

Programming in C

Prof John P. Morrison

© 1991-2017, Prof J.P. Morrison

A Little History

- 1972: Dennis M. Richie at AT&T
 - Systems programming language (operating systems and utilities)
 - High-level enough to make programs readable and portable; low-level enough to map onto the underlying machine.
- 1973: Richie and Ken Thompson rewrote most of UNIX O.S. in C.
 - Traditionally O.S.s were coded in assembly languages
 - Advantages: speed of implementation, maintainability and portability

A Little History

- 1977: Richie and Brian Kernigan wrote *The C Programming Language*
 - often referred to as the K&R standard (de facto)
- Portable C Compiler (PCC) standard
 - In the early days of C, the language was used primarily on UNIX systems. Even though there were different versions of UNIX available, the versions of the C compiler maintained a large degree of uniformity.
- With the emergence of personal computers many variants of C emerged, each differing in little ways

ANSI Standard

- American National Standards Institute
 - Foremost standards organisation in the US
 - Committees responsible for approving standards in a particular technical area.
- Feb 1983: James Brodie of Motorola Corporation applied to the X3 Committee (responsible for Computer and Information Processing Standards) to draft a C Standard
 - Mar 1983: the X3j11 technical committee was formed. It was composed of representative from all the major C compiler developers, as well as representatives from several companies that programmed their application in C.

Nature of C

- Reputation for being a mysterious and messy language that promotes bad programming habits.
 - C gives special meaning to many punctuation characters. This can be intimidating to the uninitiated.

```
int *(*(*x)())[5];
```

x is a pointer to a function returning a pointer to a 5-element array of pointers to int
- Relative dearth of rules.
 - Designed for experienced programmers, the C tenet is: *Trust the programmer.*
 - This results in tremendous liberty to write unusual code. This freedom can be abused to write needlessly tricky code

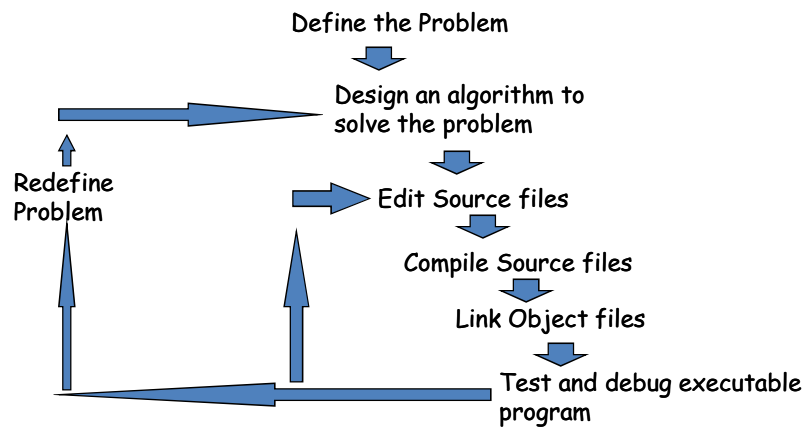
Nature of C

- C is a powerful language, but it requires self-restraint and discipline
- There is a huge difference between *good* programs and *working* programs
 - A good program not only works, but is easy to read and maintain.
- It is very possible to write good programs in C.

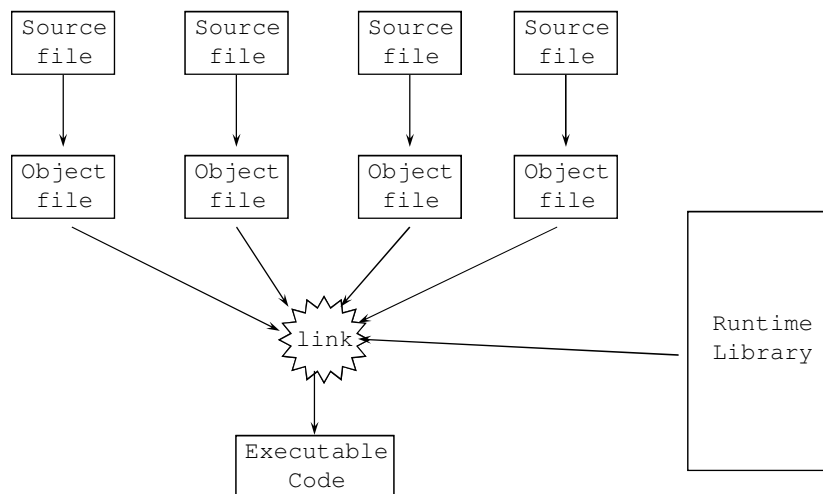
C Essentials

"A little learning is a dangerous thing." - Alexander Pope, An Essay on Criticism

Program Development



C Essentials



C Essentials

- **Compiling Source Files**

- A set of *functions* are defined in the design phase. The set of functions are typically coded in a number of *source files*. These files are input to the compiler to produce files of *object code*. The final steps are handled by two additional utilities: the *linker* and *loader*.

- **Linking Object Files**

- In addition to combining object files, the linker also links in functions from the runtime library if necessary. The result is an executable program.
- Although a separate program, the linker is invoked automatically in UNIX.

- **Loading Executable Files**

- The executable program is loaded into the computer's memory. Also performed automatically in UNIX.

C Essentials

- **The Runtime Library**

- One reason C is such a small language is that it defers many operations to a large runtime library. The RTL is a collection of object files. These are divided into groups of functions, such as I/O, memory management, math operations, and string manipulation.

- For each groups there is a source file, called a *header file*, that contains information you need to use their functions.

- By convention, the names of header files end with a *.h* extension
e.g. *stdio.h*, *stdlib.h*, *math.h*, *signal.h*, ...

To include a header file in a program, you must insert the following statement in your source file:

```
#include <filename>
```

For example, to use the I/O function `printf()`, which enables you to write to your screen, your source file must contain the line:

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

Compiling and Linking in a UNIX Environment

To compile a program, you invoke the compiler with the `gcc` command, followed by the name of the source file:

```
$ gcc test.c
```

UNIX requires the name of C source files to end with a `.c` extension. If your source file is error-free the compiler produces an object file with the same name as the source file except that it has a `.o` extension. Under UNIX the `gcc` command also invokes the linker and produces an executable file called `a.out` by default. You can override this default filename by using the `-o` option:

```
$ gcc -o test test.c
```

This forces the executable file to be named `test`. If the `gcc` command contains only one source filename, then the object file is deleted. You can, however, specify multiple source files in the same compilation command. The `gcc` program compiles each of them separately, creating an object file for each, and then it links all the object files together to create an executable file:

```
$ gcc -o test module1.c module2.c module3.c
```

This produces four files - three object files (`module1.o`, `module2.o`, `module3.o`) and an executable file called `test`.

To run the program, you enter the executable filename at the command prompt:

```
$ test
```

The loading stage is handled automatically when you execute a program.

C Essentials

• Functions

- A collection of C language operations. Adds to the operations/statements of the language. Should not be so complex that it is difficult to understand.
- Typically programs are developed with layers of functions. The lower-level functions perform the simplest operations, and higher-level functions are created by combining lower-level functions.

```
int square (int num){  
    int answer;  
    answer = num * num;  
    return answer;  
}
```

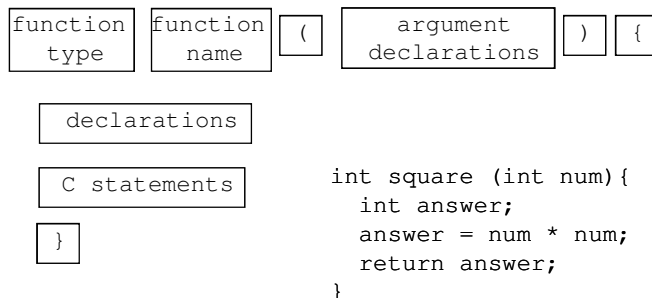
- Define once, invoke any number of times
- Abstract information

C Essentials

- The key to using functions successfully is to make them perform small pieces of a larger problem.
- Functions should be small, yet general.
- The best functions perform small autonomous tasks, but are written so that the tasks can be easily modified by changing the input.

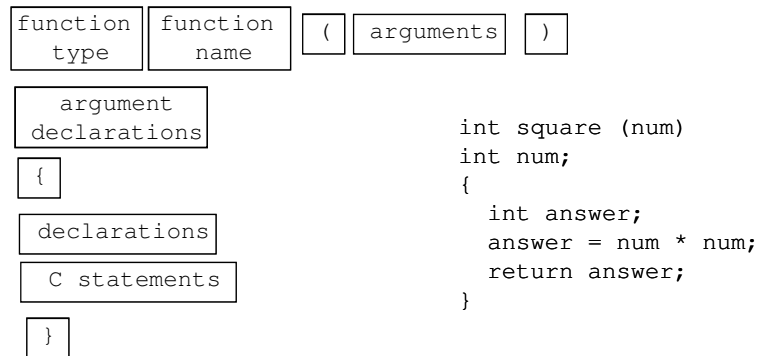
C Essentials

• Anatomy of a C Function (1)



C Essentials

- Anatomy of a C Function (2) - older form but still ANSI compliant



C Essentials

- **int** is a reserved keyword that stands for integer.
 - Keywords are always written in lowercase. C has about 30 keywords. They cannot be used as variable names.
- **square** is the name of the function
- **num** is the name of the *argument*.
 - Arguments represent data that are passed from the calling function to the function being called. On the calling side, they are known as *actual arguments*; on the called side, they are known as *formal arguments*.
 - Functions can take any number of arguments
 - Declaration of arguments: two choices.
- The function body contains all the executable statements.
 - It must begin with a left brace and end with a right brace.

C Essentials

- Declaration of the *variable answer*.
 - program variables are names for data objects whose values can be used or changed. The declaration of `answer` follows the same format as the declaration of `num`, but it lies within the function body.
- `answer = num * num;`
 - first executable statement, an assignment statement.
- `return statement`
 - causes the function to return to its caller. It may optionally return a value from the function, in this case `answer`.

C Essentials

- Variables and Constants

```
j = 5 + 10;
```

A constant is a value that never changes, whereas a variables can represent different values at different times.

A variable achieves its 'variableness' by representing a location, or address, in computer memory.

- Names
 - In C you can name just about anything: variables, constants, functions, and even locations in a program.
 - Names may contain letters, numbers, and the underscore character `_`, but must start with a letter or underscore. Names beginning with an underscore, however are generally reserved for internal system variables.

C Essentials

- *C* is case sensitive so VaR, var and VAR are all seen to be different.
- A name cannot be the same as a reserved keyword. You should also avoid using names that are used in the runtime library.
- There is no *C*-defined limit to the length of a name, although each compiler sets its own limit. The ANSI Standard requires compilers to support names of at least 31 characters.

Beware: new compilers are often built upon old linkers and these may not differentiate between names that differ only beyond a certain length.

C Essentials

- Expressions

- Any combination of objects that denotes the computation of a value.

```
5
num
5 + num
5 + num * 6
f ()
f () / 4
```

- The building blocks of expressions include variables, constants, and function calls. The building blocks themselves are expressions, but they can be combined by operators to form more complex expressions. There are dozens of operators including

+ , - , * , / .

C Essentials

- Assignment Statements

`lvalue = rvalue;`

rvalue is a value, lvalue is a place to hold a value.

- Formatting Source Files

- Style issue

```
int square( int num){int
answer; answer = num*num;return answer;}
```

```
int
square  ( int num)
{int
answer  ;
answer = num
*      num;
return answer; }
```

C Essentials

- Comments

- `/* This is a comment */`

- Comments can span multiple lines. Try to Format comments so they are readable and do not interrupt the flow of the program.

- What to comment?

- Anything that is not obvious: Complex expressions, Data Structures, and the purpose of functions.
- Comment changes, so as to keep track of modifications.
- Beware: comments without information content can make a program difficult to read. Do not comment the obvious!
- Lengthy comments cannot compensate for unreadable code.

BUG ALERT: No Nested Comments

C Essentials

- The `main()` Function

- Every executable program must contain a special function called `main()`, which is where program execution begins.

```
#include<stdlib.h>

int main(){
    extern int square(int);
    int solution;
    solution = square(5);
    exit(0);
}
```

- The rules governing `main()` are the same for every other function. `main()` can accept arguments and can return results.

C Essentials

- The `exit()` function

- is a runtime library routine that causes a program to end, returning control to the operating system. If the argument to `exit()` is zero, the program is ending normally without errors. Non-zero arguments indicate abnormal termination.
- `exit()` (or `return`) should be included in every `main()` function
- For ANSI-conforming compilers, you need to include the `stdlib` header file wherever you call `exit()`.
- The special keyword `extern` indicates that `square()` is defined elsewhere, possibly in another source file. This declaration is unnecessary if the `square()` function appears textually before the `main()` function.

C Essentials

- The `printf()` function.

- To see an answer from our program we must add a statement to display the result on our screen. The most versatile runtime routine to do this is `printf()`.

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

int main(){
extern int square(int)
int solution;
solution = square(27);
printf("%s %d\n", "The square of 34 is", solution);
exit(0);
}
```

C Essentials

- Assuming `main()` is stored in a source file called `getsquare.c`, and `square()` is located in a file called `square.c`:

```
$ gcc -o getsquare getsquare.c square.c
```

- To run the program, type `getsquare` at the prompt:

```
$ getsquare
The square of 27 is 729
$
```

- The `scanf()` Function

- `scanf()` is a mirror function of `printf()`.
- `scanf()` reads data entered from the keyboard and assigns them to variables.

C Essentials

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

int main(){
    extern int square(int)
    int solution;
    int input_val;
    printf("%s", "Enter an integer value: ");
    scanf("%d", &input_val);
    solution = square(input_val);
    printf("%s %d %s %s\n", "The square of", input_val, "is", solution);
    exit(0);
}
```

C Essentials

- *input_val* is used to store the value from the keyboard. We then pass this value as the argument to `square()`.
- The expression *&input_val* means *the address of input_val*. We pass the address of `input_val` so that `scanf()` can store a value in it.
- The `&` symbol is an important C operator.
- Typical execution of `getsquare`:

```
$ getsquare
Enter an integer value: 8
The Square of 8 is 64
$
```

C Essentials

- `printf()`
 - Can take any number of arguments.
 - The first argument is special. It is called the *format string* and it specifies how many data arguments are to follow and how they are to be formatted.
 - The format string is enclosed in quotes and may contain text and *format specifiers*. Escape sequences, such as `\n`, are used to specify unprintable characters.
 - A format specifier is a special character sequence that begins with a `%` and indicates how to write a single data item.
 - Partial list of format specifiers:

<code>%d</code> integer	<code>%o</code> octal
<code>%c</code> character	<code>%x</code> Hexadecimal
<code>%f</code> floating-point	<code>%p</code> address
<code>%s</code> string	

C Essentials

- `scanf()`
 - Like `printf()`, `scanf()` can take any number of arguments, but the first argument is a format string. It also uses many of the format specifiers.
 - The major difference between `scanf()` and `printf()` is that the data item arguments must be memory locations and they must be preceded by the *address of operator* `&`.

C Essentials

- The Preprocessor

- You can think of the C preprocessor as a separate program that runs before the actual compiler. It is automatically executed when you compile a program. The preprocessor has its own simple grammar and syntax that are only distantly related to the C language syntax. All preprocessor commands begin with the # sign and this must be the first non-space, non-tab, character on the line (Non ANSI compilers may require it to be the first character on the line).
- Preprocessor directives end with a newline, not a semicolon.

- The Include Facility

- The #include directive causes the compiler to read source text from another file as well as the file it is currently compiling. In effect, this enables you to insert the contents of one file into another file before compilation begins, although the original file is not actually altered.

C Essentials

- This is especially useful when identical information is to be shared by more than one source file. This makes program maintenance easier, since changes to shared code need only be made in one place.
- #include has two forms

`#include <filename>` and `#include "filename"`


- In Form 1, the compiler looks in a special place designated by the operating system. This is where all system include files, such as the header files for the runtime library, are kept.
- In Form 2, the compiler looks in the directory containing the source file. If it can't find the include file there, it searches for the file as if it had been specified in Form 1.
- By convention, the names of the include files usually end with a .h extension.

C Essentials

```
global_decs.h
int global_counter;
char global_char;

#include "global_decs.h"
int main(){
...
}

int global_counter;
char global_char;
int main(){
...
}
```



C Essentials

- **The #define Directive**

- Just as it is possible to associate a name with a memory location by declaring a variable, it is also possible to associate a name with a constant.

```
#define NOTHING 0
```

- NOTHING and 0 now mean the same thing to the compiler.

```
j = 5 + 0;
j = 5 + NOTHING;
```

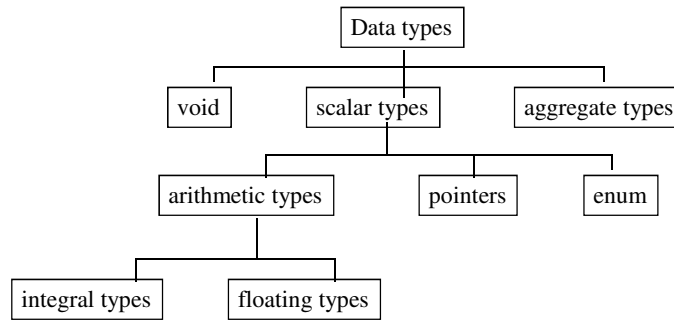
- The rules for naming constants are the same as the rules for naming variables. Beware of:

```
NOTHING = j + 5;
```

- It is common practice to use uppercase letters for constant names and lowercase letters for variables.

Scalar Data Types

What's in a name? That which we call a rose
by any other name would smell as sweet --
Shakespeare, Romeo and Juliet



Scalar Data Types

- **Declarations**

- Every variable must be declared before it is used.
- To declare `j` as an integer you would write:

```
int j;
```

- Nine reserved words for scalar data types:
`char int float double enum`
`short long signed unsigned`
- The first five are basic types, the others are qualifiers.
- As a shorthand, you can declare variables that have the same type in a single declaration by separating them with commas:

```
int j,k;
```

- All declarations in a block must appear before any executable statements. Declaration order, however, is irrelevant.

Scalar Data Types

- Declaring the return type of a function
 - Just as you declare the data type of a variable, you can declare the return type of a function:

```
float f00(int arg){  
    .....  
}
```

```
f00(int arg){  
    .....  
}
```

Here the return
type defaults to int.
Not a recommended
practice.

Scalar Data Types

- Different Types of Integer
 - The only requirement that the ANSI standard makes is that a byte must be at least eight bits long, and that ints be at least 16 bits long.
 - If size matters, you should use one of the qualifiers, *short* or *long*. On most machines a short int is two bytes, and a long int is four bytes:

```
short int j;    short j;  
long int k;     long k;
```

Type	Size(bytes)	Value range
(long) int	4	-2 ³¹ to 2 ³¹ -1
short	2	-2 ¹⁵ to 2 ¹⁵ -1
unsigned short	2	0 to 2 ¹⁶ -1
unsigned long	4	0 to 2 ³² -1
signed char	1	-2 ⁷ to 2 ⁷ -1
unsigned char	1	0 to 2 ⁸ -1

Scalar Data Types

- Characters and Integers

Most programming languages make a distinction between numeric and character data. Then number "5" is a number while the letter "A" is a letter. In C, the distinction between characters and numbers is blurred. There is a data type *char*, but it is really just a 1-byte integer value that can be used to hold either characters or numbers. After

```
char c;
```

both of the following is possible:

```
c = 'A';  
c = 65;
```

Note that character constants are enclosed in single quotes. The quotes tell the compiler to get the numeric value of the character.

Scalar Data Types

- The following program reads a character from a keyboard and then displays the code value of the character:

```
int main(){  
    char ch;  
    printf("Enter a character: ");  
    scanf("%c", &ch);  
    printf("Its numeric code value is: %d\n", ch);  
    exit(0);  
}
```

- You can perform arithmetic with characters:

```
int j;                                char to_lower(char ch){  
j = 'A' + 'B';                        return ch + 32; /* NB: ASCII dependant.  
                                     For portability  
                                     use runtime library routines  
                                     */  
}
```

Scalar Data Types

- Kinds of Integer Constants
 - Decimal, Octal, Hexadecimal

```
#include<stdio.h>
int main(){
    int num;
    printf("Enter a hexadecimal constant: ");
    scanf("%x", &num);
    printf("The decimal equivalent of %x is: %d\n", num, num);
    printf("The octal equivalent of %x is: %o\n", num, num);
    exit(0);
}
```

Scalar Data Types

- The ANSI standard states that the type of an integer constant is the first in the corresponding list in which its value can be represented

<u>Form of constant</u>	<u>List of possible types</u>
unsuffixed decimal	int, long , unsigned long
unsuffixed octal or hex	int, unsigned , long , unsigned long
suffixed by u or U	unsigned , unsigned long
suffixed by l or L	long , unsigned long

If a constant is too large to fit into the longest type in its list, the results are unpredictable. It is also possible to specifically designate that a constant have type long int by appending an l or L to the constant (L is more readable):

55L, 07777776L, -0xAAAB321L