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

UNIVERSITY OF WOLVERHAMPTON

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.

UNIVERSITY OF WOLVERHAMPTON

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

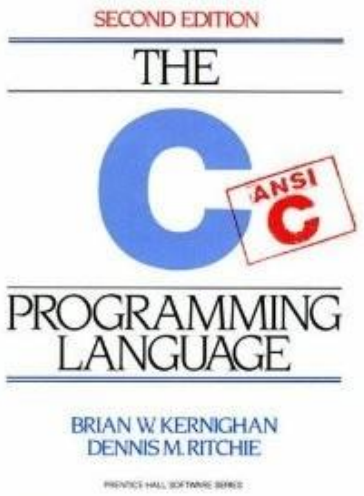
- If you find a good resource for learning C, please post a link to it on Google Classroom
  - No copyright/legal issues please.
- Many experienced programmers prefer the very concise book co-written by the author of the C language:

The C Programming Language (2nd Edition)

Brian Kernighan and Dennis Ritchie Prentice

Hall, 1988

ISBN 978-0131103627



UNIVERSITY OF WOLVERHAMPTON

Java *vs*. C

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

* = available on windows too.

C on windows

- Use a text editor
  - install notepad++
  - compiler : MinGW
- 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

- 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)
  - `\n` indicates 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.

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
  - char, static, if , while, return .....
- Data Types
  - int , char, float ……..
- Arithmetic Operators
  - + (Plus), - (Minus), * (Multiplication), /(Division)

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

```
int  read_column_numbers( int columns[], int max );

void rearrange( char *output, char const *input,
     int n_columns, int const columns[] );
```

- These look like function definitions – they have the name and all the type information – but each ends abruptly with a semicolon. Where's the body of the function – what does it actually *do*?

- (Note that each function *does* have a real definition, later in the program.)

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;
}
```

UNIVERSITY OF WOLVERHAMPTON

End of Lecture 1