C and Linux Programming
Eastern Washington University
Computer Science
March 30th – June 12th, 2020

Lecture 2

The Basics of C Programming

Joe Dumoulin

Apr 1, 2020

Eastern Washington University

C Programming Fundamentals

How do you write a C program?

All C programs are functions!

The function that starts a program is called **main()**.

We will have a lot to say about functions later.

Functions are made up of **variables** and **statements**.

This is very similar to java except that C has no classes.

The main() function

The main function can be declared a couple of different ways:

```
int main(void)
{
  ...
  return 0;
}
```

and

```
int main(int argc, char** argv)
{
  ...
  return 0;
}
```

Your C program starts and ends in main

The return value from main indicates success or failure of the program.

return 0 or EXIT_SUCCESS for a successful program.

return non-zero numbers or EXIT_FAILURE for a failure.

EXIT_SUCCESS and EXIT_FAILURE are constants defined in the file stdlib.h. More about this later.

Your C program starts and ends in main

Using constants, our program looks like this:

```
#include <stdlib.h>

int main(void)
{
  ...
  return EXIT_SUCCESS;
}
```

C Variable Types

Here is a complete list of numeric types for C.

- integer types:
  - short, long, long long
  - char
  - signed vs. unsigned
- floating point numbers
  - float
  - double
  - long double
- array variables are specified as one of the above types with a [ ] following the variable name. (more on this later)
- Question: Where is the string type?

Integer Variables - char

C integers have a fixed limit on their size based on the type name and whether they are declared signed or unsigned.

The size of a variable is measured in bytes.

Example: char type

```
char b;                  // a one-byte integer, often a character
signed char c;          // can hold a number -128 to 128.
unsigned char d;          // a number between 0 and 255
```

The **char** type is exactly 1 byte (8 bits). That can hold a number of size up to $2^8 - 1 = 255$.

Integer Variables - char

You can print C type variables using the function printf .

printf uses special formatting codes to print different variable types. For **char** types, the codes are:

type	limits	numeric	ASCII
char	-128 to 127	None	%c
signed char	-128 to 127	%hhi	%c
unsigned	0 to 255	%hhu	%c

char is also used to hold one-byte codes of different kinds - like characters of english and puctuation (ASCII).

```
unsigned char x = 65;
printf("unsigned char x = %hhu = %c", x, x);
>unsigned char x = 65 = A
```

Integer Variables - int

The size of **int** varies for different implementations of c. It is usually considered the "native word size" on the machine. On older systems this was usually 16 bits (2 bytes) modern systems typically use a word size of 32 bits, or 4 bytes.

```
int i;                   // a signed integer of 2 or 4 bytes.
signed int j;          // the same as above -128 to 128.
unsigned int k;          // a number between 0 and 255
```

Integer Variables - int

gcc uses 4 bytes for the size of **int**. You can find the size of **int** using the builtin function **sizeof**() which reports the size of a type in bytes on the machine.

```
#include <stdio.h>

int main(void)
{
        printf("The size of int in bytes is %li\n", sizeof(int);

        return EXIT_SUCCESS;
}
```

```
> The size of int in bytes is 4
```

Integer Variables - int

The compiler supports other integer sizes as well. These are used to control the amount of memory allocated for variables. each type has a particular size that you can control.

```
#include <stdio.h>

int main(void)
{
        printf("The size of int in bytes is %li\n", sizeof(int));
        printf("The size of short in bytes is %li\n", sizeof(short));
        printf("The size of long in bytes is %li\n", sizeof(long));
        printf("The size of long long in bytes is %li\n", sizeof(long long));
        return EXIT_SUCCESS;
}
```

```
> The size of int in bytes is 4
> The size of short in bytes is 2
> The size of long in bytes is 4
> The size of long long in bytes is 8
```

Integer Variables - int

Since the number of bytes varies, the size of the numbers these variable types can hold also varies. Also, there are codes for printing these types of integers using printf .

type	limits	code
int, signed int	-2,147,483,647 to +2,147,483,647	%i, %d
unsigned int	0 to 4,294,967,295	%u
short, signed short int	-32,767 to +32,767	%hi
unsigned short int	0 to 65,535	%hu
long, signed long int	-2,147,483,647 to +2,147,483,647	%li
unsigned long int	0 to 4,294,967,295	%lu
long long, signed long long int	-9,223,372,036,854,775,807 to +9,223,372,036,854,775,8077	%li
unsigned long long int	0 to 18,446,744,073,709,551,615	%llu

Integer Variables - bases

You can specify or print integer variables using base 8, 10, or 16

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

int main(void)
{
  int v = 10;
    printf("the base 10 (decimal) value of v is: %d\n", v);
    printf("the base 8 (octal) value of v is: %#o\n", v);
    printf("the base 16 (hexadecimal) value of v is: %#x\n", v);
    printf("\n");
  return EXIT_SUCCESS;
}
```

```
the base 10 (decimal) value of v is: 10
the base 8 (octal) value of v is: 012
the base 16 (hexadecimal) value of v is: 0xa
```

Floating Point Variables - float , double

Floating point variables hold decimal numbers with a scientific notation specially designed for computer storage. The format for floating point is specified in the document IEEE 754. Each of the floating point types has a portion of its space allocated for an **exponent** and **fraction** part of the number being represented.

Floating Point Variables - float , double

As with integer numbers, the sizes of the numbers that can be stored in floats and doubles vary.

type	limits	fraction bits	exponent bits	code
float	$\pm 1.175494e-38$ to $\pm 3.402823e+38$	24	8	f, e, g
double	$\pm 2.225074e-308$ to $\pm 1.797693e+308$	53	11	lf, le, lg
long double	$\pm 3.362103e-4932$ to $\pm 1.189731e+4932$	64	11	Lf, Le, Lg

String Variables - char [], char∗

In c, a string is a list of **char** ending in a zero (represented as '\0'). Strings are defined "statically" to be a pre-determined size. Later, we will discuss dynamic arrays and strings).

C strings are defined using the 'character' coding of the **char**. There are 127 'characters' in the ASCII standard character set. These include all the English alphabet upper and lower case, numeric digits, punctuation, space, tab ('\t'), carriage return ('\r'), and newline ('\n') 'characters'.

We have already seen some of these in the programs we have made so far. Lets look at an example.

String Variables - char [], char*

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

int main()
{
  char hello[] = "Hello World!";
  printf("%s\n", hello);
  return EXIT_SUCCESS;
}
```

The string "Hello World" is declared as an array of **char**. The size of the array is initialized to the number of characters plus one more byte. The extra byte is for the '\0' that is at the end of the string.

H	e	l	l	o		W	o	r	l	d	!	\0

String Variables - char [], char*

Another approach...

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

int main()
{
  char* hello = "Hello World!";
  printf("%s\n", hello);
  return EXIT_SUCCESS;
}
```

the **char*** method is almost equivalent to the previous method for defining a string. We will discuss the **char*** syntax when we talk about pointers in a later lecture.

Array Variables - []

In addition to arrays of text, we can define arrays of other types:

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

int main(void)
{
  char array1[] = "A nod is as good as a wink to a blind horse."
  char* array2 = array1; // array2 is the same as array1

  int array3[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
  float array4[] = {3.1415926535, 2.717};
}
```

Much more about arrays soon!

Statements and Control

C has the same types of control statements that you are familiar with in java.

In fact, The syntax of statements in java comes from the C++ programming language, which is an extension of C with classes. So java is a very similar language to the much older C language.

Statements and Control

Statements specify the operations we want to compute: for example:

- Doing math: `x+=1;`
- Looping: `while (x < 100) { ++x; }`
- Deciding: `if (i < 100) ++i; else i = j;`
- Calling a function: `printf ("%d\n", i);`

statements end with a semicolon (;) or a closing curly brace (}).

These will all be familiar from java. The syntax is almost identical.

Statements and Control - math

Math statements in c are usually assignments (setting the value of a variable) along with doing arithmatic.

```
int i, j, k;

i = 10;
j = i*7;
k = j - (10*i + 5*j);
```

Statements and Control - looping

The are two constructs for looping in C. These are similar to looping in java.

- **for** loops
- **while** loops

Statements and Control - looping with for

```c
int main(void)
{
  int i;

  for (i=0; i<10; ++i) // if there is only one line
     printf("%d ", i); // in the loop, no braces needed.
  printf("\n");

  for (; i<20; i++) {  // you can always add
     printf("%d ", i); // braces for clarity.
  }
  printf("\n");

  return EXIT_SUCCESS;
}
```

Statements and Control - looping with while

```
int main(void)
{
  int i=0;

  while (i<10)
     printf("%d ", i++);
  printf("\n");

  while (i<20) {
     printf("%d ", i);
     ++i;
  }
  printf("\n");

  return EXIT_SUCCESS;
}
```

Statements and Control - looping with do ... while

A lesser-used looping construct. Check the looping condition after the loop instead of before the loop.

```
int main(void)
{
  int i=0;

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

  return EXIT_SUCCESS;
}
```

Statements and Control - Deciding

Decision control can be done with **if**. This is really similar to java.

An **if** statement contains a numeric expression that is interpreted as being **True** or **False**. But C has no intrinsic boolean values. So the evaluation works like this:

- False means that the expression evaluates to zero.
- True means that the expression evaluates to anything but zero.

This is used a lot to evaluate the return status of functions that may fail.

```
int i = 0;  // initialize i
...         // some logic happens here
if (i==0) { // if i is zero,
...         // do something inside these parens
}
else {      // otherwise,
...         // do the thing in these parens
}
...         // etc.
```

Statements and Control - Functions

Functions allow you to define repetetive or complicated tasks outside of the main logic of your C program. All C programs are basically a compilation of functions.

```
// return_type function_name(param_type param_name, ..)
int read_file(char* file_name)  // function definition
{
  int return_value;

  ...                         // function logic goes here

  return return_value;        // you must return a value of the
                              //   type you declared at the start
}
```

Statements and Control - Example: atoi.c

This function converts an ASCII string of digits into an **int** type.

```
int atoi(char* integer)   // call this function with a string variable.
{
  int x = 0;                        // Initialize return value to 0;
  char sign = '+';                  // Initialize a character to
                                    //  capture the sign of the string.
  char* p = integer;                // Start at the first character.
  if (*p == '+' || *p == '-')       // Is it a '+' or '-'?
    sign = *p++;                    // Set the sign var and go to the
                                    //  next character in the array.
  while (*p >= '0' && *p <= '9') {  // While the character is a digit,
    x = x*10 + (*p-'0');            // Get the numeric part of the ASCII
                                    //  value and add to output times 10.
    ++p;                            // Advance to the next character.
  }
  if (sign != '+')                  // Check if the number is negative.
    x = -x;                         // If so, reverse the sign.
  return x;                         // Return x
}
```