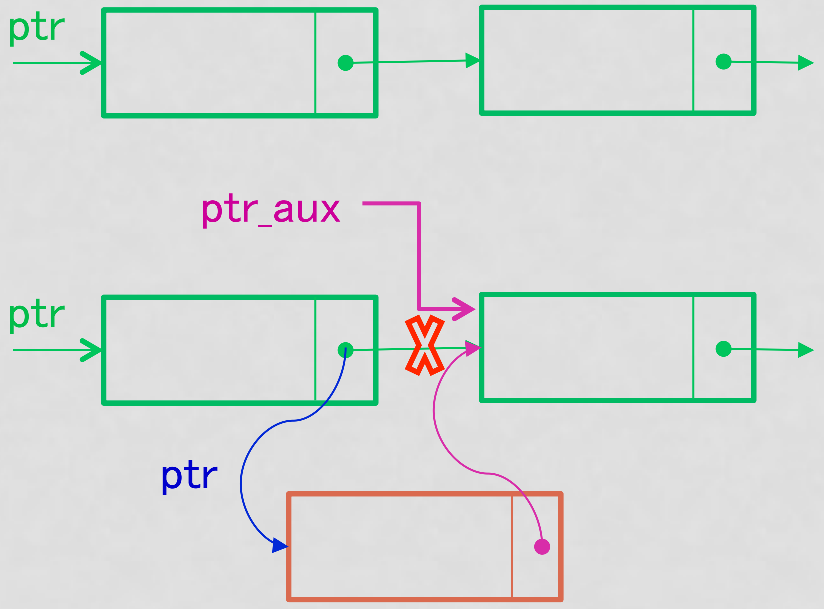
```
#include <stdio.h>
main()
char std_name[20],
std_address[32];
unsigned int std_age;
float std_height;
typedef struct person
{
   char name[20];
   char address[32];
   unsigned int age;
   float height;
   struct person *next;
} student;
person *root_ptr, *curr_ptr;
```

```
/ * Chain the first element to a root_ptr */
root_ptr=(*student)malloc(sizeof(student));
if (root_ptr == NULL) error_message();
/* Insert the data into the first element */
get_student_data (&std_name,&std_address,
                  &std_age,&std_height);
root_ptr->name=&std_name;
root_ptr->address=&std_address;
root_ptr->age=std_age;
root_ptr->height=std_height;
curr_ptr=root_ptr;
/ * Chain the second element to the first * /
curr_ptr->next=(*student)malloc(sizeof(student));
if(curr_ptr->next == NULL) error_message();
curr_ptr=curr_ptr->next;
/* Insert the data to second element */
get_student_data (&std_name,&std_address,
                  &std_age,&std_height);
curr_ptr->name=&std_name;
curr_ptr->address=&std_address;
curr_ptr->age=std_age;
curr_ptr->height=std_height;
curr_ptr->next=NULL;
```

INFORMATION REPRESENTATION LINKED LISTS

- Example: Inserting a new element to a linked list.

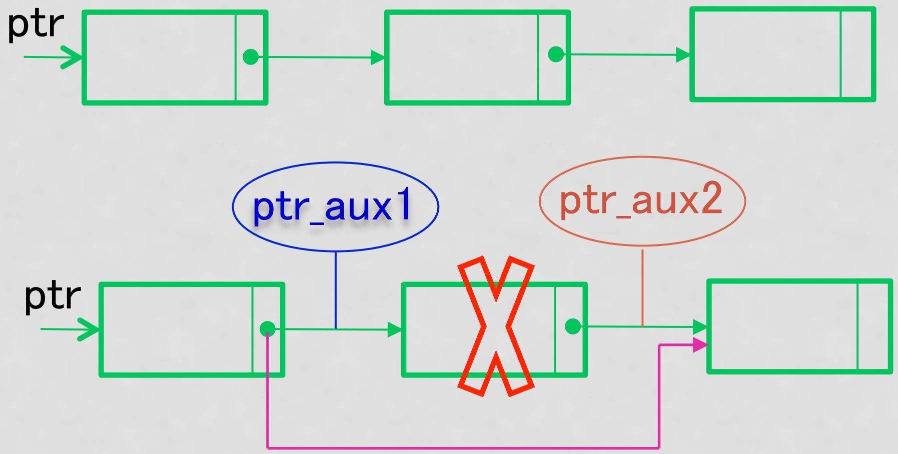
```
ptr_aux=ptr->next;
ptr->next=(*<type>)malloc(sizeof(<type>);
ptr=ptr->next;
<input the useful data into [    ] >
ptr->next=ptr_aux;
```



INFORMATION REPRESENTATION
LINKED LISTS

- Example: Deleting an element from a linked list.

```
ptr_aux1=ptr->next;
ptr_aux2=ptr_aux1->next;
ptr->next=ptr_aux2;
free(ptr_aux1);
```


ptr
ptr_aux1
ptr_aux2
ptr