C Programming

- Data structures and Conditionals

Dr. Hyun Lim
h.lim@lboro.ac.uk

Data structure

- Three basic data types
  - The integer (*int*), e.g. int a= 1;
  - The character (*char*), e.g. char b='name';
  - The decimal number (*float*), e.g. float c=3.1415;

- Software has to deal with **related data**, such as, module marks, customer information including name, address, phone, etc.

- There is an **efficient** way of dealing with such related data items

  : They can be **grouped together** because they are **related** in some way

Array

- If the structure is an ***array***, all the component data items are of **the same type**, such as,

  int my_birth_day[3]; // All the 3 items are integer numbers.

my_birth_day	6	4	1997

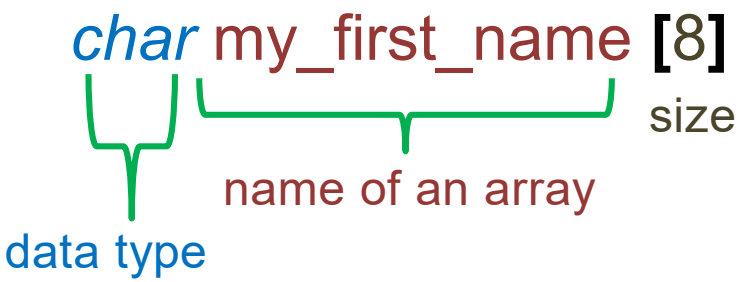
  char my_first_name[8] // All the 8 items are characters.

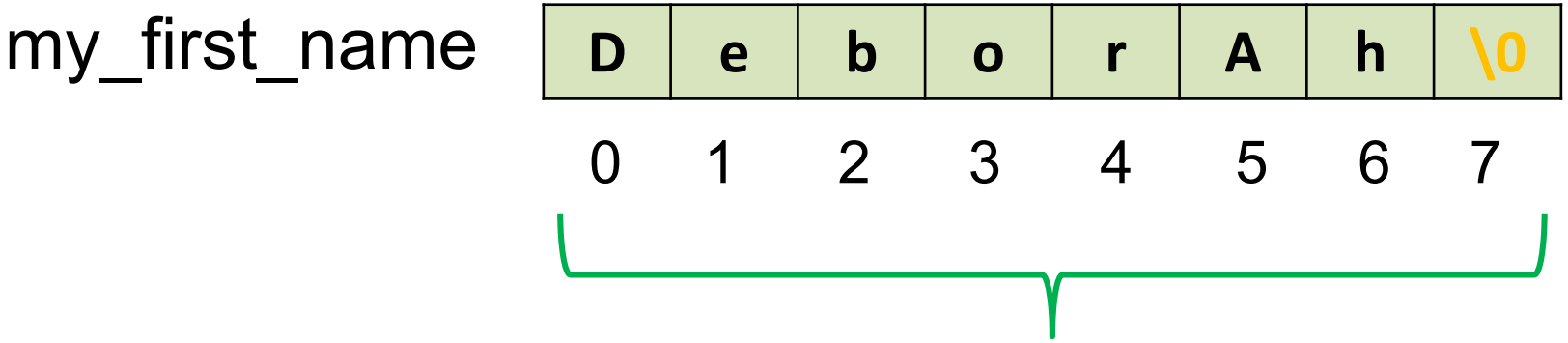
my_first_name	D	e	b	o	r	A	h	\0

- Far more elaborate structures (complex data types) are available, such as ***struct*** (in *C*), ***class*** (in *C++* and *Python*) that we will learn later in this module

Array - Indices

Example





- The storage boxes are ***numbered (indexed)***, *e.g.* from 0 to 7
- First ***element*** is always numbered **0**

Character Array

char name[ ] = "John";

What this array looks like in memory is the following:

Note: an additional character is needed to store the ***null character*** '\0' that indicates **the end of the string**.

char full_name[10] = "John";

where 5 array elements are currently unused.

```
char name[20];
char title[30];
```

Declaring & Calling Array Variables

```
// The same as any other variable, such as
 int birth [3];    // declaration: (data type) (array name)[(size)]
  birth[0]=6; birth[1]=4; birth[2]=1997;
     // assigning values, DD/MM/Year

int k=1;
printf("Month is %d \n", birth[k]);
                        // calling the vales stored in birth[1]
k++;  // k=k+1
printf("Year is %d \n\n\n", birth[k]);   // be calculated
```

Why do we need an *Array*?

- Storing 1000 numbers using the previous approach would require 1000 variables
- Processing the data would require 1000 times

```
int my_int000=12;
int my_int001=5;
int my_int002=4;
int my_int003=5879;
…
int my_int999=46;
// 1000 different (variable) names
// Unstructured, no logic
```

VS

```
int my_int[1000];
my_int[0]=12;
my_int[1]=5;
…..
my_int[999]=46;
// only one array name
// Structured way of thinking
```

Program compactness

- ***Array*** - data repetition, whilst ***loop*** - statement repetition
- Loops, and arrays give program **compactness**
- However, arrays do **not reduce size of data** in the memory area.

Exercise

```
#include <stdio.h>
#include <string.h>      // to use strcpy()

 int main(void)
 {
    int nr=1;
    char name[20]="John";     // char array [ (long enough length) ]

    printf("\n\n\n Display book information \n");       // \n means 'next line'
    printf("No.[%d] \n", nr);           //%i integer number
    printf("Author: %s \n", name);   //%s character string

    nr++; // nr=nr+1;
    name= "Gargamel"; // illegal, it does not work.   cf. It is all right in Python
    strcpy(name, "Gargamel");
    printf("\n\n\n");
    printf("No.[%d] \n", nr);           //%i integer number
    printf("Author: %s \n", name);   //%s character string
                    // getchar();  // optional
 return 0;
 }
```

Exercise

```
Book information

No.[1]
Author  : John
Title   : Application development in C
Page no.: 189

No.[2]
Author  : Saint-Exupery
Title   : The Little Prince
Page no.: 91
```

```
// Answer
int number=1;
char name[20]="John";
char title[50]="Application development in C";
int page=189;

printf("Display book information\n");
printf("\nNo.[%d]\n", number);
printf("Author: %s \n", name);
printf("Title: %s \n", title);
printf("Page number: %d\n", page);

number++;
// .....
// getchar();
```

Loop - *while*

Exercise: Add 2 while i is smaller than the maximum number.

```
int i = 0;
int max_nr = 20;
while (i < max_nr)
 {
   printf("i = %d \n", i);
   i=i+2;
 }
```

- A ***loop*** can reduce the size of a code.
- The code can be expressed more efficiently.

Loop - *for*

- Single statement initialising a control variable, testing, repeating and altering the control variable

```
for (int i=0; i<10; i++)
    printf("i = %d \n", i);
```

1. Before the loop starts, the assignment i=0 is executed
2. Next, the ***condition*** i < 10 is evaluated whether it is ***true*** or ***false***
3. If ***false***, the loop exits
4. If ***true***, the body of the loop is executed, followed by **increasing the control variable,** i.e. i++

Loop - *for*

- Can use compound statement for the body of the loop (i.e. between { and })

```
for (int i = 0; i <5; i++)
    {
      printf("i*10 = %d \n", i*10);
      printf("i*100 = %d \n\n", i*100);
     }
```

Loop & Conditionals
- *while* & *if, else if, else*


```
int index= 1;                  // User's choice options, i.e. 1, 2, or 3
int number = 20;               // How many books have you put?
const int max_nr = 50;     // How many books can your library store?

while (1)   // infinity loop as 1 means true
 {
  if (index == 1 && number < max_nr)
      { printf("Insert your book information. \n");
                  getchar();
        number++;     // increase input number
       }
  else if (index == 1 && number >= max_nr)  // more than the max. nr.
       {  printf("You cannot put any more item.\n");
          index++;         getchar( );
        }
   else if (index == 2)    // display the data set on the screen
       {  printf("Show the details (author/ title/ page number) ....\n");
          index++;          getchar();
       }
   else if (index == 3)
        { printf("Finish the program! \n\n");
          getchar();
          break;
         }
  else      // In case of a wrong index number, e.g. 4, 5,....
        printf("Mistake! Please start it again. \n");
}
```