EL2310 – Scientific Programming

Lecture 7: Basics of C

Yasemin Bekiroglu
(yaseminb@kth.se)

Royal Institute of Technology – KTH

Overview

Lecture 7: Basics of C

Wrap Up

Basic Datatypes and printf

Branching and Loops in C

Constant values

Arrays

Functions and return values

Lecture 7: Basics of C

Wrap Up

Basic Datatypes and printf

Branching and Loops in C

Constant values

Arrays

Functions and return values

Wrap up

- Started with C

Hello world

```
#include <stdio.h>

main()
{
  printf(‘‘Hello world\n’’);
}
```

Steps to a running program

- **Write**
- **Compile**
- **Link**
- **Execute**

From: http://www.physics.drexel.edu/courses/Comp_Phys/General/C_basics/compile.html

Homework from previous lecture

- Install and run the virtual machine
- Start Emacs or Vim
- Type, compile and run a Hello-world program
- Check out coding conventions!

Acknowledgement

- The C part of the course is based on the book by Kernighan & Ritchie

Compiling and running the program

- To compile program hello.c to executable file which will be called `hello` and run under Unix/Linux
-
  ```
  gcc -o hello hello.c
  ./hello
  ```
- The prefix `./` – the program is in the current directory
- Just like in Matlab there is a `PATH` variable that tells the system where to look for programs to run
- In Unix/Linux systems this `PATH` does normally not contain the current directory

Lecture 7: Basics of C

Wrap Up

Basic Datatypes and printf

Branching and Loops in C

Constant values

Arrays

Functions and return values

C statement

- A statement in C can be a single line followed by semicolon, or
- many statements enclosed by braces { }

Comments

- Multi-line comments
  The compiler will ignore everything between `/*` and `*/`
- Single-line comments (starting from C99)
  The compiler will ignore the rest of the line after `//`

```
#include <stdio.h>

main()
{
  /* This is a nice comment, is it not!  */
  printf("Hello world\n"); // This line prints
}
```

Data types

- There are only a few data types in C
  - `char`: character - a single byte
  - `int`: integer
  - `float`: floating point number
  - `double`: double precision floating point
- Can add qualifiers to get versions of these
  - `short int`: fewer bytes integer (maybe, depends on platform)
  - `long int`: integer with more bytes (maybe, depends on platform)
  - `unsigned int`: unsigned version (i.e. min value 0)
  - `signed int`: signed version (the default)
- More at `http://en.wikipedia.org/wiki/C_data_types`

Variable declarations

- In Matlab we could just use a variable, but not in C
- In C you need to declare the variables before you use them
- Old C: typically at the head of the function (or block)
- C99: Can be as close to where they are used as possible
- Declaring: `<type> <variable_name>`

```
int some_number;
int anumber, anothernumber, yetanothernumber;
int some_number=3;
```

printf

- You can use `printf` to print not only for strings but the value of variables
  Ex: `printf("This is iteration %d and the error is %f\n", iter, err);`
- To indicate that you want to print out a variable value you use the % character followed by a specification for what variable that is
  `%d` to print integer
  `%f` to print floating point

`printf` cont'd

- You can specify how many characters should be printed (at least)

  ```
  printf("The number of participants is %6d\n", dist)
  ```

  Will print at least 6 character

  Ex: `The number of participants is      4`
- Can be used to align things

`printf` cont'd

- You can specify how many characters after a decimal point you want (at least)

  ```
  printf("The distance is %.2fm\n", dist)
  ```

  Will print 2 decimals

  Ex: `The distance is 4.00m`
- Can combine number of characters and number of decimals

  ```
  printf("The distance is %6.2fm\n", dist)
  ```

  Will print 6 characters and 2 decimals

  Ex: `The distance is   4.00m`

  Notice that the dot counts as a character
- Can pad with zeros

  ```
  printf("The distance is %06.2fm\n", dist)
  ```

  Ex: `The distance is 004.00m`

`printf` cont'd

- More switches to printf
  - `%o` octal
  - `%x` hexadecimal
  - `%c` character
  - `%s` character string
  - `%%` to get % itself
- `www.cplusplus.com/reference/clibrary/cstdio/printf/` or `man 3 printf` in Linux

Task 1

- Declare an integer and print this integer in decimal, octal and hexadecimal form

`sizeof`

- Different types have different sizes
- The function `sizeof` can be used to get the size, i.e. number of bytes of a variable or data type
- Syntax: `sizeof(<variable/data type>)`
- Is an operator not a function
- Relates data types to the Machine type

Task 2

- Write a program that lists the number of bytes for some of the basic data types
- Is there a different between `short int`, `int` and `long int` on your machine?
- Do **NOT** assume the size of a type

Lecture 7: Basics of C

Wrap Up

Basic Datatypes and printf

Branching and Loops in C

Constant values

Arrays

Functions and return values

if-else

- Can control the flow with `if-else`

```
if (<expression>) <statement>
```

- or

```
if (<expression>)
  <statement>
else
  <statement>
```

- Remmeber that statement could be one line followed by semicolon
- or many lines with semicolon enclosed in { }
- Difference from MATLAB: The logical expressions have to be inside parentheses

if-else cont'd

- If you want to test more than one thing you can extend it with

```
if <expression>
  <statement>
else if <expression>
  <statement>
else
  <statement>
```

Logical expressions

- Similar to MATLAB
- Everything non-zero evaluates to true, zero is false
- Ex:

```
int value = 1;
if (value) {
  printf("Yippie, it is true\n");
} else {
  printf("Too bad, it is false\n");
}
```

Simple manipulations

- Assign a value to a variable: `i = 0`
- Increment a variable: `i += 2;`
  (which is short for `i = i + 2;`)
- If increment is 1 we can also write: `i++;`
  `i--;` is the same as `i = i - 1;`
- More advanced note: `i++` vs `++i`
  What if we have a stupid compiler without any optimizations?

switch

- Just like in matlab you can use `switch`
- Syntax:

```
switch (<variable>)
{
  case value1:
    <statement>
  break;
  case value2:
    <statement>
  break;
  default:
    <statement>
}
```

Task 3

- Write a program that generates a random number 0,1,2,. . . ,9 and prints out a special message for 0 and 1 and a general message for 2-9.
- stdlib.h, time.h
  `www.cplusplus.com/reference/clibrary/cstdlib/`
- Seed: `srand(seed)`, one can use current epoch time: `time(NULL)`
- Random number: `rand()` from 0 to `RAND_MAX` (at least 32767)
- Modulo (MATLAB mod): `%`

`for`-loop

- Can repeat code with `for`-loop
- Syntax:
```
for(<statement1>; <expression>; <statement2>)
   <statement3>
```
- Typically:
```
for(variable=value1; <expression>; variable++)
   <statement3>
```
- Need to declare `variable` and `value1` above
  This can be done inside `for` in C99
- `<expression>` is typically something that tests the value of the `variable` against some limits
- Ex:
```
for (i = 0; i < 10; i++)
     printf("i=%d\n",i);
```

Task 4

- Write a program that loops over two variables until one reaches limit. The first one should go from 0 to 9 and the second from 42 to 60 with step 2
- Use operator `,` (coma)
- `http://en.wikipedia.org/wiki/Comma_operator`

`while`-loop

- Syntax: `while(<expression>) <statement>`
- `<expression>` is typically something that test the value of some variable changed inside the loop
- Ex:

```
while (i < 10) {
  printf("i=%d\n",i);
  i++;
}
```

do-while-loop

- Syntax: `do <statement> while(<expression>)`
- `<expression>` is typically something that test the value of some variable changed inside the loop
- Will always execute the loop at least once!
- Ex:

```
i = 10;
do {
  printf("i=%d\n", i);
  i++;
} while (i < 10);
```

Task 5

- Write a program that prints a table with conversion from Celsius to Fahrenheit
- Tip: F = 32 + 9/5*C

Division

- Did you notice problems with accuracy when converting from Celcius to Fahrenheit?
- `9/5*tempC` where tempC is a double will be interpreted as integer division. Will result in `1*tempC`
- To fix you can:
  - Make sure that the compiler understands that it is a double `9.0/5*tempC`
  - Switch the order so that the tempC variable (which is a double) comes first `tempC*9/5`

Lecture 7: Basics of C

Wrap Up

Basic Datatypes and printf

Branching and Loops in C

Constant values

Arrays

Functions and return values

Constant values: Literals

- Integers
  - Ex: `1234`
  - Will be assumed to be an int (if it fits)
  - To tell the compiler that it should be a long int, use suffix `l` or `L`, e.g. `1234L`
  - Can specify in decimal (normal), octal or hexadecimal form
  - Octal: prefix with `0` (zero)
  - Hexadecimal: prefix with `0x`
- Floating points
  - Ex: `123.4`
  - Assumed to be a double
  - Suffix `f` or `F` gives float, e.g. `123.4f`

Character literals

- Character constants
- Ex: `'x'` or `'\n'`
- Character in single quotes
- Can be interpreted as a number
- `'0'` is 48

String literals

- Sequence of characters in double quotes
  Ex: `"Hello, world"`
- Can contain zero or more characters
- Converted to an array of characters (`char`) with character '\0' at the end.
- String constants are concatenated by the compiler
  Ex: `"Hello" ", world"` is the same

Defined constants

- It is often bad to use numerical constants directly in the code
- Makes the code hard to read
- Can use constants defined using preprocessor statements
- Syntax: `#define <name> <replacement text>` Ex: `#define LOWER_LIMIT 100`
- Remember `RAND_MAX`

Preprocessor

- An additional step before compilation:
  - 1. Preprocessor
  - 2. Compiler
  - 3. Linker
- Preprocessor statements start with `#`
- Includes files with `#include`
- Replaces constants defined with `#define`
- Conditional compilation with `#if #endif`

Data types

- There are only a few data types in C

  `char`: character - a single byte
  `int`: integer
  `float`: floating point number
  `double`: double precision floating point

- Can add qualifiers to get versions of these

  `short int`: fewer bytes integer (maybe, depends on platform)
  `long int`: integer with more bytes (maybe, depends on platform)
  `unsigned int`: unsigned version (i.e. min value 0)
  `signed int`: signed version (the default)

printf

- Some switches to printf
  - `%d` integer (decimal format)
  - `%6d` 6 character wide integer (can be any number)
  - `%f` floating point number
  - `%6.2f` floating point number with 6 characters out of which 2 are decimals
  - `%o` octal
  - `%x` hexadecimal
  - `%c` character
  - `%s` character string
  - `%%` to get % itself

for-loop

- Can repeat code with for-loop
- Syntax: for(variable=value1; <expression>; variable++) <statement>
- Need to declare variable and value1 above
- <expression> is typically something that test the value of the variable against some limits
- Ex:

```
for (i = 0; i < 10; i++) {
  printf("i=%d\n",i);
}
```

while-loop

- Can repeat code with `while`-loop
- Syntax: `while(<expression>) <statement>`
- `<expression>` is typically something that test the value of some variable changed inside the loop
- Ex:

```
i = 0;
while (i < 10) {
  printf("i=%d\n",i);
  i++;
}
```

- Identical result to the `for`-loop above

`for`-loop continue

- Given:

  ```
  for (A;B;C) D;
  ```

- `A` will be executed once first followed by

  ```
  while (B) {
    D;
    C;
  }
  ```

- Notice that you can squeeze in more than one assignment in `A` and `C`. Separate with comma (,)
- Ex: `for (i=0,j=1;i<10;i++,j+=2)`

  ```
    printf("%d\n",i*j);
  ```

do-while-loop

- Can repeat code with `do-while`-loop
- Syntax: `do <statement> while(<expression>)`
- `<expression>` is typically something that test the value of some variable changed inside the loop
- Ex:

```
i = 10;
do {
  printf("i=%d\n", i);
  i++;
} while (i < 10);
```

- Will always execute the loop at least once!

break and continue

- Can break out of a loop with `break`
- Can skip to the top of the loop with `continue`:

```
for (i = 0; i < 100; i++) {
  if (i < 10) continue; /* Too small */
  if (i == 42) break; /* Leave the loop */
  /* Perform interesting calculation */
...
}
```

Division

- Did you notice problems with accuracy when converting from Celcius to Fahrenheit?
- `9/5*tempC` where tempC is a double will be interpreted as integer division. Will result in `1*tempC`
- To fix you can:
  - Make sure that the compiler understands that it is a double `9.0/5*tempC`
  - Switch the order so that the tempC variable (which is a double) comes first `tempC*9/5`

Effecient assignments

- Alternative to `i = i + 1;` is `i ++;`
- Alternative to `i = i + 2;` is `i += 2;`
- Most operators have this version as well
- `expr1 = expr1 [op] expr2` can be written
- `expr1 [op]= (expr2)`

Task 1

- What will the following do

```
x = 1;
y *= x + 2;
```

Lecture 7: Basics of C

- Wrap Up
- Basic Datatypes and printf
- Branching and Loops in C
- Constant values
- **Arrays**
- Functions and return values

Arrays

- You declare an array by adding `[size]` after the variable name
- Ex: `int values[10];`
- Note: In C the index into an array starts at 0
- You set/get elements using syntax `values[i]`

Assigning initial values to arrays

- You can assign values to the array when you declare them
- `int values[3] = {1,2,3};`
- You do not have to assign all values but you cannot assign too many
- You can also let the assignment define the number of elements
- `double matrix[] = {1,2,3,4};`
  will give you an array with 4 elements

Character arrays

- The most commonly used array in C is the character array
  Ex: `char myname[32];`
- Assigning initial value to a character array:
  `char myname[]="This is my name";`

Multidimensional arrays

- You can have more than one dimension in the array
- You add more `[]` at the end
- Ex: double matrix[3][3];
- You set/get elements using syntax `matrix[i][j]`

Assigning initial values to arrays cont'd

- For two dimensional arrays
  ```
  double matrix[3][2] = {1,2,3,4,5,6};
  ```
  or a bit more clear
  ```
  double matrix[3][2] = {{1,2},
                         {3,4},
                         {5,6}};
  ```
- Can let assigned value define size (but only one of them!)
- `double matrix[][2] = {1,2,3,4};`
  will give you a 2x2 matrix

Task 2

- Write a program that multiplies two matrices and prints the result

Lecture 7: Basics of C

Wrap Up

Basic Datatypes and printf

Branching and Loops in C

Constant values

Arrays

Functions and return values

Functions

- Functions provide a way to encapsulate a piece of code
- Gives it a well defined input and output
- Makes code easier to read
- Often can assume the contents of a function based on its description

Functions, cont'd

- Syntax:

```
return-type function-name([arguments])
{
  declarations
  statements
}
```

- If the function does return anything you give it return-type `void`
- If you return something you leave the function with statement:

  ```
  return value;
  ```

  where `value` is of the return-type
- If the function has return-type `void` you leave with `return` if you want to leave before the function ends, otherwise you do not have to give an explicit `return`

Functions, cont'd

- NOTE: If your function has a return type and you do not have an explicit return the function will return something undefined.

return of main?

- `main` should return an `int`
- The return value can be read by whoever is calling `main` e.g. the OS
- When you have run a program in a *bash* shell you can see the return value in the special variable $?
- Ex:

```
./hello
echo $?
```

Arguments to functions

- Can pass arguments into functions like in Matlab
- `double convert_to_fahrenheit(double tempC);`
- `double convert(double in, int type);`
- The arguments become independent local variables inside function

Declaring functions

- A function just like a variable need to be declared before it is used
  - Either put the definition of the function before it is used or,
  - add a declaration of it first and then later define it
- File example:

```
#includes
#defines

function declarations

main() { ...}

function definitions
```