EL2310 – Scientific Programming

Lecture 6: Basics of C (contd.)

Ramviyas Parasuraman (ramviyas@kth.se)

Royal Institute of Technology – KTH

Overview

Lecture 6: Basics of C (contd.)

- Wrap up
- Tasks from last lecture
- Constant values
- Arrays
- Functions and return values
- Operators

Wrapup

- Analysis of the simple ”Hello World” program
- basic data types - int, char, float, double
- data type identifiers - short, long, signed/unsigned int
- get the memory size of a variable using `sizeof()`
- variable declaration and definition
  `int i,j=40;`
- `printf` statement - options such as `%d, %f %c`

Wrapup

- Branching: `if (expr) --- ; else if (expr) ---; else ---;`
- Switch case: `switch (variable) case(A) ---; case(B) ---; default ---;`
- For loop:

```
for (statement1; expression1; statement2)
{
statements;
}
```

- While loop:

```
while (expression) statements;
```

- do while loop:

```
do { statements } while (expression)
```

Task 5.2

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

Task 5.3

- 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): `%`

Task 5.4

- 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)

Task 5.5

- 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`

`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 */
...
}
```

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)`

Exercise

- What will the following do

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

Lecture 6: Basics of C (contd.)

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

Define constants

- using `const` keyword: `const double Pi = 3.18`

- 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 <identifier> <value>`
  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`

Lecture 6: Basics of C (contd.)

Arrays

- You declare an array by adding `[size]` after the variable name
- Ex: `int values[10];`
- Note: In C the index of 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 / Matrices

- 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 6.1

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

Lecture 6: Basics of C (contd.)

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

Functions, cont'd

- 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`
- 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
```

Relational operators

`>`	greater than
`>=`	greater than or equal to
`<`	less than
`<=`	less than or equal to
`==`	equal to
`!=`	not equal to

Exercise

- What will the following do?

```
i = 4;
printf("i=%d\n", i);
if (i = 1) printf("i=%d\n", i);
printf("i=%d\n", i);
```

Assignment

- Assignment returns value
- Therefore, we can assign multiple variables
- Ex: `x = y = 0;`
- Assigns from right to left