C Programming
- Function

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

Institute for Digital Technologies
Loughborough University London

Function

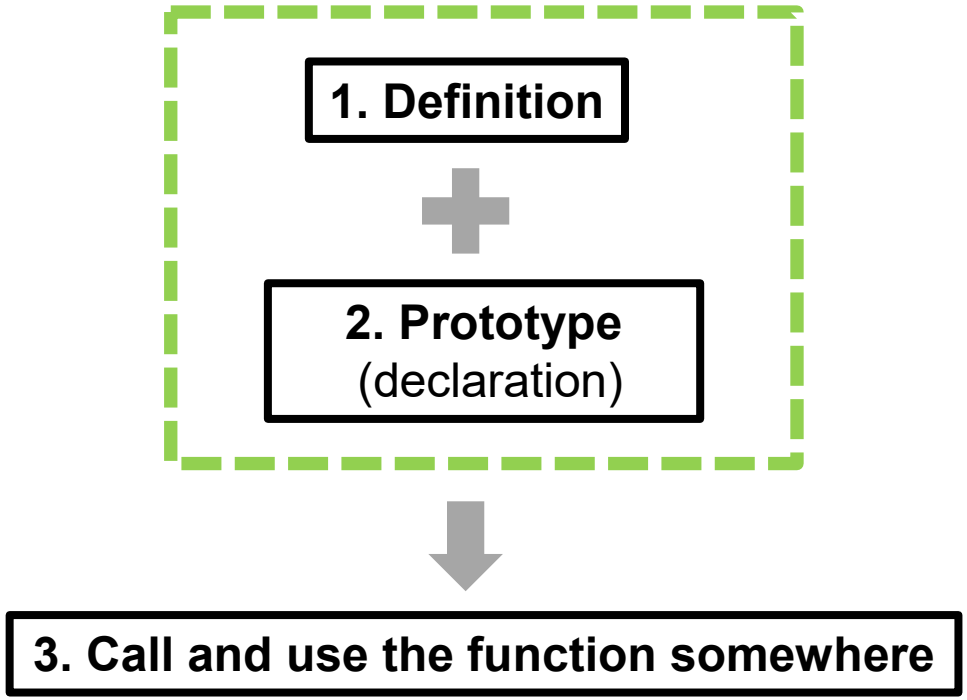
Functions

A function is an **independent** section of a C/C++/Python code that performs a specific task and optionally returns or/and receives values.

- Two categories of functions:

1. **Predefined** functions (e.g. printf(), strcpy()) : defined in standard libraries in C, such as *stdio.h, conio.h*
2. **User-defined** functions: functions that programmers create for special tasks such as to put values into a dataset, to run a specific algorithm.

User-defined functions in C



1. Definition

Header

[return type] [function name] ([data type] [input argument])

```
{
    // Body of the function, which contains the code.

 return (return-variable/-value);  // A function can return a value.
}
```

```
int my_func (int x)      // function header
 {
     int y = x*x;            // body
     return y;
 }
```

```
int main (void)
{
   //........
   return 0;
}
```

2. Prototyping

- Remember that we declare variables (primarily to give them a type) before using them,

  e.g. int AA;
- Similarly we must declare functions before using them.
- We could do this by putting function **declarations before** *main()*.
- However, *main()* does not require any prototype (declaration).
- Prototype of a function==Header of the function;, e.g.

```
float my_func (float x); // It is a prototype
    // input arguments can be omitted, such as
float my_func (float );
```

Example

```
float my_func(float x);          //  2. PROTOTYPE
                                 //       (declaration)

int main()
  {     float y;
         y=my_func (z);          //  3. CALL
    return 0; }

float my_func (float x)          //  1. DEFINITION
  {
      return x*x;
  }   // No semicolon ";"
```

function name

data type of the return variable

input variable

- Functions with no input arguments:
  [return type] [function name] (**void**)

- Functions with no return value:
  void [function name] ([data types] [argument list…])

```
int my_print (void)
 {      // void can be omitted
     printf ("I am happy!");
     return 0;
 }
```

```
int my_print ( )
 { …
  }
```

```
void my_print (int x)
 {
   if (x==0)
      printf ("False");
   else
      printf ("True");
   return;
 } // return; can be omitted
```

Homework (for Coursework)

Recall " String & Keyboard input "

```
int main(void)
{
    int nr=1;
    char name[20], title[40];   // struct book bkk; bkk.name, bkk.title
    int page;

    printf("Could you put book information details?\n");
    printf("Name: ");
      scanf_s("%[^\n]s", name, sizeof(name));
      rewind(stdin); // Empty stdin
    printf("Title: ");
      scanf_s("%[^\n]s", title, 40); rewind(stdin);
    printf("Page number: ");
      scanf_s("%i", &page); rewind(stdin);

    printf("\n\n\n[The details you've saved] \n");
    printf("No.[%d] \n", nr);
    printf("Name: %s \n", name);
    printf("Title: %s \n", title);
    printf("Page number: %d \n", page);

getchar();    // It is optional   // system("pause");   // while (!_kbhit());
return 0;
}
```

(1) Create your book struct

e.g,

```
struct book { …..
        char name[20];
        …..}
```

(2) Create your insert() function: it may use ***scanf()*** for old compilers. **For example,**

```
struct book my_insert()
{ …..
  struct book my_bk;
  ……
  return return my_bk;
}
```

(3) Create your display() function, e.g.

```
void show_details(int a_nr, char* a_name, char*
a_title, int a_page)
{  printf(…..)
   …..  }
```

%[^\n]: ***scanf_s()*** will read **all** characters including a 'space' until reach **\n** (*newline, enter*).

- **`%[^\n]`**: ***scanf_s()*** will read all characters including a 'space' until reach **`\n`** (*newline, enter)*. It is a common idiom to read a whole line in C.
- **`rewind()`** will empty the memory area.

Differences between *main()* and all other Functions

- ***main ()*** **runs first** among all the functions defined in a program.
- Only **one** ***main ()*** can be in a program.
- ***main ()*** requires **no prototype** (declaration).

Exception/error routine

- In practice, a **nonempty** input of characters is required in ***scanf_s*** and ***scanf().***
- Insert an **exception/ error routine** into your code to prevent a garbage output.
- `rewind()` will empty the memory area '*stdin*'.

```
//For example
   printf("Name: ");
   int input = scanf_s("%[^\n]s", name, sizeof(name));
             //  scanf("%[^\n]s", name); // C89, old
   while(0==input)
    {  rewind(stdin); // Empty stdin
       printf("Failed. Put a valid name.");
       printf("Name: ");
       input = scanf_s("%[^\n]s", name, sizeof(name));
     }
   rewind(stdin);
   printf("Title: ");
```

Last Session

- ***scanf() & rewind()***
- ***struct***
- ***User-defined function***

Keyboard input

```
// string input
        printf("Name: ");
            scanf_s("%[^\n]s", name, sizeof(name));
        rewind(stdin);  // Empty the stdin buffer holding \n
          .....
// number input
        printf("Page number: ");
            scanf_s("%d", &page);
```

// Input including spaces

// scanf() requires a pointer input

Structure

Name of a structure (or *Tag*)

```
struct book{
   char author[20];
    char title[30];
    int page, year;
    …
   };
```

Member variables

```
struct book Book;    // somewhere
```

User-defined function

```
float my_func(float x);          //  2. PROTOTYPE
                                 //     (declaration)

int main()
  {    float y, z;   z=2.5;
       y=my_func (z);            //  3. CALL
    return 0; }

float my_func (float x)          //  1. DEFINITION
  {
      return x*x;
  }  // No semicolon ";"
```

function name

data type of the return variable

input variable (optional)

Homework Answers

Homework – answers (structure array)

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

struct book // define a structure
  {  char author[20]; char title[30]; int page, year; };


int main() {
  struct book bk[10];  // declare structure array instances for 10 books
  bk[0].year = 2019;    bk[0].page = 230; // assign values to those structure's members
  strcpy(bk[0].author, "Mike Taylor C");       strcpy(bk[0].title, "C Programming III");
   //strcpy_s(bk[0].author, sizeof(bk[0].author), "Mike Taylor C"); // on Visual Studio
  bk[1].year = 2018;    bk[1].page = 220; // assign values to the members
  strcpy(bk[1].author, "Mike Taylor B");       strcpy(bk[1].title, "C Programming II");
  bk[2].year = 2017;    bk[2].page = 210;
  strcpy(bk[2].author, "Mike Taylor A");       strcpy(bk[2].title, "C Programming I");

  for(int i=0;i<3;i++)
    {
     printf("\n\n[bk%d]\n Author: %s\n", i, bk[i].author);
     printf("Title: %s\n Year: %d, Page: %d \n\n", bk[i].title, bk[i].year, bk[i].page);
    }
   //while (!_kbhit());  // getchar(); // optional
return 0;
}
```

Homework - answer (functions)

```c
#include <stdio.h>

struct book // define a structure
{
char name[20];  // name, title - pointers
char title[30];
int page, year; // numbers
};

void show_details(int, char*, char*, int);
struct book user_input();

int main()
{
int nr = 1;
struct book buk = user_input();
show_details(nr, buk.name, buk.title, buk.page);

getchar();
return 0;
}
```

Can we make it any better (compact)?

```c
void show_details(int no, char* nm, char* ttl, int pg)
 {
  printf("\n\n\n[The details you've saved] \n");
  printf("No.[%d] \n", no);
  printf("Name: %s \n", nm);
  printf("Title: %s \n", ttl);
  printf("Page number: %d \n", pg);
 }
```

```c
struct book user_input()
 {
  struct book my_bk;
  printf("Could you put book information details?\n");
  printf("Name: ");
  scanf_s("%[^\n]s", my_bk.name, sizeof(my_bk.name));
    // scanf("%[^\n]s", my_bk.name); // C89, old
  rewind(stdin); // Empty stdin
  printf("Title: ");
  scanf_s("%[^\n]s", my_bk.title, 40); rewind(stdin);
   // scanf("%[^\n]s", . . .); // C89, old
  printf("Page number: ");
  scanf_s("%i", &my_bk.page); rewind(stdin);
   // scanf("%[^\n]s", . . .); // C89, old
  return my_bk;
 }
```

Homework – answer (functions)

```c
#include <stdio.h>

struct book // define a structure
 {
  char name[20];
  char title[30];
  int page, year;
 };



void show_details(int, struct book);
struct book user_input();


int main()
{
    int nr = 1;
    struct book buk = user_input();
    show_details(nr, buk);
    getchar();
return 0;
}
```

```c
void show_details(int no, struct book bki)
 {
  printf("\n\n\n[The details you've saved] \n");
  printf("No.[%d] \n", no);
  printf("Name: %s \n", bki.name);
  printf("Title: %s \n", bki.title);
  printf("Page number: %d \n", bki.page);
 }
```

```c
struct book user_input()
 {
  struct book my_bk;
  printf("Could you put book information details?\n");
  printf("Name: ");
  scanf_s("%[^\n]s", my_bk.name, sizeof(my_bk.name));
   // scanf("%[^\n]s", my_bk.name); // C89, old
  rewind(stdin); // Empty stdin
  printf("Title: ");
  scanf_s("%[^\n]s", my_bk.title, 40);
   // scanf("%[^\n]s", . . .); // C89, old
  rewind(stdin);
  printf("Page number: ");
  scanf_s("%i", &my_bk.page); rewind(stdin);
   // scanf("%[^\n]s", . . .); // C89, old
  return my_bk;
 }
```

Much **simpler**, when you use a ***structure.***

cf, show_details(nr, **buk**.name, **buk**.title, **buk**.page);

Function - *struct* type input/output

```
struct book // define a structure
{
   char author[20];
   char title[30];
   int page, year;
};
```

```
struct book insert();
struct book insert()
{
 struct book bkk;
 strcpy(bkk.author, "Jamie");
 // (x) bkk.author = "Jamie";
 return bkk;
}
```

```
int main()
{
  struct book bk[10];  // We have 10 books!

  bk[1]=insert();

  printf("\n\n bk[1]\nAuthor: %s\n", bk[1].author);

  // while (!_kbhit());   //getchar(); // optional
  return 0;
}
```

Exercise

Function + *struct* type input/output + keyboard input

Coursework

```
struct book // define a structure
{
   char author[20];
   char title[30];
   int page, year;
};
```

```
int main()
{
  struct book bk[10];  // We have 10 books!

  bk[1]=insert();

 printf("\n\n bk[1]\nAuthor: %s\n", bk[1].author);

   // while (!_kbhit());   //getchar(); // optional
return 0;
}
```

```
struct book insert();
struct book insert()
 {
  struct book bkk;
  printf("Author(without space): ");
  scanf_s("%s \n", bkk.author, sizeof(bkk.author));   // instead of strcpy()
  return bkk;
 }
```

struct type input/output

Example

```
struct db
 {
    int year1, isbn;
  } str_inst;

void my_func (struct db);

void my_func (struct db item0)
 { printf("item0.year1 = %d\n", item0.year1);  }

int main()
 {
   str_inst.year1=1969;
   my_func (str_inst);
            // while (getchar()) // optional
   return 0;
 }
```

struct type input/output

```
struct db
 {
     int year1, isbn;
  } str_inst;

void my_func (struct db *);
void my_func (struct db *str_array) // all the items
 {  ............... }
          // my_func (struct db str_item)  // a single item
void main()
{ struct db my_str_array[5];
  my_func (my_str_array);
...............
}
```

Coursework (optional)

In addition

Multiple Returns

```
int input_test (int x)
 {
   switch (x)   // if, else if, else if, else
    {
      case 1: case 2: case 3:
        return 8;
      case 4: case 5: case 6:
        return 14;
      case 11: case 18:
        return 16;
      default:
        return 0;
    }
}
```

User-defined Header File

- Include the header file where you defined structures.
- Use double quotes because the header file is local, instead of the default.

```
struct book // save it in my_bk_strc.h file
   {  char author[20];
      char title[30];
      int page, year;
   };
```

```
#include "my_bk_strc.h"

void main()
 { …
   struct book bk1;
   bk1.year=1969;
   bk1.page=243;
   …
 }
```

Homework (Optional)

Or, alternatively set the PATH of a file: e.g.
#define PATH "C:\\aaa\\my_header.h"