UNIVERSITY OF WOLVERHAMPTON

6CS005 High Performance Computing

Lecture 1

Fundamentals of C Programming

Jnaneshwar Bohara

Topics Covered

- **Introduction to C Programming**
- **Preprocessor directives**
- **The main() function**
- **Input / Output**
- **The printf() function**
- **The scanf() function**
- **Arrays**
- **Strings in C**
- **Function prototypes**
- **Structures**

C for Java Programmers

Aims

- To be able to adapt your skills as a Java programmer to develop programs in the C language.
- To be able to use C development tools.
- To understand memory allocation, management and the use of pointers.

UNIVERSITY OF WOLVERHAMPTON

Why C?

- This is a High Performance Computing module.
  - We are aiming to get the highest possible performance from our hardware.
  - C enables good performance with low level control over the computer and its operating system.
- High performance comes at a price.
  - Things that you were shielded from with Java become your responsibility.
- Excellent frameworks for HPC come in the form of extensions to C.

C Is Just Like Java

- A lot of inspiration for Java has its roots in C.
- You will be able to read and understand most parts of a C program.
- In this course we will only concentrate on the things that differ from Java.
- It is up to you get a good book on C and start going through it yourself.
- In the First Part, we will look at input, output, array, functions and strctures
- In the Second Part, we will look at concept of pointers and Dynamic Memory Allocation.

UNIVERSITY OF WOLVERHAMPTON

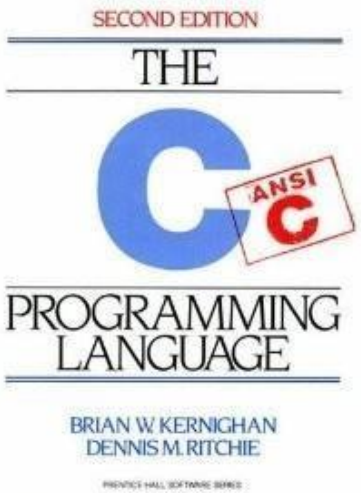
Resources

- If you find a good resource for learning C, please post a link to it on Google Classroom
  - No copyright/legal issues please.
- Many experienced programmers prefer the very concise book co-written by the author of the C language:

The C Programming Language (2nd Edition)

Brian Kernighan and Dennis Ritchie Prentice Hall, 1988

ISBN 978-0131103627



UNIVERSITY OF WOLVERHAMPTON

Java *vs.*

Java

1.A programming language

2. Object oriented

3. Garbage collector

4. No pointers

5.Better programming style, security

C

1.A programming language

2. Function oriented

3.Manage your own memory

4. Pointers

5.More efficient and powerful

About C

- *GNU : GNU's Not Unix*
  - *GNU C: gcc is a standard compiler*
- *C is non portable*
- *C is a high level language*

Programming on Linux

- Linux command line: GNU-C
  - Use console based editors: vi, emacs, nano
  - Or text based editors: kwrite, gedit, kate
- IDE
  - Eclipse * http://www.eclipse.org/cdt/downloads.php
  - Codeblocks
  - VS Code

* = available on windows too.

C on windows

- Use a text editor
  - install notepad++
  - compiler : MinGW
  - VS Code
- IDE
  - Eclipse *
  - Microsoft Visual C++ Express Edition
  - Codeblocks

My first C program!

```
#include <stdio.h>
// program prints hello world
int main() {
        printf ("Hello world!");
        return 0;
}
```

Output: Hello world!

Example 1

```
#include <stdio.h>
// program prints a number of type int
int main() {
        int number = 4;
        printf ("Number is %d", number);
        return 0;
}
```

Output: Number is 4

Example 2

```
#include <stdio.h>
// program reads and prints the same thing
int main() {
        int number ;
        printf (" Enter a Number: ");
        scanf ("%d", &number);
        printf ("Number is %d\n", number);
        return 0;
}
```

Output : Enter a number: 4
Number is 4

more and more

```
#include <stdio.h>

int main() {
        /* this program adds two numbers */
        int a = 4; //first number
        int b = 5; //second number
        int answer = 0; //result
        answer = a + b;
}
```

Preprocessor directives

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

- The #include directives "paste" the contents of the files stdio.h, stdlib.h and string.h into your source code, at the very place where the directives appear.
- These files contain information about some library functions used in the program:
  - stdio stands for "standard I/O", stdlib stands for "standard library", and string.h includes useful string manipulation functions.

The main() function

- main() is always the first function called in a program execution.

```
int main( void )
{ …
```

- void indicates that the function takes no arguments
- int indicates that the function returns an integer value
  - Q: Integer value? Isn't the program just printing out some stuff and then exiting? What's there to return?
  - A: Through returning particular values, the program can indicate whether it terminated "nicely" or badly; the operating system can react accordingly.

Input/Output

Formatted Input/Output function		
Type	**Input**	**Output**
char	`scanf()`	`printf()`
int	`scanf()`	`printf()`
float	`scanf()`	`printf()`
string	`scanf()`	`printf()`

Unformatted Input/Output function		
Type	**Input**	**Output**

char	`getch()`
	`getche()`
	`getchar()`
		`putch()`
		`putchar()`

int	-	-
float	-	-
string	`gets()`	`puts()`

Input / Output

- printf (); //used to print to console(screen)

- scanf (); //used to take an input from console(keyboard)

  - example: printf(“%c”, a); scanf(“%d”, &a);

  - More format specifiers

%c	The character format specifier.
%d	The integer format specifier.
%f	The floating-point format specifier.
%s	The string format specifier.

The printf()function

printf("Original input : %s\n", input);

- printf()is a library function declared in <stdio.h>
- Syntax: printf(*FormatString*, *Expr*, *Expr*...)
  - *FormatString*: String of text to print
  - *Expr*s: Values to print
  - *FormatString* has placeholders to show where to put the values (note: #placeholders should match #*Expr*s)
  - Placeholders: %s(print as string),
    %c(print as char),
    %d(print as integer),
    %f(print as floating-point)
  - \nindicates a newline character

The scanf()function

- & in scanf.
  - It is used to access the address of the variable used.
  - example:
    - scanf(%d,&a);
    - we are reading into the address of a.
- Data Hierarchy.
  - example:
    - int value can be assigned to float not vice-versa
    - **Type casting**.

Type Casting

- Also called as **type conversion/data conversion**
- Type casting refers to changing an variable of one data type into another
- C programming provides two types of type casting operations
  - **Implicit type casting**(Meaning: indirect, automatic typecasting)

  Rules for converting data type automatically in C

  (lower type)char ➔ short ➔ int ➔unsigned int ➔ long ➔unsigned long ➔long long ➔unsigned long long ➔float ➔dluble ➔long double

  (higher type)
  - **Explicit type casting**(Meaning: direct)

Implicit type casting

```
#include <stdio.h>
   main() {
   int  number = 1;
   char character = 'a'; /*ASCII value is 97 */
   int sum;
   sum = number + character;
   /*
            Here 'a' is a char data type and sum is int data type, so before
            calculating sum the compiler convert char values of 'a' into ASCII
            int value 97 is called integer promotion. After that sum is
            calculated with int type
Integer promotion: process of converting lower type than int or unsigned int into int or unsigned
int and Converting from smaller data type into larger data type is also called as type promotion
   */
   printf("Value of sum : %d\n", sum );
}
```

Explicit type casting

- The type conversion performed by the programmer directly by type cast the result(or expression) to make it of particular data type
- Syntax:
  - (data_type_name) expression;
- For instance:
  - int result, var1=10, var2=3;
  - result=var1/var2;
  - Guess the result?
  - Result will store in the variable **result** with **int format** i.e. result loss its meaning
  - In that situation we required to use explicit type casting

Explicit type casting

```
#include<stdio.h>
int main(){
        int var1=10, var2=3;
//var1 and var2 will converted into float type ➔ explicit type casting
        float result=(float)var1/var2;
        printf("result:  %f\n",result);
        return 0;

}
```

Inbuilt Type casting Function

- Five different type casting are
  1. **atof()**
     Converts string to float
  2. **atoi()**
     Converts string to int
  3. **atol()**
     Converts string to long

return vs. exit

- EXIT_SUCCESS is a constant defined in stdlib. Returning this value signifies successful termination

- EXIT_FAILURE is another constant, signifying that something bad happened requiring termination.

- exit differs from return in that execution terminates immediately - control is *not* passed back to the calling function main().

Some more basics

- Keywords (32 keywords)
  - char, static, if , while, return .....
- Format Specifiers
  - %f,%d,%s,%c……..
- Data Types
  - int , char, float..……..
- Operators and Expressions
  - Arithmetic Operators
    - + (Plus), - (Minus), * (Multiplication), /(Division)
  - Relational Operators
  - Logical Operators
  - Assignment Operators
  - Increment/decrement Operators

Arrays

- An Array is a collection of variables of the same type that are referred to through a common name.

- Declaration

  type var_name[size]

  e.g

- int A[6];
- float f[15];

Array Initialization

After declaration, array contains some garbage value.

Static initialization

```
int month_days[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

Run time initialization

```
int i;
int A[6];
for(i = 0; i < 6; i++)
        A[i] = 6 - i;
```

Strings in C

- No “Strings” keyword
- A string is an array of characters.

```
char msg[] = “hello world”;
char *msg= “hello world”;
```

A C String of Characters with Addresses

1234:0000	1234:0001	1234:0002	1234:0003	1234:0004	1234:0005	1234:0006	1234:0007	1234:0008	1234:0009	1234:000A	1234:000B
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
H	e	l	l	o		W	o	r	l	d	'\0'

Function prototypes

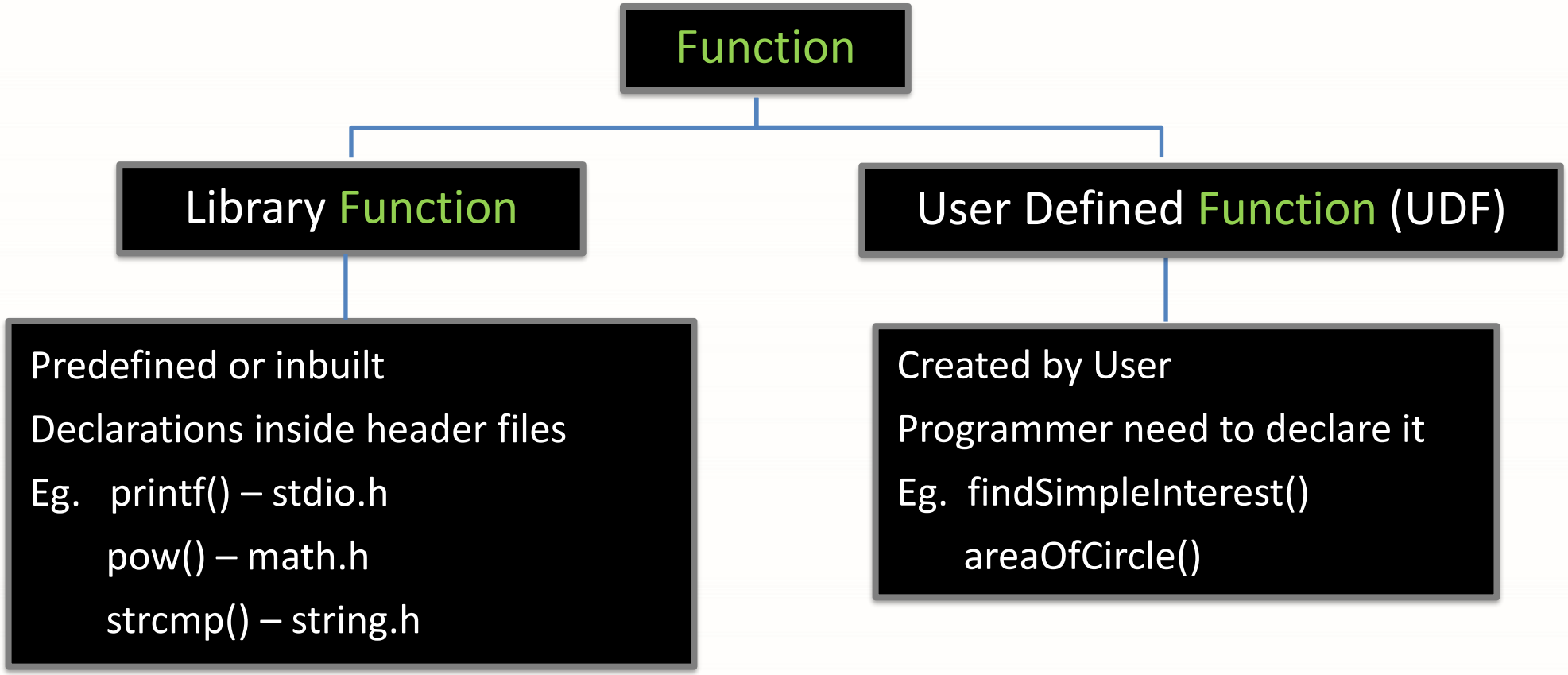
- A function is a group of statements that perform a specific task.
- It divides a large program into smaller parts.
- A function is something like hiring a person to do a specific job for you.
- Every C program can be thought of as a collection of these functions.
- Program execution in C language starts from the main function.

Syntax

```
void main()
{
    // body part
}
```

- Why function ?
  - Avoids rewriting the same code over and over.
  - Using functions it becomes easier to write programs and keep track of what they doing.

Function


Function
Library Function
User Defined Function (UDF)
Predefined or inbuilt
Declarations inside header files
Eg. printf() – stdio.h
pow() – math.h
strcmp() – string.h
Created by User
Programmer need to declare it
Eg. findSimpleInterest()
areaOfCircle()

Program Structure for Function

When we use a user-defined function program structure is divided into three parts.

Function Structure

```
void func1();


void main()
{
  ....
  func1();
}

void func1()
{
  ....
  //function body
  ....
}
```

Function Prototype

Function call

Function definition

Function Prototype and Function Definition

Syntax

Declaration

```
return-type function-name (arg-1, arg 2, …);
```

Definition

```
return-type function-name (arg-1, arg 2, …)
{
        //... Function body
}
```

Example

```
void addition(int, int);
```

```
void addition(int x, int y)
{
 printf("Addition is=%d“,(x+y));
}
```

Function prototypes

- A function Prototype also know as function declaration.
- A function declaration tells the compiler about a function name and how to call the function.
- It defines the function before it is being used or called.
- A function prototype needs to be written at the beginning of the program.

Syntax

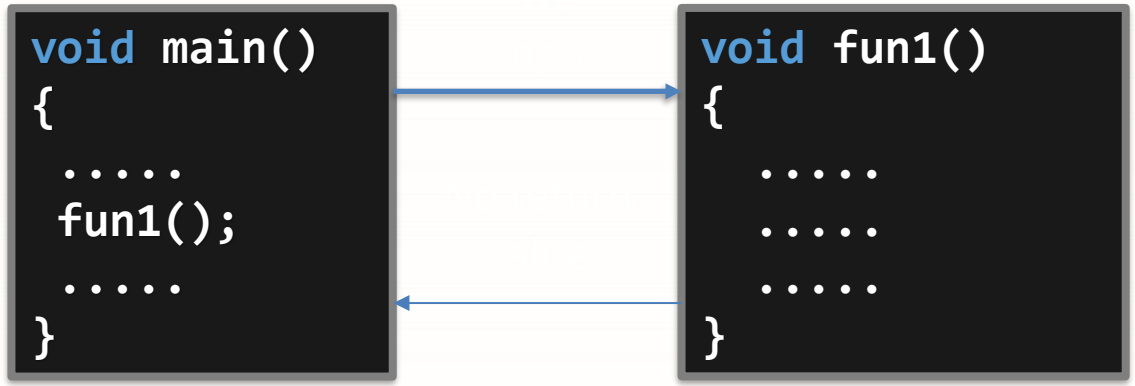
```
return-type function-name (arg-1, arg 2, …);
```

Example

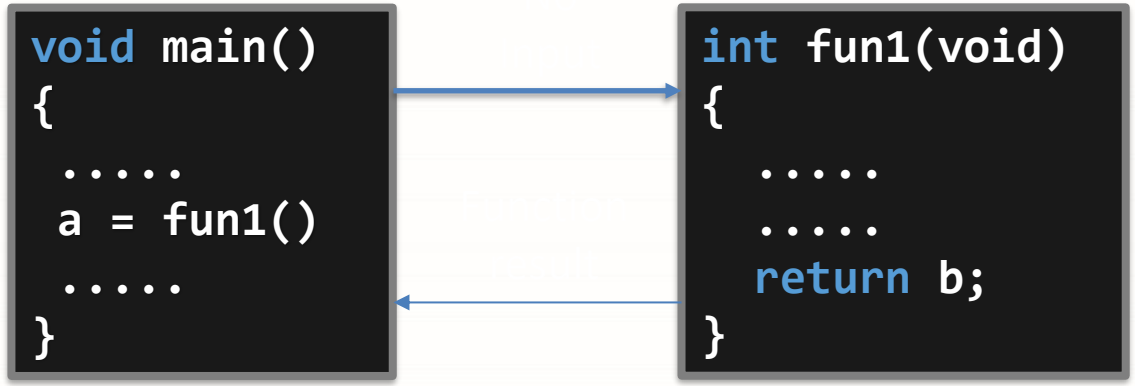
```
void addition(int, int);
```

Category of Function

(1) Function with no argument and but no return value

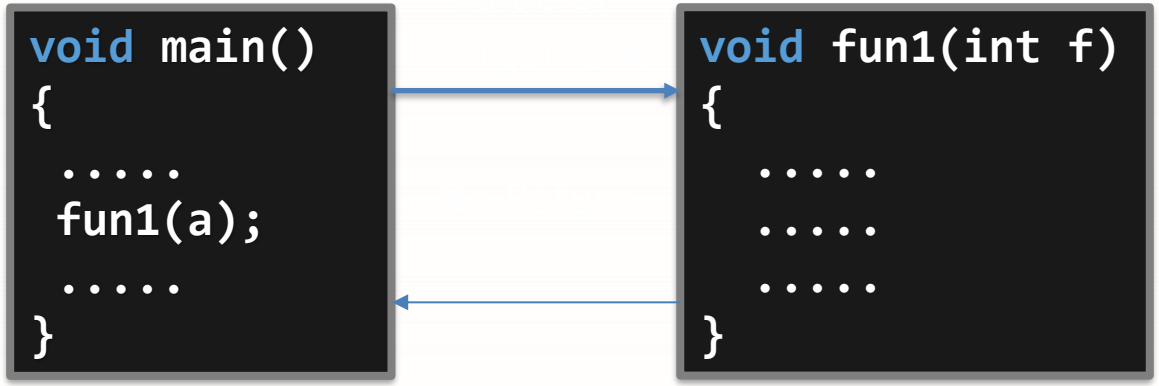


(2) Function with no argument and returns value

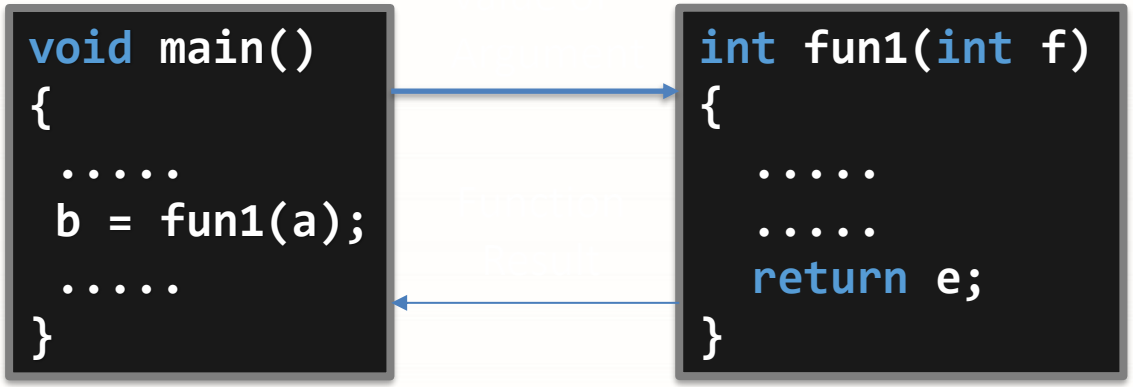


Category of Function cont.

(3) Function with argument and but no return value



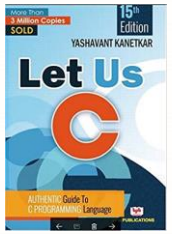
(4) Function with argument and returns value



Structures

We need combination of various datatypes to understand different entity/object.

Example-1:

Book		
	**Title**: Let Us C	**Datatype**: char / string
	Author: Yashavant Kanetkar	**Datatype**: char / string
	Page: 320	**Datatype**: int
	Price: 255.00	**Datatype**: float

Example-2:

Student		
	**Name**: ABC	**Datatype**: char / string
	Roll_No: 180540107001	**Datatype**: int
	CPI: 7.46	**Datatype**: float
	Backlog: 01	**Datatype**: int

Structures

A Structure is a collection of related data items, possibly of different types.

A structure type in C/C++ is called struct.

A struct is heterogeneous in that it can be composed of data of different types.

In contrast, array is homogeneous since it can contain only data of the same type.

Structures

- Structures hold data that belong **together**.

- Examples:

  Student record: student id, name, major, gender, start year, …

  Bank account: account number, name, currency, balance, …

- Address book: name, address, telephone number, …
  In database applications, structures are called records.

Structures

- Individual components of a struct type are called members (or fields).

  Members can be of different types (simple, array or struct).

  A struct is named as a whole while individual members are named using field identifiers.

  Complex data structures can be formed by defining arrays of structs.

struct basics

- Definition of a structure:

```
struct <struct-type>{
    <type> <identifier_list>;
    <type> <identifier_list>;
    ...
} ;
```

- Example:

```
struct Date {
    int day;
    int month;
    int year;
} ;
```

struct examples

- Example:

```
struct BankAccount{
    char Name[15];
    int AcountNo[10];
    double balance;
    Date Birthday;
};
```

- Example:

```
struct StudentRecord{
    char Name[15];
    int Id;
    char Dept[5];
    char Gender;
};
```

struct examples

- Example:

```
struct StudentInfo{
    int Id;
    int age;
    char Gender;
    double CGA;
};
```

- Example:

```
struct StudentGrade{
    char Name[15];
    char Course[9];
    int Lab[5];
    int Homework[3];
    int Exam[2];
};
```

Example Program

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


struct Books {
        char  title[50];
        char  author[50];
        char  subject[100];
        int   book_id;
};
```

Example Program

```
int main( ) {

   struct Books Book1;        /* Declare Book1 of type Book */
   struct Books Book2;        /* Declare Book2 of type Book */

   /* book 1 specification */
   strcpy( Book1.title, "C Programming");
   strcpy( Book1.author, "Nuha Ali");
   strcpy( Book1.subject, "C Programming Tutorial");
   Book1.book_id = 6495407;
```

Example Program

```
/* print Book1 info */

printf( "Book 1 title : %s\n", Book1.title);
printf( "Book 1 author : %s\n", Book1.author);
printf( "Book 1 subject : %s\n", Book1.subject);
printf( "Book 1 book_id : %d\n", Book1.book_id);
return 0;
}
```

Union

- Union is a user defined data type similar like Structure.
- It holds different data types in the same memory location.
- You can define a union with various members, but only one member can hold a value at any given time.
- Union provide an efficient way of using the same memory location for multiple-purpose.

Syntax to Define and Access Union

- Declaration of union must start with the keyword union followed by the union name and union's member variables are declared within braces.

Syntax

```
union union_name
{
    member1_declaration;
    member2_declaration;
. . .
    memberN_declaration;
};
```

union_name is name of custom type.

memberN_declaration is individual member declaration.

- Accessing the union members:
  - You need to create an object of union to access its members.
  - Object is a variable of type union. Union members are accessed using the dot operator(.) between union's object and union's member name.

Syntax

```
union union_name union_variable;
```

Command Line Argument(CLA) in C

- The arguments passed from command line are command line arguments
- CLA are simple arguments that are specified after the name of the program in the system's command line, and these argument values are passed on to your program during program execution as like this:
  - `gcc program.c -o program` //compile the program
  - `./program arg1 arg2 arg3` //run the program with arguments
- Arguments are handled by main().

Components of Command Line Argument(CLA)

- There are two components of Command Line Arguments in C:
  - **argc**: It refers to *"argument count"*, it is the first parameter that we use to store the number of command line arguments. It is important to note that the value of the argc should be greater than or equal to one.
  - **argv**: It refers to *"argument vector"*, It is basically an array of character pointer which we use to list all the command line arguments.
    - ***argv[0]:*** It contains the name of the program itself
    - ***argv[1]*** through ***argv[argc-1]*** contains any additional arguments provided when the program is executed.
- Argv[argc] is the NULL pointer
  - *Null pointer: Pointer in programming that does not point to any memory location or object*

Syntax

- *Syntax:*

```
int main(int argc, char *argv[]){
        //body
}
```

Example

source code

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

int main(int argc, char *argv[]) {
    if (argc == 1) {
        printf("No command-line arguments provided.\n");
        exit(EXIT_SUCCESS);  // Exit successfully
    } else {
        printf("Program name: %s\n", argv[0]);
        printf("Arguments:\n");
        for (int i = 1; i < argc; i++) {
            printf("Argument %d: %s\n", i, argv[i]);
        }

        exit(EXIT_SUCCESS);  // Exit successfully
    }
}
```

UNIVERSITY OF WOLVERHAMPTON

End of Lecture 1

Any Questions??☺