CMSC 246 Systems Programming

Spring 2018
Bryn Mawr College
Instructor: Deepak Kumar

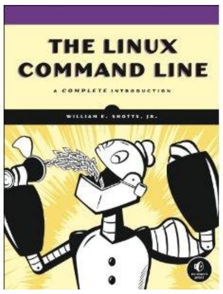
CMSC 246 Systems Programming

BACK TO THE FUTURE

Go to class web page…

Goals

- Learn Linux (CLI, not WIMP!)
- Learn C
- Learn Linux tools



THE C PROGRAMMING LANGUAGE

THE C PROGRAMMING LANGUAGE

Evolution of C

Algol60
Designed by an international committee, 1960

CPL
Combined Programming Language
Cambridge & Univ. of London, 1963
Was an attempt to bring Algol down
To earth and retail contact with the
Realities of an actual computer.
Features:
- Big
- Too many features
- Hard to learn
- Intended for numerical as well as non-numerical applications

BCPL
Basic CPL
Designed by Martin Richards, Cambridge 1967
Intended as a tool for writing compilers.
Designed to allow for separate compilation.
Features:
- Typeless language (only binary words)
- Introduced static variables
- Compact code
- Provodes access to address of data objects
- Stream-based I/O

B
Designed by Ken Thompson, Bell Labs 1970
A true forerunner of C
Features:
- Typeless (with floating pt. capabilities
- Designed for separate compilation
- Easily implementable
- Pre-processor facility
- Expensive library

Evolution of C

Algol60
Designed by an international committee, 1960

CPL
Combined Programming Language
Cambridge & Univ. of London, 1963
Was an attempt to bring Algol down
To earth and retail contact with the
Realities of an actual computer.
Features:
- Big
- Too many features
- Hard to learn
- Intended for numerical as well as non-numerical applications

BCPL
Basic CPL
Designed by Martin Richards, Cambridge 1967
Intended as a tool for writing compilers.
Designed to allow for separate compilation.
Features:
- Typeless language (only binary words)
- Introduced static variables
- Compact code
- Provodes access to address of data objects
- Stream-based I/O

B
Designed by Ken Thompson, Bell Labs 1970
A true forerunner of C
Features:
- Typeless (with floating pt. capabilities
- Designed for separate compilation
- Easily implementable
- Pre-processor facility
- Expensive library

C
1971-72
Developed at Bell Laboratories by
Ken Thompson, Dennis Ritchie, and others.
C is a by-product of UNIX.
Ritchie began to develop an extended version of B.
He called his language NB ("New B") at first.
As the language began to diverge more from B,
he changed its name to C.
The language was stable enough by 1973 that
UNIX could be rewritten in C.

K&R C
Described in Kernighan and Ritchie,
The C Programming Language (1978)
De facto standard
Features:
- Standard I/O Library
- long int data type
- Unsigned int data type
- Compound assignment operators

C89/C90
ANSI standard X3.159-1989
Completed in 1988
Formally approved in December 1989
International standard ISO/IEC 9899:1990
A superset of K&R C
Heavily influenced by C++, 1979-83
- Function prototypes
- void pointers
- Modified syntax for parameter declarations
- Remained backwards compatible with K&R C

C99
International standard ISO/IEC 9899:1999
Incorporates changes from Amendment 1 (1995)
Features:
- Inline functions
- New data types (long long int, complex, etc.)
- Variable length arrays
- Support for IEEE 754 floating point
- Single line comments using //

Onwards to C11…

7

Properties of C

- Low-level
- Small
- Permissive

8

Strengths of C

- Efficiency
- Portability
- Power
- Flexibility
- Standard library
- Integration with UNIX

Weaknesses of C

- Programs can be error-prone.
- Programs can be difficult to understand.
- Programs can be difficult to modify.

Effective Use of C

- Learn how to avoid pitfalls.
- Use software tools to make programs more reliable.
- Take advantage of existing code libraries.
- Adopt a sensible set of coding conventions.
- Avoid “tricks” and overly complex code.
- Stick to the standard.
- Try and adapt the good habits from programming in Java!
- MAKE SURE YOU WRITE YOUR OWN CODE.

11

First C Program: Hello, World!

```
#include <stdio.h>

int main(void) {
  printf(“Hello, World!.\n");
  return 0;
}
```

- This program might be stored in a file named `hello.c`.
- The file name doesn’t matter, but the `.c` extension is often required.

12

First C Program: Hello, World!

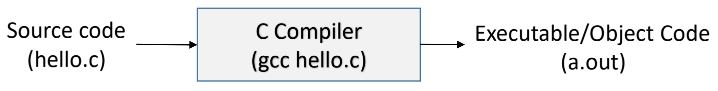
```
// Name: Xena W. Princess
// Purpose: My first C Program, prints: Hello, World!
// Written on January 22, 2018

#include <stdio.h>

int main(void) {
  printf("Hello, World!.\n");
  return 0;
} // end of main()
```

- This program might be stored in a file named `hello.c`.
- The file name doesn't matter, but the `.c` extension is often required.

13

Compilation Process

```
[xena@codewarrior cs246]$ gcc hello.c
```



```
[xena@codewarrior cs246]$ ./a.out
Hello, World!
[xena@codewarrior cs246]$
```

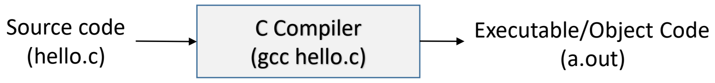
14

Excursion to Linux

Learn to use: pwd, ls, cd, cp, cat/less/more, mv

Compilation Process – GNU C Compiler

```
[xena@codewarrior cs246]$ gcc –o hello hello.c
```



```
[xena@codewarrior cs246]$ ./hello
Hello, World!
[xena@codewarrior cs246]$
```

Compilation Process

Compilation is a 3-step process

1. **Preprocessing**
   Source code commands that begin with a # are preprocessed. E.g.,

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

2. **Compiling**
   Source code is translated into object code (m/c language)

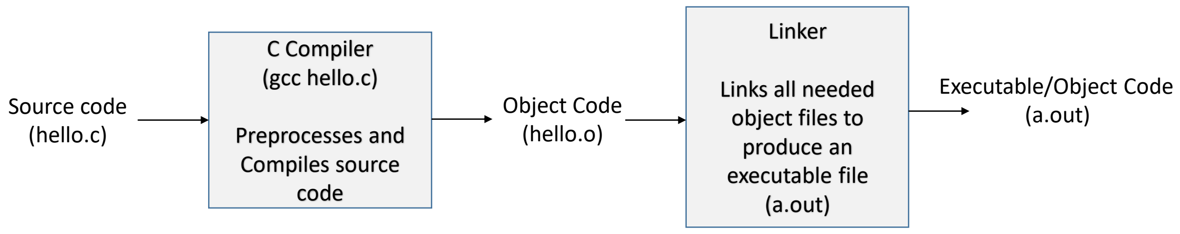
3. **Linking**
   All libraries/modules used by the program are linked to produce an executable object code

Preprocessing is normally integrated into the compiler. Linking is done by a separate program/command. The gcc command, in its simplest form, integrates all three steps.

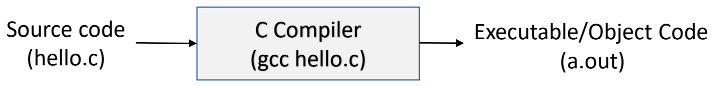
17

Compilation Process

Compilation is a 3-step process



The **gcc** command, in its simplest form, integrates all three steps.



18

C Program Structure (for now)

directives

```
int main(void) {
```
statements
```
}
```

```
#include <stdio.h>

int main(void) {
  printf(“Hello, World!.\n");
  return 0;
} // end of main()
```

C Program Structure (for now)

directives

```
int main(void) {
```
statements
```
}
```

`#include <stdio.h>`

```
int main(void) {
  printf(“Hello, World!.\n");
  return 0;
} // end of main()
```

- Before a C program is compiled, it is first edited by a preprocessor.
- Commands intended for the preprocessor are called directives.
- `<stdio.h>` is a ***header*** containing information about C’s standard I/O library.

main()

- The `main()` function is mandatory.
- `Main()` is special: it gets called automatically when the program is executed.
- `main` returns a status code; the value 0 indicates normal program termination.
- If there's no `return` statement at the end of the `main` function, many compilers will produce a warning message.

Printing Strings

- The statement

```
printf("To C, or not to C: that is the question.\n");
```

  could be replaced by two calls of `printf`:

```
printf("To C, or not to C: ");
printf("that is the question.\n");
```

- The new-line character can appear more than once in a string literal:

```
printf("Brevity is the soul of wit.\n  --Shakespeare\n");
```

Comments – Two styles /*…*/ or //

- Begins with `/*` and end with `*/`.

```
/* No comment */
```

- Comments can also be written in the following way:

```
// No comment
```

- Advantages of `//` comments:
  - Safer: there's no chance that an unterminated comment will accidentally consume part of a program.
  - Multiline comments stand out better.

23

Another Program (variables, assignment, formatted output)

```
File: small.c
#include <stdio.h>

int main(void) {

    int A, B, C;

    A = 24;
    B = 18;
    C = A + B;

    printf("C = %d\n", C);
} // main()

[xena@codewarrior cs246]$ gcc –o small small.c
[xena@codewarrior cs246]$ ./small
C = 42
[xena@codewarrior cs246]$
```

24

Printing the Value of a Variable

- `%d` works only for `int` variables; use %f to print a `float` variable
- By default, `%f` displays a number with six digits after the decimal point.
- To force `%f` to display *p* digits after the decimal point, put `.`*p* between `%` and `f`.
- To print the line

  ```
  Profit: $2150.48
  ```

  use the following call of `printf`:

  ```
  printf("Profit: $%.2f\n", profit);
  ```
- There's no limit to the number of variables that can be printed by a single call of `printf`:

  ```
  printf("Height: %d  Length: %d\n", height, length);
  ```

25

Input

- `scanf()` is the C library's counterpart to `printf`.
- Syntax for using `scanf()`

  `scanf(<`***`format-string>`***`, <`***`variable-reference(s)`***`>)`

- Example: read an integer value into an `int` variable `data`.

  ```
  scanf("%d", &data); //read an integer; store into data
  ```

- The `&` is a reference operator. More on that later!

26

Reading Input

- Reading a `float`:

  `scanf("%f", &x);`
- `"%f"` tells `scanf` to look for an input value in `float` format (the number may contain a decimal point, but doesn't have to).

27

Standard Input & Output Devices

- In Linux the standard I/O devices are, by default, the keyboard for input, and the terminal console for output.
- Thus, input and output in C, if not specified, is always from the standard input and output devices. That is,

  `printf()` always outputs to the terminal console

  `scanf()` always inputs from the keyboard
- Later, you will see how these can be reassigned/redirected to other devices.

28

Program: Convert Fahrenheit to Celsius

- The `celsius.c` program prompts the user to enter a Fahrenheit temperature; it then prints the equivalent Celsius temperature.
- Sample program output:

```
Enter Fahrenheit temperature: 212
Celsius equivalent: 100.0
```

- The program will allow temperatures that aren't integers.

29

Program: Convert Fahrenheit to Celsius `ctof.c`

```
#include <stdio.h>

int main(void)
{
  float f, c;

  printf("Enter Fahrenheit temperature: ");
  scanf("%f", &f);

  c = (f - 32) * 5.0/9.0;

  printf("Celsius equivalent: %.1f\n", c);

  return 0;
} // main()
```

Sample program output:

```
Enter Fahrenheit temperature: 212
Celsius equivalent: 100.0
```

30

Improving ctof.c

Look at the following command:

```
c = (f – 32) * 5.0/9.0;
```

First, 32, 5.0, and 9.0 should be floating point values: 32.0, 5.0, 9.0

Second, by default, in C, they will be assumed to be of type `double`
Instead, we should write

```
c = (f – 32.0f) * 5.0f/9.0f;
```

What about using constants/magic numbers?

Defining constants - macros

```
#define FREEZING_PT 32.0f
#define SCALE_FACTOR (5.0f/9.0f)
```

So we can write:

```
c = (f – FREEZING_PT) * SCALE_FACTOR;
```

When a program is compiled, the preprocessor replaces each macro by the value that it represents.
During preprocessing, the statement

```
c = (f – FREEZING_PT) * SCALE_FACTOR;
```

will become

```
c = (f – 32.f) * (5.0f/9.0f);
```

This is a safer programming practice.

Program: Convert Fahrenheit to Celsius `ctof.c`

```
#include <stdio.h>

#define FREEZING_PT 32.0f
#define SCALE_FACTOR (5.0f/9.0f)

int main(void)
{
  float f, c;

  printf("Enter Fahrenheit temperature: ");
  scanf("%f", &f);

  c = (f - FREEZING_PT) * SCALE_FACTOR;

  printf("Celsius equivalent: %.1f\n", c);

  return 0;
} // main()
```

Sample program output:

```
Enter Fahrenheit temperature: 212
Celsius equivalent: 100.0
```

Identifiers

- Names for variables, functions, macros, etc. are called ***identifiers.***
- An identifier may contain letters, digits, and underscores, but must begin with a letter or underscore:

  `times10  get_next_char  _done`

  It’s usually best to avoid identifiers that begin with an underscore.
- Examples of illegal identifiers:

  `10times  get-next-char`

Identifiers

- C is ***case-sensitive:*** it distinguishes between upper-case and lower-case letters in identifiers.
- For example, the following identifiers are all different:

```
job   joB   jOb   jOB   Job   JoB   JOb   JOB
```

- Many programmers use only lower-case letters in identifiers (other than macros), with underscores inserted for legibility:

```
symbol_table  current_page  name_and_address
```

- Other programmers use an upper-case letter to begin each word within an identifier:

```
symbolTable  currentPage  nameAndAddress
```

- C places no limit on the maximum length of an identifier.

35

Keywords

- The following ***keywords*** can't be used as identifiers:

```
auto      enum      restrict*  unsigned
break     extern    return     void
case      float     short      volatile
char      for       signed     while
const     goto      sizeof     _Bool*
continue  if        static     _Complex*
default   inline*   struct     _Imaginary*
do        int       switch
double    long      typedef
else      register  union
```

- Keywords (with the exception of `_Bool`, `_Complex`, and `_Imaginary`) must be written using only lower-case letters.
- Names of library functions (e.g., `printf`) are also lower-case.

36

If and Switch statements in C

- A compound statement has the form

  `{` *statements* `}`

- In its simplest form, the `if` statement has the form

  `if (` *expression* `)` *compound/statement*

- An `if` statement may have an `else` clause:

  `if (` *expression* `)` *compound/statement* `else` *compound/statement*

- Most common form of the `switch` statement:

```
switch ( expression ) {
  case constant-expression : statements
  …
  case constant-expression : statements
  default : statements
}
```

37

Arithmetic Operators

- C provides five binary ***arithmetic operators:***

`+`	addition
`-`	subtraction
`*`	multiplication
`/`	division
`%`	remainder

- An operator is ***binary*** if it has two operands.
- There are also two ***unary*** arithmetic operators:

`+`	unary plus
`-`	unary minus

38

Logical Expressions

- Several of C's statements must test the value of an expression to see if it is "true" or "false."
- In many programming languages, an expression such as `i < j` would have a special "Boolean" or "logical" type.
- In C, a comparison such as `i < j` yields an integer: either 0 (false) or 1 (true).

Relational Operators

- C's ***relational operators:***

`<`	less than
`>`	greater than
`<=`	less than or equal to
`>=`	greater than or equal to

- C provides two ***equality operators:***

`==`	equal to
`!=`	not equal to

- More complicated logical expressions can be built from simpler ones by using the ***logical operators:***

`!`	logical negation
`&&`	logical *and*

These operators produce 0 (false) or 1 (true) when used in expressions.

Logical Operators

- Both `&&` and `||` perform "short-circuit" evaluation: they first evaluate the left operand, then the right one.
- If the value of the expression can be deduced from the left operand alone, the right operand isn't evaluated.
- Example:

```
(i != 0) && (j / i > 0)
```

  `(i != 0)` is evaluated first. If `i` isn't equal to 0, then `(j / i > 0)` is evaluated.
- If `i` is 0, the entire expression must be false, so there's no need to evaluate `(j / i > 0)`. Without short-circuit evaluation, division by zero would have occurred.

Relational Operators & Lack of Boolean Watch out!!!

- The expression

```
i < j < k
```

  is legal, but does not test whether `j` lies between `i` and `k`.
- Since the `<` operator is left associative, this expression is equivalent to

```
(i < j) < k
```

  The 1 or 0 produced by `i < j` is then compared to `k`.
- The correct expression is `i < j && j < k`.

Loops

- The `while` statement has the form

  `while (` *expression* `)` *statement*
- General form of the `do` statement:

  `do` *statement* `while (` *expression* `) ;`
- General form of the `for` statement:

  `for (` *expr1* `;` *expr2* `;` *expr3* `)` *statement*

  expr1, *expr2*, and *expr3* are expressions.
- Example:

```
for (i = 10; i > 0; i--)
  printf("T minus %d and counting\n", i);
```

- In C99, the first expression in a `for` statement can be replaced by a declaration.
- This feature allows the programmer to declare a variable for use by the loop:

```
for (int i = 0; i < n; i++)
  …
```

STOP!!

Acknowledgements

Some content from these slides is based on the book, C Programming – A Modern Approach, By K. N. King, 2nd Edition, W. W. Norton 2008.

Some content is also included from the lecture slides provided by Prof. K. N. King. Thank You!