Introduction to Programming

Week – *7*, Lecture – *1*

Functions in C – Part 1

SAURABH SRIVASTAVA

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

IIT KANPUR

We have been using others' code already

We've been using some code already, that we did not write

- For example, we never really saw the code of the `printf()` and `scanf()` functions

We have been using others' code already

We've been using some code already, that we did not write

- For example, we never really saw the code of the `printf()` and `scanf()` functions

These "functions", as we already know, are "library functions"

We have been using others' code already

We've been using some code already, that we did not write

- For example, we never really saw the code of the `printf()` and `scanf()` functions

These "functions", as we already know, are "library functions"

Different languages provide different varieties of libraries

- While languages like C++ and Java provide a lot of libraries (and hence, a lot of library functions)…
- … the libraries shipped with most C compilers are often fairly basic

We have been using others' code already

We've been using some code already, that we did not write

- For example, we never really saw the code of the `printf()` and `scanf()` functions

These "functions", as we already know, are "library functions"

Different languages provide different varieties of libraries

- While languages like C++ and Java provide a lot of libraries (and hence, a lot of library functions)…
- … the libraries shipped with most C compilers are often fairly basic

Even with a rich set of library functions, we may not get "exactly what we wish"

- For example, while we have a library function to convert a character from lowercase to uppercase…
- … if we want to do it for a complete string, we had to do it on our own

We have been using others' code already

We've been using some code already, that we did not write

- For example, we never really saw the code of the `printf()` and `scanf()` functions

These "functions", as we already know, are "library functions"

Different languages provide different varieties of libraries

- While languages like C++ and Java provide a lot of libraries (and hence, a lot of library functions)…
- … the libraries shipped with most C compilers are often fairly basic

Even with a rich set of library functions, we may not get "exactly what we wish"

- For example, while we have a library function to convert a character from lowercase to uppercase…
- … if we want to do it for a complete string, we had to do it on our own

Thus, almost all programming languages, provide developers the option to write custom functions

- So, if you do not have a library function matching your requirements, you can define your own function

Declaring your own functions

Similar to a variable, a function can have a declaration as well as a definition…

- … or both can be merged together

Declaring your own functions

Similar to a variable, a function can have a declaration as well as a definition…

- … or both can be merged together

A function's declaration includes

- its *name* – which should be unique in within the program

Declaring your own functions

Similar to a variable, a function can have a declaration as well as a definition…

- … or both can be merged together

A function's declaration includes

- its *name* – which should be unique in within the program
- its *arguments* – an "optional" list of the type(s) of input(s) that the function expects, and

Declaring your own functions

Similar to a variable, a function can have a declaration as well as a definition…

- … or both can be merged together

A function's declaration includes

- its *name* – which should be unique in within the program
- its *arguments* – an "optional" list of the type(s) of input(s) that the function expects, and
- its *return type* – the type of the "single" output that this function will provide

Declaring your own functions

Similar to a variable, a function can have a declaration as well as a definition…

- … or both can be merged together

A function's declaration includes

- its *name* – which should be unique in within the program
- its *arguments* – an "optional" list of the type(s) of input(s) that the function expects, and
- its *return type* – the type of the "single" output that this function will provide

If the function does not return any output value, the return type should be declared as `void`

- `void` essentially means "nothing" in languages like C, C++ and Java

Declaring your own functions

Similar to a variable, a function can have a declaration as well as a definition…

- … or both can be merged together

A function's declaration includes

- its *name* – which should be unique in within the program
- its *arguments* – an "optional" list of the type(s) of input(s) that the function expects, and
- its *return type* – the type of the "single" output that this function will provide

If the function does not return any output value, the return type should be declared as `void`

- `void` essentially means "nothing" in languages like C, C++ and Java

An example declaration:

```
char* toUppercaseString(char*, int);
```

Declaring your own functions

Similar to a variable, a function can have a declaration as well as a definition…

- … or both can be merged together

A function's declaration includes

- its *name* – which should be unique in within the program
- its *arguments* – an "optional" list of the type(s) of input(s) that the function expects, and
- its *return type* – the type of the "single" output that this function will provide

If the function does not return any output value, the return type should be declared as `void`

- `void` essentially means "nothing" in languages like C, C++ and Java

An example declaration:

```
char* toUppercaseString(char*, int);
```

- Note the semi-colon at the end – this means that it is a declaration, and not definition

Declaring your own functions

Similar to a variable, a function can have a declaration as well as a definition…

- … or both can be merged together

A function's declaration includes

- its *name* – which should be unique in within the program
- its *arguments* – an "optional" list of the type(s) of input(s) that the function expects, and
- its *return type* – the type of the "single" output that this function will provide

If the function does not return any output value, the return type should be declared as `void`

- `void` essentially means "nothing" in languages like C, C++ and Java

An example declaration:

```
char* toUppercaseString(char*, int);
```

- Note the semi-colon at the end – this means that it is a declaration, and not definition

If the function does not take any inputs, the list of arguments should be replaced by `(void)`

Defining the functions

The definition of a function involves providing the C code that must be run when it is executed

Defining the functions

The definition of a function involves providing the C code that must be run when it is executed

It starts with the function *header* – something that looks very close to a declaration, i.e.

Defining the functions

The definition of a function involves providing the C code that must be run when it is executed

It starts with the function *header* – something that looks very close to a declaration, i.e.

- there is a return type,
- the name of the function, and
- the list of inputs that the function takes, along with their names

Defining the functions

The definition of a function involves providing the C code that must be run when it is executed

It starts with the function *header* – something that looks very close to a declaration, i.e.

- there is a return type,
- the name of the function, and
- the list of inputs that the function takes, along with their names

However, when we provide the definition for the function, we do not use a semi-colon…

- … instead, you put an opening bracket, provide the definition, and end it with a closing bracket

Defining the functions

The definition of a function involves providing the C code that must be run when it is executed

It starts with the function *header* – something that looks very close to a declaration, i.e.

- there is a return type,
- the name of the function, and
- the list of inputs that the function takes, along with their names

However, when we provide the definition for the function, we do not use a semi-colon…

- … instead, you put an opening bracket, provide the definition, and end it with a closing bracket

An example definition:

```
char* toUppercaseString(char* word, int number_of_characters)
{
    return word; // This is not really a definition !!
}
```

Defining the functions

The definition of a function involves providing the C code that must be run when it is executed

It starts with the function *header* – something that looks very close to a declaration, i.e.

- there is a return type,
- the name of the function, and
- the list of inputs that the function takes, along with their names

However, when we provide the definition for the function, we do not use a semi-colon…

- … instead, you put an opening bracket, provide the definition, and end it with a closing bracket

An example definition:

```
char* toUppercaseString(char* word, int number_of_characters)
{
    return word; // This is not really a definition !!
}
```

- Note that we also need to give each input variable a name, when we define the function

Example – Function to change case

```c
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>

#define MAX_CHARS 21

char* to_uppercase_word(char* word, int number_of_characters);

int main()
{
        char word[MAX_CHARS];
        int characters_count = 0, i = 0;
        char* uppercase_word = NULL;
        printf("Enter a word (up to 20 characters):\n");
        fgets(word, MAX_CHARS, stdin);
        // Find out the number of characters
        while(word[i++] != '\n')
                characters_count++;
        uppercase_word = to_uppercase_word(word, characters_count);
        printf("%s, when converted to uppercase, is %s\n", word, uppercase_word);
        free(uppercase_word);
        return 0;
}

char* to_uppercase_word(char* word, int number_of_characters)
{
        char* characters = NULL;
        int i;
        characters = (char*)malloc(sizeof(char) * (number_of_characters+1));
        for(i = 0; i < number_of_characters; i++)
                characters[i] = toupper(word[i]);
        characters[number_of_characters] = word[number_of_characters] = '\0';
        return characters;
}
```

Example – Function to change case

```c
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>

#define MAX_CHARS 21

char* to_uppercase_word(char* word, int number_of_characters);

int main()
{
        char word[MAX_CHARS];
        int characters_count = 0, i = 0;
        char* uppercase_word = NULL;
        printf("Enter a word (up to 20 characters):\n");
        fgets(word, MAX_CHARS, stdin);
        // Find out the number of characters
        while(word[i++] != '\n')
                characters_count++;
        uppercase_word = to_uppercase_word(word, characters_count);
        printf("%s, when converted to uppercase, is %s\n", word, uppercase_word);
        free(uppercase_word);
        return 0;
}

char* to_uppercase_word(char* word, int number_of_characters)
{
        char* characters = NULL;
        int i;
        characters = (char*)malloc(sizeof(char) * (number_of_characters+1));
        for(i = 0; i < number_of_characters; i++)
                characters[i] = toupper(word[i]);
        characters[number_of_characters] = word[number_of_characters] = '\0';
        return characters;
}
```

This is the same as the program we saw last week !!

Example – Function to change case

```c
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>

#define MAX_CHARS 21

char* to_uppercase_word(char* word, int number_of_characters);

int main()
{
        char word[MAX_CHARS];
        int characters_count = 0, i = 0;
        char* uppercase_word = NULL;
        printf("Enter a word (up to 20 characters):\n");
        fgets(word, MAX_CHARS, stdin);
        // Find out the number of characters
        while(word[i++] != '\n')
                characters_count++;
        uppercase_word = to_uppercase_word(word, characters_count);
        printf("%s, when converted to uppercase, is %s\n", word, uppercase_word);
        free(uppercase_word);
        return 0;
}

char* to_uppercase_word(char* word, int number_of_characters)
{
        char* characters = NULL;
        int i;
        characters = (char*)malloc(sizeof(char) * (number_of_characters+1));
        for(i = 0; i < number_of_characters; i++)
                characters[i] = toupper(word[i]);
        characters[number_of_characters] = word[number_of_characters] = '\0';
        return characters;
}
```

This is the same as the program we saw last week !!

But this one defines a function to convert the case of the entered word…

Example – Function to change case

```c
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>

#define MAX_CHARS 21

char* to_uppercase_word(char* word, int number_of_characters);

int main()
{
        char word[MAX_CHARS];
        int characters_count = 0, i = 0;
        char* uppercase_word = NULL;
        printf("Enter a word (up to 20 characters):\n");
        fgets(word, MAX_CHARS, stdin);
        // Find out the number of characters
        while(word[i++] != '\n')
                characters_count++;
        uppercase_word = to_uppercase_word(word, characters_count);
        printf("%s, when converted to uppercase, is %s\n", word, uppercase_word);
        free(uppercase_word);
        return 0;
}

char* to_uppercase_word(char* word, int number_of_characters)
{
        char* characters = NULL;
        int i;
        characters = (char*)malloc(sizeof(char) * (number_of_characters+1));
        for(i = 0; i < number_of_characters; i++)
                characters[i] = toupper(word[i]);
        characters[number_of_characters] = word[number_of_characters] = '\0';
        return characters;
}
```

This is the declaration for the function

Example – Function to change case

```c
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>

#define MAX_CHARS 21

char* to_uppercase_word(char* word, int number_of_characters);

int main()
{
        char word[MAX_CHARS];
        int characters_count = 0, i = 0;
        char* uppercase_word = NULL;
        printf("Enter a word (up to 20 characters):\n");
        fgets(word, MAX_CHARS, stdin);
        // Find out the number of characters
        while(word[i++] != '\n')
                characters_count++;
        uppercase_word = to_uppercase_word(word, characters_count);
        printf("%s, when converted to uppercase, is %s\n", word, uppercase_word);
        free(uppercase_word);
        return 0;
}

char* to_uppercase_word(char* word, int number_of_characters)
{
        char* characters = NULL;
        int i;
        characters = (char*)malloc(sizeof(char) * (number_of_characters+1));
        for(i = 0; i < number_of_characters; i++)
                characters[i] = toupper(word[i]);
        characters[number_of_characters] = word[number_of_characters] = '\0';
        return characters;
}
```

This is the declaration for the function

Remember that the names of the input variables are not required

Example – Function to change case

```c
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>

#define MAX_CHARS 21

char* to_uppercase_word(char* word, int number_of_characters);

int main()
{
        char word[MAX_CHARS];
        int characters_count = 0, i = 0;
        char* uppercase_word = NULL;
        printf("Enter a word (up to 20 characters):\n");
        fgets(word, MAX_CHARS, stdin);
        // Find out the number of characters
        while(word[i++] != '\n')
                characters_count++;
        uppercase_word = to_uppercase_word(word, characters_count);
        printf("%s, when converted to uppercase, is %s\n", word, uppercase_word);
        free(uppercase_word);
        return 0;
}

char* to_uppercase_word(char* word, int number_of_characters)
{
        char* characters = NULL;
        int i;
        characters = (char*)malloc(sizeof(char) * (number_of_characters+1));
        for(i = 0; i < number_of_characters; i++)
                characters[i] = toupper(word[i]);
        characters[number_of_characters] = word[number_of_characters] = '\0';
        return characters;
}
```

This is the declaration for the function

Remember that the names of the input variables are not required

In fact, this is a specific type of declaration, called a *prototype*; you don't even need to tell the type of input variables in a simple declaration !!

Example – Function to change case

```c
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>

#define MAX_CHARS 21

char* to_uppercase_word(char* word, int number_of_characters);

int main()
{
        char word[MAX_CHARS];
        int characters_count = 0, i = 0;
        char* uppercase_word = NULL;
        printf("Enter a word (up to 20 characters):\n");
        fgets(word, MAX_CHARS, stdin);
        // Find out the number of characters
        while(word[i++] != '\n')
                characters_count++;
        uppercase_word = to_uppercase_word(word, characters_count);
        printf("%s, when converted to uppercase, is %s\n", word, uppercase_word);
        free(uppercase_word);
        return 0;
}
```

```c
char* to_uppercase_word(char* word, int number_of_characters)
{
        char* characters = NULL;
        int i;
        characters = (char*)malloc(sizeof(char) * (number_of_characters+1));
        for(i = 0; i < number_of_characters; i++)
                characters[i] = toupper(word[i]);
        characters[number_of_characters] = word[number_of_characters] = '\0';
        return characters;
}
```

This is the definition for the function

Example – Function to change case

```c
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>

#define MAX_CHARS 21

char* to_uppercase_word(char* word, int number_of_characters);

int main()
{
        char word[MAX_CHARS];
        int characters_count = 0, i = 0;
        char* uppercase_word = NULL;
        printf("Enter a word (up to 20 characters):\n");
        fgets(word, MAX_CHARS, stdin);
        // Find out the number of characters
        while(word[i++] != '\n')
                characters_count++;
        uppercase_word = to_uppercase_word(word, characters_count);
        printf("%s, when converted to uppercase, is %s\n", word, uppercase_word);
        free(uppercase_word);
        return 0;
}
```

```c
char* to_uppercase_word(char* word, int number_of_characters)
{
        char* characters = NULL;
        int i;
        characters = (char*)malloc(sizeof(char) * (number_of_characters+1));
        for(i = 0; i < number_of_characters; i++)
                characters[i] = toupper(word[i]);
        characters[number_of_characters] = word[number_of_characters] = '\0';
        return characters;
}
```

This is the definition for the function

The `return` statement is used for sending the output of the function – if it has one !!

Example – Function to change case

```c
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>

#define MAX_CHARS 21

char* to_uppercase_word(char* word, int number_of_characters);

int main()
{
        char word[MAX_CHARS];
        int characters_count = 0, i = 0;
        char* uppercase_word = NULL;
        printf("Enter a word (up to 20 characters):\n");
        fgets(word, MAX_CHARS, stdin);
        // Find out the number of characters
        while(word[i++] != '\n')
                characters_count++;
        uppercase_word = to_uppercase_word(word, characters_count);
        printf("%s, when converted to uppercase, is %s\n", word, uppercase_word);
        free(uppercase_word);
        return 0;
}

char* to_uppercase_word(char* word, int number_of_characters)
{
        char* characters = NULL;
        int i;
        characters = (char*)malloc(sizeof(char) * (number_of_characters+1));
        for(i = 0; i < number_of_characters; i++)
                characters[i] = toupper(word[i]);
        characters[number_of_characters] = word[number_of_characters] = '\0';
        return characters;
}
```

You call the function in the same fashion as you call a library function

Example – Function to change case

```c
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>

#define MAX_CHARS 21

char* to_uppercase_word(char* word, int number_of_characters);

int main()
{
        char word[MAX_CHARS];
        int characters_count = 0, i = 0;
        char* uppercase_word = NULL;
        printf("Enter a word (up to 20 characters):\n");
        fgets(word, MAX_CHARS, stdin);
        // Find out the number of characters
        while(word[i++] != '\n')
                characters_count++;
        uppercase_word = to_uppercase_word(word, characters_count);
        printf("%s, when converted to uppercase, is %s\n", word, uppercase_word);
        free(uppercase_word);
        return 0;
}

char* to_uppercase_word(char* word, int number_of_characters)
{
        char* characters = NULL;
        int i;
        characters = (char*)malloc(sizeof(char) * (number_of_characters+1));
        for(i = 0; i < number_of_characters; i++)
                characters[i] = toupper(word[i]);
        characters[number_of_characters] = word[number_of_characters] = '\0';
        return characters;
}
```

You call the function in the same fashion as you call a library function

You pass it the expected inputs, and, either assign the output to a variable, use it in an expression, or simply ignore it !!

Some important points to note…

The inputs to a function are also called arguments or parameters

Some important points to note…

The inputs to a function are also called arguments or parameters

- The variables defined in the function header are called *formal parameters*

Some important points to note…

The inputs to a function are also called arguments or parameters

- The variables defined in the function header are called *formal parameters*
- The inputs to the function, either constants, or current values of variables, are called *actual parameters*

Some important points to note…

The inputs to a function are also called arguments or parameters

- The variables defined in the function header are called *formal parameters*
- The inputs to the function, either constants, or current values of variables, are called *actual parameters*

You must use a return statement in a function, if its return type is not `void`

Some important points to note…

The inputs to a function are also called arguments or parameters

- The variables defined in the function header are called *formal parameters*
- The inputs to the function, either constants, or current values of variables, are called *actual parameters*

You must use a return statement in a function, if its return type is not `void`

If the return type of a function is `void`, you are not required to give a return statement

Some important points to note…

The inputs to a function are also called arguments or parameters

- The variables defined in the function header are called *formal parameters*
- The inputs to the function, either constants, or current values of variables, are called *actual parameters*

You must use a return statement in a function, if its return type is not `void`

If the return type of a function is `void`, you are not required to give a return statement

- Although, a return statement, without any value, is acceptable; e.g. `return;`

Some important points to note…

The inputs to a function are also called arguments or parameters

- The variables defined in the function header are called *formal parameters*
- The inputs to the function, either constants, or current values of variables, are called *actual parameters*

You must use a return statement in a function, if its return type is not `void`

If the return type of a function is `void`, you are not required to give a return statement

- Although, a return statement, without any value, is acceptable; e.g. `return;`
- The return statement, if present (and encountered), is the last executed statement during the function's call

A slightly different example function

```c
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>

#define MAX_CHARS 21

void to_uppercase_word(char* word, int number_of_characters)
{
        if(number_of_characters == 0)
        {
                printf("You entered an empty string\n");
                return;
        }
        char* characters = NULL;
        int i;
        characters = (char*)malloc(sizeof(char) * (number_of_characters+1));
        for(i = 0; i < number_of_characters; i++)
                characters[i] = toupper(word[i]);
        characters[number_of_characters] = word[number_of_characters] = '\0';
        printf("%s, when converted to uppercase, is %s\n", word, characters);
        free(characters);
}

int main()
{
        char word[MAX_CHARS];
        int characters_count = 0, i = 0;
        printf("Enter a word (up to 20 characters):\n");
        fgets(word, MAX_CHARS, stdin);
        // Find out the number of characters
        while(word[i++] != '\n')
                characters_count++;
        to_uppercase_word(word, characters_count);
}
```

A slightly different example function

```c
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>

#define MAX_CHARS 21

void to_uppercase_word(char* word, int number_of_characters)
{
        if(number_of_characters == 0)
        {
                printf("You entered an empty string\n");
                return;
        }
        char* characters = NULL;
        int i;
        characters = (char*)malloc(sizeof(char) * (number_of_characters+1));
        for(i = 0; i < number_of_characters; i++)
                characters[i] = toupper(word[i]);
        characters[number_of_characters] = word[number_of_characters] = '\0';
        printf("%s, when converted to uppercase, is %s\n", word, characters);
        free(characters);
}

int main()
{
        char word[MAX_CHARS];
        int characters_count = 0, i = 0;
        printf("Enter a word (up to 20 characters):\n");
        fgets(word, MAX_CHARS, stdin);
        // Find out the number of characters
        while(word[i++] != '\n')
                characters_count++;
        to_uppercase_word(word, characters_count);
}
```

A slightly different version of the same program is shown below

A slightly different example function

```c
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>

#define MAX_CHARS 21

void to_uppercase_word(char* word, int number_of_characters)
{
        if(number_of_characters == 0)
        {
                printf("You entered an empty string\n");
                return;
        }
        char* characters = NULL;
        int i;
        characters = (char*)malloc(sizeof(char) * (number_of_characters+1));
        for(i = 0; i < number_of_characters; i++)
                characters[i] = toupper(word[i]);
        characters[number_of_characters] = word[number_of_characters] = '\0';
        printf("%s, when converted to uppercase, is %s\n", word, characters);
        free(characters);
}

int main()
{
        char word[MAX_CHARS];
        int characters_count = 0, i = 0;
        printf("Enter a word (up to 20 characters):\n");
        fgets(word, MAX_CHARS, stdin);
        // Find out the number of characters
        while(word[i++] != '\n')
                characters_count++;
        to_uppercase_word(word, characters_count);
}
```

Here, the function prints the output as well, and hence, does not require returning anything

A slightly different example function

```c
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>

#define MAX_CHARS 21

void to_uppercase_word(char* word, int number_of_characters)
{
        if(number_of_characters == 0)
        {
                printf("You entered an empty string\n");
                return;
        }
        char* characters = NULL;
        int i;
        characters = (char*)malloc(sizeof(char) * (number_of_characters+1));
        for(i = 0; i < number_of_characters; i++)
                characters[i] = toupper(word[i]);
        characters[number_of_characters] = word[number_of_characters] = '\0';
        printf("%s, when converted to uppercase, is %s\n", word, characters);
        free(characters);
}

int main()
{
        char word[MAX_CHARS];
        int characters_count = 0, i = 0;
        printf("Enter a word (up to 20 characters):\n");
        fgets(word, MAX_CHARS, stdin);
        // Find out the number of characters
        while(word[i++] != '\n')
                characters_count++;
        to_uppercase_word(word, characters_count);
}
```

Even here, a return statement without any value, can be used to complete the function call explicitly

Some important points to note…

The inputs to a function are also called arguments or parameters

- The variables defined in the function header are called *formal parameters*
- The inputs to the function, either constants, or current values of variables, are called *actual parameters*

You must use a return statement in a function, if its return type is not `void`

If the return type of a function is `void`, you are not required to give a return statement

- Although, a return statement, without any value, is acceptable; e.g. `return;`
- The return statement, if present (and encountered), is the last executed statement during the function's call

The `main()` is also a function – a special function from where the execution of a program starts

Some important points to note…

The inputs to a function are also called arguments or parameters

- The variables defined in the function header are called *formal parameters*
- The inputs to the function, either constants, or current values of variables, are called *actual parameters*

You must use a return statement in a function, if its return type is not `void`

If the return type of a function is `void`, you are not required to give a return statement

- Although, a return statement, without any value, is acceptable; e.g. `return;`
- The return statement, if present (and encountered), is the last executed statement during the function's call

The `main()` is also a function – a special function from where the execution of a program starts

If you define a function *before* you call it in the code, you don't need to provide its declaration

A slightly different example function

```c
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>

#define MAX_CHARS 21

void to_uppercase_word(char* word, int number_of_characters)
{
        if(number_of_characters == 0)
        {
                printf("You entered an empty string\n");
                return;
        }
        char* characters = NULL;
        int i;
        characters = (char*)malloc(sizeof(char) * (number_of_characters+1));
        for(i = 0; i < number_of_characters; i++)
                characters[i] = toupper(word[i]);
        characters[number_of_characters] = word[number_of_characters] = '\0';
        printf("%s, when converted to uppercase, is %s\n", word, characters);
        free(characters);
}

int main()
{
        char word[MAX_CHARS];
        int characters_count = 0, i = 0;
        printf("Enter a word (up to 20 characters):\n");
        fgets(word, MAX_CHARS, stdin);
        // Find out the number of characters
        while(word[i++] != '\n')
                characters_count++;
        to_uppercase_word(word, characters_count);
}
```

A slightly different example function

```c
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>

#define MAX_CHARS 21

void to_uppercase_word(char* word, int number_of_characters)
{
        if(number_of_characters == 0)
        {
                printf("You entered an empty string\n");
                return;
        }
        char* characters = NULL;
        int i;
        characters = (char*)malloc(sizeof(char) * (number_of_characters+1));
        for(i = 0; i < number_of_characters; i++)
                characters[i] = toupper(word[i]);
        characters[number_of_characters] = word[number_of_characters] = '\0';
        printf("%s, when converted to uppercase, is %s\n", word, characters);
        free(characters);
}

int main()
{
        char word[MAX_CHARS];
        int characters_count = 0, i = 0;
        printf("Enter a word (up to 20 characters):\n");
        fgets(word, MAX_CHARS, stdin);
        // Find out the number of characters
        while(word[i++] != '\n')
                characters_count++;
        to_uppercase_word(word, characters_count);
}
```

Just like a declaration is not required here !!

Homework !!

Find out a use case from the previous weeks for which you can use a custom function

- Implement the same

Find out what happens if…

- … we attempt to assign the output of a function to a variable, but the function's return type is `void`
- … or, we do not use the output of a function, even though the function is returning some value
- … or, we put a return statement without a value, but the function's return type is not `void`