Crash Course in C (Part 1)

The Design of C Language Features and Data Types and their Operations and Representations
Goals of this Lecture

Help you learn about:

• The decisions that were made by the designers* of C
• ... and thereby...
• The fundamentals of C

Why?

• Learning the design rationale of the C language provides a richer understanding of C itself
• A mature programmer knows the philosophy of a language, not just the syntax

* Dennis Ritchie & subsequent members of standardization committees
The Design of C

“C is quirky, flawed, and an enormous success. While accidents of history surely helped, it evidently satisfied a need for a system implementation language efficient enough to displace assembly language, yet sufficiently abstract and fluent to describe algorithms and interactions in a wide variety of environments.”

– Dennis Ritchie

<table>
<thead>
<tr>
<th>Designers wanted C to:</th>
<th>But also:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support system programming</td>
<td>Support application programming</td>
</tr>
<tr>
<td>Be low-level</td>
<td>Be portable</td>
</tr>
<tr>
<td>Be easy for people to handle</td>
<td>Be easy for computers to handle</td>
</tr>
</tbody>
</table>
DECLARATIONS AND ASSIGNMENTS
Declaring Variables

Decision: Should C require variable declarations? (Not all languages do!)

```bash
[cmoretti@tars:]$ awk 'END {print "myVariable"; print (myOtherVariable+0)}' < /dev/null
...
0
[cmoretti@tars:]$ echo "$someNewVariable"
...
```

Thought process:

- Declaring variables allows compiler to check “spelling”
- Declaring variables allows compiler to allocate memory more efficiently
- Declaring variables’ types produces fewer surprises at runtime
- Declaring variables requires more from the programmer
  - Extra verbiage
  - Type foresight
  - “Do what I mean, not what I say”
Declaring Variables

Decisions:
- Require variable declarations
- Provide declaration statement
- Programmer specifies type of variable (and other attributes too)

Examples
- int i;
- int i, j;
- int i = 5;
- const int i = 5; /* value of i cannot change */
- static int i; /* covered later in course */
- extern int i; /* covered later in course */
Declaring Variables

Another Decision:
- Unlike Java, declaration statements in C90 must appear before any other kind of statement in compound statement

```c
{ int i;
  /* Non-declaration stmts that use i. */
  ...
  int j;
  /* Non-declaration stmts that use j. */
  ...
}
```

Illegal in C

```c
{ int i;
  int j;
  /* Non-declaration stmts that use i. */
  ...
  /* Non-declaration stmts that use j. */
  ...
}
```

Legal in C
Issue: What about assignment?

Thought process

• Must have a way to assign a value to a variable
• Many high-level languages provide an assignment statement
• Would be more expressive to define an assignment operator
  • Performs assignment, and then evaluates to a value
  • Allows assignment to appear within larger expressions

Decisions

• Provide assignment operator
  • =
    • Variable on left, expression on right
• Define assignment operator to change the value of a variable, and emit the new value of that variable
• Right-to-left associativity
Assignment Operator Examples

Examples

```
i = 0;
    /* Side effect: assign 0 to i.
        Evaluate to 0.
    */

j = i = 0;
    /* Side effect: assign 0 to i.
        Evaluate to 0.
        Side effect: assign 0 to j.
        Evaluate to 0. */

while ((i = getchar()) != EOF) ...
    /* Read a character (maybe).
        Side effect: assign that character to i.
        Evaluate to that character.
        Compare that emitted value to EOF.
        Evaluate to 0 (FALSE) or 1 (TRUE). */
```
CONTROL STATEMENTS
Control Statements: History

What the computer does “under the hood”:

```c
/* add up numbers from 1 to value in R2 */
1 R0 = 0
2 R1 = 1
3 compare R1, R2
4 if greater goto 8
5 R0 = R0 + R1
6 R1 = R1 + 1
7 goto 3
8 /* answer in R0 */
```

Early programming languages (1950’s):

```c
/* add up numbers from 1 to n */
sum = 0
i = 1
LOOP:
if (i > n) goto DONE
sum = sum + i
i = i + 1
goto LOOP
DONE:
/* answer in sum */
```

Some high-level conveniences (variable names, labels) but control flow based on `if` and `goto`
Control Statements

Algol-60 language (1960)
- BEGIN-END, IF-THEN-ELSE, WHILE-DO, FOR, (and also GOTO)

Scientific background
- Böhm and Jacopini proved (1966) that any algorithm can be expressed as the nesting of only 3 control structures:

1. **Sequence**
   - statement1
   - statement2

2. **Selection**
   - TRUE
   - condition
   - FALSE
   - statement1
   - statement2

3. **Repetition**
   - TRUE
   - condition
   - FALSE
   - statement
Thought Process

- Dijkstra argued that any algorithm should be expressed using only those control structures (Go To Statement Considered Harmful, 1968)

C language design (1972)

- Basically follow ALGOL-60, but use \{ braces \} instead of the more heavyweight BEGIN – END
Sequence Statement

Compound statement, alias block

```
{ statement1;
  statement2;
  ...
}
```
Selection Statements

**if and if-else statements**

```
if (expr)
  statement1;
else
  statement2;
```
Selection Statements

**switch** and **break** statements

- for multi-path decisions on a single integer expression

```java
switch (integerExpr)
{
    case integerLiteral1:
        ...
        break;
    case integerLiteral2:
        ...
        break;
    ...
    default:
        ...
}
```

What happens if you forget to **break**?
Repetition Statements

• while statement: test at leading edge

while (expr)
statement;

• for statement: test at leading edge, increment at trailing

for (initExpr; testExpr; incrExpr)
bodyStatement;

• do-while statement: test at trailing edge

do
statement;
while (expr);
Repetition Statements

Cascading implications

• Declarations must come at the beginning of a block → cannot declare loop control variable in for statement

```c
{  
  ...
  for (int i = 0; i < 10; i++)
      /* Do something */
  ...
}
```

Illegal in C

```c
{  
  int i;
  ...
  for (i = 0; i < 10; i++)
      /* Do something */
  ...
}
```

Legal in C
Other Control Statements

Issue: What other control statements should C provide?

Decisions

• break statement
  • Breaks out of closest enclosing switch or repetition statement
• continue statement
  • Goes back to condition check, skipping remainder of current iteration
  • When used within for, still executes increment step
• goto statement grudgingly provided
  • Jump to label

https://xkcd.com/292/
I/O
I/O Facilities

Decisions

• Do not provide I/O facilities in the language
• Instead provide I/O facilities in standard library
  • Constant: EOF
  • Data type: FILE (described later in course)
  • Variables: stdin, stdout, and stderr
  • Functions: ...

21
Reading Data Types

Issue: What functions should C provide for reading data of primitive types?

Thought process
- Must convert external form (sequence of character codes) to internal form
- Could provide getchar(), getshort(), getint(), getfloat(), etc.
- Could provide one parameterized function to read any primitive type of data

Decisions
- Provide scanf() function
- Can read any primitive type of data
- First parameter is a format string containing conversion specs

See King book for details
Issue: Should reading characters be granted special status?

Thought process

- Desirable to have a function to read a single byte from stdin
- Function must have a way to indicate failure, that is, to indicate that no bytes remain

Decisions

- Provide `getchar()` function
- Make return type of `getchar()` wider than `char`
  - Make it `int`; that's the natural word size
- Define `getchar()` to return EOF (a special non-character `int`) to indicate failure

Reminder: there is no such thing as "the EOF character"
Writing Data Types

Issue: What functions should C provide for writing data of primitive types?

Thought process
- Must convert internal form to external form (sequence of character codes)
- Could provide putchar(), putshort(), putint(), putfloat(), etc.
- Could provide one parameterized function to write any primitive type of data

Decisions
- Provide printf() function
- Can write any primitive type of data
- First parameter is a format string containing conversion specs

See King book for details
Issue: What functions should C provide for writing a character to standard output?

Thought process
- Desirable to have a function to write a single character to stdout

Decisions
- Provide a putchar() function
- Define putchar() to accept one parameter
  - For symmetry with getchar(), parameter is an int
Other I/O Facilities

**Issue:** What other I/O functions should C provide?

**Decisions**

- `fopen()`: Open a stream
- `fclose()`: Close a stream
- `fgetc()`: Read a character from specified stream
- `fputc()`: Write a character to specified stream
- `fgets()`: Read a line/string from specified stream
- `fputs()`: Write a line/string to specified stream
- `fscanf()`: Read data from specified stream
- `fprintf()`: Write data to specified stream

Described in King book, and later in the course after covering files, arrays, and strings
Java only
- Declarations anywhere within block
- Declare immutable variables with final
- Conditionals of type boolean
- “Labeled” break and continue
- No goto

C only
- Declarations only at beginning block
- Declare immutable variables with const
- Conditionals of any type (checked for zero / nonzero)
- No “labeled” break and continue
- goto provided (but don’t use it except in flattened C at end of course)
Q: Why do computer programmers confuse Christmas and Halloween?
A: Because 25 Dec == 31 Oct

NUMBER SYSTEMS
The Decimal Number System

Name
• “decem” (Latin) ⇒ ten

Characteristics
• For us, these symbols (Not universal ...)
  • 0 1 2 3 4 5 6 7 8 9
• Positional
  • 2945 ≠ 2495
  • 2945 = (2*10^3) + (9*10^2) + (4*10^1) + (5*10^0)

(Most) people use the decimal number system
The Binary Number System

**binary**

*adjective*: being in a state of one of two mutually exclusive conditions such as on or off, true or false, molten or frozen, presence or absence of a signal.

From Late Latin *bīnārius* (“consisting of two”).

**Characteristics**

- Two symbols: 0 1
- Positional: $1010_B \neq 1100_B$

**Most (digital) computers use the binary number system**

**Terminology**

- **Bit**: a single binary symbol (“binary digit”)
- **Byte**: (typically) 8 bits
- **Nybble**: 4 bits
### Decimal-Binary Equivalence

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
</tr>
<tr>
<td>16</td>
<td>10000</td>
</tr>
<tr>
<td>17</td>
<td>10001</td>
</tr>
<tr>
<td>18</td>
<td>10010</td>
</tr>
<tr>
<td>19</td>
<td>10011</td>
</tr>
<tr>
<td>20</td>
<td>10100</td>
</tr>
<tr>
<td>21</td>
<td>10101</td>
</tr>
<tr>
<td>22</td>
<td>10110</td>
</tr>
<tr>
<td>23</td>
<td>10111</td>
</tr>
<tr>
<td>24</td>
<td>11000</td>
</tr>
<tr>
<td>25</td>
<td>11001</td>
</tr>
<tr>
<td>26</td>
<td>11010</td>
</tr>
<tr>
<td>27</td>
<td>11011</td>
</tr>
<tr>
<td>28</td>
<td>11100</td>
</tr>
<tr>
<td>29</td>
<td>11101</td>
</tr>
<tr>
<td>30</td>
<td>11110</td>
</tr>
<tr>
<td>31</td>
<td>11111</td>
</tr>
</tbody>
</table>

...
Decimal-Binary Conversion

Binary to decimal: expand using positional notation

\[
100101_2 = (1 \times 2^5) + (0 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) \\
= 32 + 0 + 0 + 4 + 0 + 1 \\
= 37
\]

Least-significant bit (lsb)

Most-significant bit (msb)
(Decimal) Integer to binary: do the reverse

- Determine largest power of 2 that's \( \leq \) number; write template

\[
37 = (\square \times 2^5) + (\square \times 2^4) + (\square \times 2^3) + (\square \times 2^2) + (\square \times 2^1) + (\square \times 2^0)
\]

- Fill in template

\[
37 = (1 \times 2^5) + (0 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)
\]

\[-32 \quad 5 \quad -4 \quad 1 \quad -1 \quad 0\]

\[
100101_B
\]
**Integer-Binary Conversion**

**Integer to binary division method**
- Repeatedly divide by 2, consider remainder

\[
\begin{align*}
37 \div 2 &= 18 \text{ R } 1 \\
18 \div 2 &= 9 \text{ R } 0 \\
9 \div 2 &= 4 \text{ R } 1 \\
4 \div 2 &= 2 \text{ R } 0 \\
2 \div 2 &= 1 \text{ R } 0 \\
1 \div 2 &= 0 \text{ R } 1
\end{align*}
\]

Read from bottom to top: \(100101_B\)
The Hexadecimal Number System

Name
- “hexa-” (Ancient Greek ἕξα-) ⇒ six
- “decem” (Latin) ⇒ ten

Characteristics
- Sixteen symbols
  - 0 1 2 3 4 5 6 7 8 9 A B C D E F
- Positional
  - A13DH ≠ 3DA1H

Computer programmers often use hexadecimal or “hex”
- In C: 0x prefix (0xA13D, etc.)

Why?
### Decimal-Hexadecimal Equivalence

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>A</td>
</tr>
<tr>
<td>11</td>
<td>B</td>
</tr>
<tr>
<td>12</td>
<td>C</td>
</tr>
<tr>
<td>13</td>
<td>D</td>
</tr>
<tr>
<td>14</td>
<td>E</td>
</tr>
<tr>
<td>15</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>10</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
</tr>
<tr>
<td>24</td>
<td>18</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
</tr>
<tr>
<td>26</td>
<td>1A</td>
</tr>
<tr>
<td>27</td>
<td>1B</td>
</tr>
<tr>
<td>28</td>
<td>1C</td>
</tr>
<tr>
<td>29</td>
<td>1D</td>
</tr>
<tr>
<td>30</td>
<td>1E</td>
</tr>
<tr>
<td>31</td>
<td>1F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>20</td>
</tr>
<tr>
<td>33</td>
<td>21</td>
</tr>
<tr>
<td>34</td>
<td>22</td>
</tr>
<tr>
<td>35</td>
<td>23</td>
</tr>
<tr>
<td>36</td>
<td>24</td>
</tr>
<tr>
<td>37</td>
<td>25</td>
</tr>
<tr>
<td>38</td>
<td>26</td>
</tr>
<tr>
<td>39</td>
<td>27</td>
</tr>
<tr>
<td>40</td>
<td>28</td>
</tr>
<tr>
<td>41</td>
<td>29</td>
</tr>
<tr>
<td>42</td>
<td>2A</td>
</tr>
<tr>
<td>43</td>
<td>2B</td>
</tr>
<tr>
<td>44</td>
<td>2C</td>
</tr>
<tr>
<td>45</td>
<td>2D</td>
</tr>
<tr>
<td>46</td>
<td>2E</td>
</tr>
<tr>
<td>47</td>
<td>2F</td>
</tr>
</tbody>
</table>

... ...
Integer-Hexadecimal Conversion

Hexadecimal to (decimal) integer: expand using positional notation

\[ 25_{H} = (2 \times 16^1) + (5 \times 16^0) \]
\[ = 32 + 5 \]
\[ = 37 \]

Integer to hexadecimal: use the division method

\[ 37 \div 16 = 2 \text{ R } 5 \]
\[ 2 \div 16 = 0 \text{ R } 2 \]

Read from bottom to top: \( 25_{H} \)
Binary-Hexadecimal Conversion

Observation: $161 = 24$
- Every 1 hexit corresponds to 4 bits

Binary to hexadecimal

Digit count in binary number not a multiple of 4 ⇒ pad with zeros on left

Hexadecimal to binary

Discard leading zeros from binary number if appropriate
Base Conversion Quick Quiz

Convert binary 101010 into decimal and hex

A. 21 decimal, 1A hex  
   hint: convert to hex first

B. 42 decimal, 2A hex

C. 48 decimal, 32 hex

D. 55 decimal, 4G hex

10 1010
2 A
32 + 10 = 42
2 + 8 + 32 = 42
The Octal Number System

Name
• “octo” (Latin) ⇒ eight

Characteristics
• Eight symbols
  • 0 1 2 3 4 5 6 7
• Positional
  • $17430 \neq 73140$

Computer programmers often use octal (so does Mickey!)
• In C: 0 prefix (01743, etc.)

```
cmorettitars:tmp$ls -l myFile
-rw-r--r-- 1 cmoretti wheel 0 Sep 7 10:58 myFile

cmorettitars:tmp$chmod 755 myFile

cmorettitars:tmp$ls -l myFile
-rwxr-xr-x 1 cmoretti wheel 0 Sep 7 10:58 myFile
```