C Developer

Complex Data Types

Course Objectives

- ✓ Study the implementation of arrays and structures
- ✓ Know how to manipulate strings

Course Plan

1. Arrays
2. Strings
3. Structures

1. Arrays

1. Arrays

One-dimensional arrays

- Declaration syntax:

```
type tab[dim];
```

  - **type** is the type of stored objects: **int**, **float**, **char**, *etc*.
  - **dim** is the number of slots

```
int myArray[12];
```

- Static allocation: in C89, **dim** must be a **const**

- Dynamic allocation: in C99, **dim** can be a variable

1. Arrays

One-dimensional arrays

- You can access a value using "**[…]**"

- The values are from the first **[0]** to the last **[dim-1]**

- Each element of the array behaves like a variable, and this at all points of view: assignment, use, reading

```
int i = myArray[0];
int j = myArray[n-1];
myArray[3] = 8;
```

1. Arrays

One-dimensional arrays

```c
#include <stdio.h>

int main()
{
    int tab[5];
    int i, min;
    for(i = 0; i < 5; i++) {
        printf("Value %d: ", i+1);
        scanf("%d", &tab[i]);
    }
    min = tab[0];
    for(i = 1; i < 5; i++) {
        if(tab[i] < min) min = tab[i];
    }
    printf("min = %d\n", min);
    return 0;
}
```

```
Value 1: 8
Value 2: 3
Value 3: 2
Value 4: 9
Value 5: 4
min = 2
```

1. Arrays

One-dimensional arrays

- You can initialize the array when you declare it

```
int myArray[5] = {1, 3, 5, 7, 11};
```

- You can then omit the dimension of the array, it will be calculated automatically

```
int myArray[] = {1, 3, 5, 7, 11};
```

1. Arrays

One-dimensional arrays

- If at the initialization, less values than the array dimension are set, the last cells of the array will be assigned the **0** value

```
int myArray[10] = {1, 3, 5, 7, 11};
```

1	3	5	7	11	0	0	0	0	0

1. Arrays

One-dimensional arrays

- There is no compiler control over the value of the indices of an array
- For example, if you declare an array of 5 elements and use the 7th one you will not receive any error messages, but you may overwrite other existing variables

```c
#include <stdio.h>

int main()
{
    int myArray[5] = {1, -3, 5};
    for(int i = 0; i < 7; i++) {
        printf("myArray[%d] = %d\n", i, myArray[i]);
    }
    return 0;
}
```

```
myArray[0] = 1
myArray[1] = -3
myArray[2] = 5
myArray[3] = 0
myArray[4] = 0
myArray[5] = 5
myArray[6] = 6422284
```

1. Arrays

Two-dimensional arrays

- Declaration syntax:

```
type tab[dim1][dim2];
```

  - **type** is the type of stored objects: **int**, **float**, **char**, *etc*.
  - **dim1** is the number of lines
  - **dim2** is the number of columns

```
int myArray[8][9];
```

- The value located at the intersection of the i^{th} row and the j^{th} column will be

```
myArray[i-1][j-1]
```

1. Arrays

Two-dimensional arrays

```c
#include <stdio.h>

int main()
{
    int a, i, j;
    int mat[3][3];
    printf("a = ");
    scanf("%d", &a);
    for(i = 0; i < 3; i++) {
        for(j = 0; j < 3; j++) {
            printf("mat[%d][%d] = ", i, j);
            scanf("%d", &mat[i][j]);
            mat[i][j] *= a;
        }
    }
    for(i = 0; i < 3; i++) {
        for(j = 0; j < 3; j++) {
            printf("%d\t", mat[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

```
a = 5
mat[0][0] = 1
mat[0][1] = 2
mat[0][2] = 3
mat[1][0] = 4
mat[1][1] = 5
mat[1][2] = 6
mat[2][0] = 0
mat[2][1] = -1
mat[2][2] = 2
5       10      15
20      25      30
0       -5      10
```

1. Arrays

Two-dimensional arrays

- You can initialize the array when you declare it

```
int mat[3][3] = {{1, 2, 3}, {1, 2, 3}, {1, 2, 3}};
```

- You can then omit only the **first** dimension of the array, it will be calculated automatically

```
int mat[][3] = {{1, 2, 3}, {1, 2, 3}, {1, 2, 3}};
```

- You can set less values than the array dimension

```
int mat[3][3] = {{1, 2, 3}, {2, 3}};
```

1. Arrays

Two-dimensional arrays

```c
#include <stdio.h>

int main()
{
    int mat[3][3] = {{1, 2, 3}, {1, 2, 3}, {1, 2, 3}};
    for(int i = 0; i < 3; i++) {
        for(int j = 0; j < 3; j++) {
            printf("%d\t", mat[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

```
1	2	3
1	2	3
1	2	3
```

1. Arrays

Two-dimensional arrays

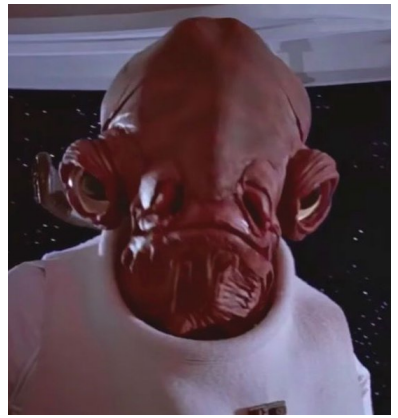
```c
#include <stdio.h>

int main()
{
    int mat[3][3] = {{1, 2, 3}, {2, 3}};
    for(int i = 0; i < 3; i++) {
        for(int j = 0; j < 3; j++) {
            printf("%d\t", mat[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

```
1       2       3
2       3       0
0       0       0
```

1. Arrays

Two-dimensional arrays

```c
#include <stdio.h>

int main()
{
    int mat[][3] = {{1, 2, 3}, {2, 3}};
    for(int i = 0; i < 3; i++) {
        for(int j = 0; j < 3; j++) {
            printf("%d\t", mat[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```



```
1       2       3
2       3       0
0       2       6422284
```

You can now create n-dimensional arrays too!

1. Arrays

Exercise

- Ask an integer to the user

- Calculate the number of occurrences of this integer in a given array

1. Arrays

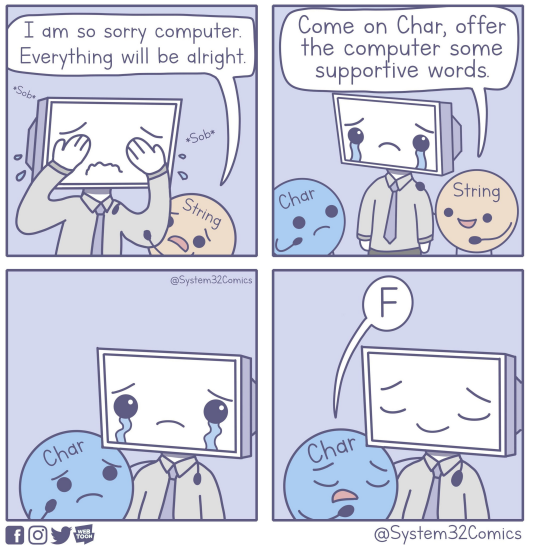
Questions

2. Strings

2. Strings

Representation

- There is no string type in C
- A string will be assimilated to a one-dimensional array of characters
- This array will have an end of string flag: the null character “**\0**”



2. Strings

Representation

- You can therefore declare a string in two ways:

```
char myString[] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

```
char myString[] = "Hello";
```

- We will prefer the second one

2. Strings

Representation

- You can then access, modify and use the different characters of the string, in the same way as for the elements of an array

```c
#include <stdio.h>

int main()
{
    char myString[] = "Hello";
    myString[1] = 'a';
    printf("%c", myString[1]);
    return 0;
}
```

```
a
```

2. Strings

Representation

- A string can only be assigned a value in a global way with the “=” operator during the declaration

```c
#include <stdio.h>

int main()
{
    char myString[] = "Hello";
    myString = "Bye";
    return 0;
}
```

File	Line	Message
		=== Build: Debug in Useless (compiler: GNU GCC Compiler) ===
C:\Users\blab...		In function 'main':
C:\Users\blab...	6	error: assignment to expression with array type
C:\Users\blab...	5	warning: variable 'myString' set but not used [-Wunused-but-set-variable]
		=== Build failed: 1 error(s), 1 warning(s) (0 minute(s), 0 second(s)) ===

2. Strings

Display

- It is possible to display a string character by character, but it would be laborious

- It is easier to use the **printf** function with the **%s** format specifier

```c
#include <stdio.h>

int main()
{
    char myString[10] = "Test";
    printf("%s", myString);
    return 0;
}
```

```
Test
```

2. Strings

Display

- You can also use the **puts** function

- It handles only one string at a time and follows it with a line break

```c
#include <stdio.h>

int main()
{
    char myString[10] = "Test";
    puts(myString);
    return 0;
}
```

```
Test
```

2. Strings

Input

- You can use the **scanf** function with the **%s** format specifier
- The reading of the string will stop as soon as a space or a line break is entered

```c
#include <stdio.h>

int main()
{
    char myString[10];
    printf("String: ");
    scanf("%s", myString);
    printf("Result: %s\n", myString);
    return 0;
}
```

```
String: Benjamin
Result: Benjamin
```

```
String: Benj amin
Result: Benj
```

2. Strings

Input

- To avoid overflows, remember that the size of a string is fixed, you can complete the **%s** format specifier with a value indicating the maximum number of characters to read

```c
#include <stdio.h>

int main()
{
    char myString[10];
    printf("String: ");
    scanf("%3s", myString);
    printf("Result: %s\n", myString);
    return 0;
}
```

```
String: Benjamin
Result: Ben
```

2. Strings

Input

- You can also use the **gets** function to enter a string
- The reading of the string will stop as soon as a line break is entered, the spaces will therefore be integrated into the chain
- Whenever possible, this function should be used, but only one string can be processed at a time

2. Strings

Input

```c
#include <stdio.h>

int main()
{
    char myString[10];
    printf("String: ");
    gets(myString);
    printf("Result: %s\n", myString);
    return 0;
}
```

```
String: Benj amin
Result: Benj amin
```

2. Strings

Operations

#include <string.h>

- **strlen**
  - It takes a string as input
  - It returns an integer indicating the number of characters in the string (without the "**\0**")

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

int main()
{
    char myString[10] = "Hello";
    printf("%d\n", strlen(myString));
    return 0;
}
```

```
5
```

2. Strings

Operations

- **strcpy**
  - It takes two strings and copies the content of the second one (including "**\0**") into the first one

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

int main()
{
    char myString1[10] = "Hello";
    char myString2[10];
    strcpy(myString2, myString1);
    printf("%s\n", myString2);
    return 0;
}
```

```
Hello
```

2. Strings

Operations

- **strncpy**
  - It is like **strcpy**, but it takes also an integer parameter indicating the number of characters to copy

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

int main()
{
    char myString1[10] = "Hello";
    char myString2[10];
    strncpy(myString2, myString1, 4);
    myString2[4] = '\0';
    printf("%s\n", myString2);
    return 0;
}
```

```
Hell
```

2. Strings

Operations

- **strcat**
  - It takes two strings and copies the second one (including "**\0**") at the end of the first one (overwriting "**\0**" of the latter)

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

int main()
{
    char myString1[20] = "Hello ";
    char myString2[10] = "Benjamin";
    strcat(myString1, myString2);
    printf("%s\n", myString1);
    return 0;
}
```

```
Hello Benjamin
```

2. Strings

Operations

- **strncat**
  - It is like **strcat**, but it takes also an integer parameter indicating the number of characters to concatenate

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

int main()
{
    char myString1[20] = "Hello ";
    char myString2[10] = "Benjamin";
    strncat(myString1, myString2, 3);
    printf("%s\n", myString1);
    return 0;
}
```

```
Hello Ben
```

2. Strings

Operations

- **strcmp**
  - It compares two strings and returns a positive integer if the first one is higher than the second one (based on the order of the characters in the ASCII table), 0 if the two strings are identical and a negative integer if the first one is lower than the second one

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

int main()
{
    char myString1[10] = "Hello";
    char myString2[10] = "Hell";
    printf("%d\n", strcmp(myString1, myString2));
    return 0;
}
```

```
1
```

2. Strings

Operations

- **strncmp**
  - It is like **strcmp**, but it takes also an integer parameter indicating the number of characters to compare

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

int main()
{
    char myString1[10] = "Hello";
    char myString2[10] = "Hell";
    printf("%d\n", strncmp(myString1, myString2, 4));
    return 0;
}
```

```
0
```

2. Strings

Exercise

- Ask the user to enter several strings
- Stop the input process when the user ends the sentence with a “.”
- Display the whole text and its size

2. Strings

Questions

3. Structures

3. Structures

Declaration

- C language is not an object-oriented language

- You can use structures:
  - they are like arrays
  - they group together characteristics representing entities
  - they can contain several types

3. Structures

Declaration

- Structure declaration syntax:

```
struct structureName {
    type attribute1;
    type attribute2;
    ...
};
```

3. Structures

Declaration

```c
#include <stdio.h>

int main()
{
    struct user {
        int id;
        char name[55];
        int level;
    };
    return 0;
}
```

3. Structures

Declaration

- Structure variable declaration syntax:

```
struct structureName variableName;
```

```
#include <stdio.h>

int main()
{
    struct user {
        int id;
        char name[55];
        int level;
    };
    struct user John;
    return 0;
}
```

3. Structures

Declaration

- Attributes are accessed using “.”

```
variableName.attribute1
```

- Each attribute behaves like a variable, and this at all points of view: assignment, use, reading

3. Structures

Declaration

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

int main()
{
    struct user {
        int id;
        char name[55];
        int level;
    };
    struct user John;
    John.id = 4;
    strcpy(John.name, "John");
    John.level = 2;
    printf("John\n- ID: %d\n- Name: %s\n- Level: %d\n",
           John.id, John.name, John.level);
    return 0;
}
```

```
John
- ID: 4
- Name: John
- Level: 2
```

3. Structures

Arrays of structures

- You can create an array where each cell is of the type of a previously defined structure

- Your structure is a new data type

- Syntax:

```
struct structureName tab[...];
```

3. Structures

Arrays of structures

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

int main()
{
    struct user {
        int id;
        char name[55];
        int level;
    };
    struct user customers[500];
    customers[25].id = 4;
    strcpy(customers[25].name, "John");
    customers[25].level = 2;
    printf("Customer 25\n- ID: %d\n- Name: %s\n- Level: %d\n",
           customers[25].id, customers[25].name, customers[25].level);
    return 0;
}
```

```
Customer 25
- ID: 4
- Name: John
- Level: 2
```

3. Structures

Nested structures

- Remember:
  - a structure can contain several types of variables
  - a defined structure is a type of variable

- Result:
  - it is possible to nest structures within each other
  - an attribute of a structure can be of the type of another structure already defined

3. Structures

Nested structures

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

int main()
{
    struct date {
        int day, month, year;
    };
    struct user {
        int id;
        char name[55];
        int level;
        struct date birthDate;
    };
    struct user John;
    John.id = 4;
    strcpy(John.name, "John");
    John.level = 2;
    John.birthDate.day = 12;
    John.birthDate.month = 02;
    John.birthDate.year = 1985;
    printf("John\n- ID: %d\n- Name: %s\n- Level: %d\n- Born on: %d/%d/%d\n",
           John.id, John.name, John.level, John.birthDate.month,
           John.birthDate.day, John.birthDate.year);
    return 0;
}
```

```
John
- ID: 4
- Name: John
- Level: 2
- Born on: 2/12/1985
```

3. Structures

Typedef structures

- Quite easier to use

```
John
- ID: 4
```

```c
#include <stdio.h>

int main()
{
    struct user {
        int id;
        char name[55];
        int level;
    };
    struct user John;
    John.id = 4;
    printf("John\n- ID: %d\n", John.id);
    return 0;
}
```

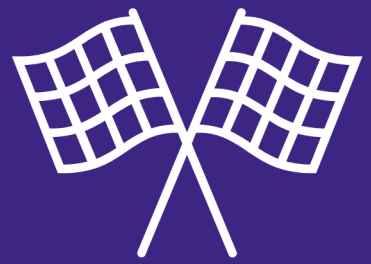
```c
#include <stdio.h>

int main()
{
    typedef struct {
        int id;
        char name[55];
        int level;
    } user;
    user John;
    John.id = 4;
    printf("John\n- ID: %d\n", John.id);
    return 0;
}
```

3. Structures

Questions

C Developer

Complex Data Types



Thank you for your attention

SUPINFO