EL2310 – Scientific Programming

Lecture 8: Basics of C

Andrzej Pronobis
(pronobis@kth.se)

Royal Institute of Technology – KTH

Overview

Lecture 8: Basics of C

- Wrap Up
- Arrays
- Functions and return values
- Other tasks and useful stuff
- Strings

Last time

- `printf`
- `for, while, do-while`
- `if-else, switch`

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 8: Basics of C

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 8: Basics of C

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`
- NOTE: If your function has a return type and you do not have

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
```

Lecture 8: Basics of C

Task 3

- Write function that returns the probability to draw a certain value x given that it is from a normal distribution $\mathcal{N}(\mu, \sigma)$
- `double getprob(double x, double mean, double sigma);`
- Print a table with x and p(x)

Hint: You will have to include <math.h> and link with libm (math)

Linking to extra libraries

- Often use function defined in other libraries, such as `cos`, `sin`, `exp` from libm
- Need to tell linker that it should use libm as well
- `gcc -o mymathprg mymathprg.c -lm`

enum

- enumeration constant
- An alternative to using many `#define`
- Ex:

```
enum state { STATE_START, STATE_RUN,
STATE_STOP};
```

- First name assigned value 0, next 1, etc
- The same with `#define`

```
#define STATE_START 0
#define STATE_RUN 1
#define STATE_STOP 2
```

- Can give value to all names manually
- Unassigned names will be assigned “last + 1”

Task 4

- Test enum
- What if you add as a last item `NUMBER_OF_ITEMS` in the enum?

Lecture 8: Basics of C

`char` array: C style strings

- Ex: `char name[] = "Tulou";`
- `strlen(...)` return length of a string
- A string is terminated by `\0`
- The variable `name` will be of length 6 where last character has value `\0`

Hint: You have to include <string.h>

Task 5

- Experiment with char arrays, strlen and sizeof
- What if `char [] name= "John Smith"`, what is the string length?
- What is the array size in bytes?
- What happens if you set `name[4] = 0;`

Relational operators

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

Task 6

- 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

Task 7

- Make the following expression clear by adding parentheses

```
x = y = z = 4;
```

Precedence

Incomplete table of precedence

1. `()` `[]` `->` `.`
2. `!` `~` `--` `++` `&`
3. `*` `/` `%`
4. `+` `-`
5. `>` `>=` `<` `<=`
6. `==` `!=`
7. `&&`
8. `||`
9. `=` `+=` `-=` `...`

Evaluating logical expressions

- Logical expressions are evaluated left to right
- Guaranteed to stop as soon as expression value is determined
- A logical expression that evaluates to true is assigned value 1
- A logical expression that evaluates to false is assigned value 0

Task 8

- Write function `double atof(char s[])`
- Should take a char array as input and return a double representation of the string
- Assume that the string is a number like `-1.234` or `123.4`

Hint: Functions `isdigit, isspace` from stdlib.h are useful
`http://www.asciitable.com/`