The Design of C: A Rational Reconstruction (cont.)

Goals of this Lecture

- Recall from last lecture...
- Help you learn about:
  - The decisions that were available to the designers of C
  - The decisions that were made by the designers of C
  - C!
- Why?
  - Learning the design rationale of the C language provides a richer understanding of C itself
  - ... and might be more interesting than simply learning the language itself!!!
  - A power programmer knows both the programming language and its design rationale

Character Data Types

- Issue: What character data types should C have?
- Thought process
  - The most common character codes are (were!) ASCII and EBCDIC
  - ASCII is 7-bit
  - EBCDIC is 8-bit
- Decisions
  - Provide type char
  - Type char should be one byte

Was that a good decision?
### Character Data Types (cont.)

- **Tangential Decision**
  - `char` should be an integer type
  - Can use type `char` to store small integers
  - Can do arithmetic with data of type `char`
  - Can freely mix `char` and integer data
    - `('a' + 1)` is `'b'` (assuming ASCII)
    - `('0' + 5)` is `'5'` (assuming ASCII)

  **Was that a good decision?**

### Character Constants

- **Issue:** How should C represent character constants?
- **Thought process**
  - Could represent character constants as int constants, with truncation of high-order bytes
  - More readable to use single quote syntax (`'a'`, `'b'`, etc.); but then...
    - Need special way to represent the single quote character
    - Need special ways to represent non-printable characters (e.g. newline, tab, space, etc.)
- **Decisions**
  - Provide single quote syntax
  - Use backslash to express special characters

### Character Constants (cont.)

- **Examples**
  - `'a'` the a character
  - `(char) 97` the a character
  - `(char) 0141` the a character
  - `'\0141'` the a character, octal character form
  - `'\x61'` the a character, hexadecimal character form
  - `'\0'` the null character
  - `'\a'` bell
  - `'\b'` backspace
  - `'\f'` formfeed
  - `'\n'` newline
  - `'\r'` carriage return
  - `'\t'` horizontal tab
  - `'\v'` vertical tab
  - `'\"'` backslash
  - `'\'` single quote
Strings

- **Issue**: How should C represent strings?
- **Thought process**
  - String can be represented as a sequence of chars
  - How to know where char sequence ends?
  - Store length before char sequence?
  - Store special "sentinel" char after char sequence?
  - Strings are common in systems programming
  - C language should be small/simple
  - C compiler/runtime should be lightweight

Strings (cont.)

- **Decisions**
  - Adopt a convention
    - String consists of a sequence of chars terminated with the null ("\0") character
  - Use double-quote syntax (e.g. "abc", "hello") to represent a string constant
  - Delegate remaining string handling to standard library functions
  - Easy to fix a library, hard to fix a language
- **Examples**
  - "abc" is a string constant
  - 'a' is a char constant
  - "a" is a string constant

Logical Data Type

- **Issue**: How should C represent logical data?
- **Thought process**
  - Representing a logical value (TRUE or FALSE) requires only one bit
  - Smallest entity that can be addressed is one byte
  - Type `char` is one byte, so could be used to represent logical values
  - C should be small/simple
Logical Data Type (cont.)

- Decisions
  - Don’t define a logical data type
  - Represent logical data using type `char`, or any integer type
  - Convention: 0 => FALSE, non-0 => TRUE
  - Convention used by:
    - Relational operators (<, >, etc.)
    - Logical operators (!, &&, ||)
    - Statements (if, while, etc.)

Was that a good decision? (See the next 2 slides)

Logical Data Type (cont.)

- Note
  - Using integer data to represent logical data permits shortcuts

```java
int i;
if (i) /* same as (i != 0) */
    statement1;
else
    statement2;
```

Are such shortcuts beneficial?

Logical Data Type (cont.)

- Note
  - The lack of logical data type hinders compiler’s ability to detect some errors

```java
int i;
i = 0;
if (i = 5)
    statement1;
else
    statement2;
```

What is the problem with this code?

What is the effect of this code?

How does Java handle this code?
Floating-Point Data Types

• Issue: What floating-point data types should C have?

• Thought process
  • Systems programs use floating-point data infrequently
  • But some application domains (e.g., scientific) use floating-point data often

• Decisions
  • Provide three floating-point data types: float, double, and long double
  • bytes in float <= bytes in double <= bytes in long double

• Incidentally, on hats using gcc217
  • float: 4 bytes
  • double: 8 bytes
  • long double: 12 bytes

Floating-Point Constants

• Issue: How should C represent floating-point constants?

• Thought process
  • Convenient to allow both fixed-point and scientific notation
  • Decimal is sufficient; no need for octal or hexadecimal

• Decisions
  • Any constant that contains decimal point or "E" is floating-point
  • The default floating-point type is double
  • Append "F" to indicate float
  • Append "L" to indicate long double

• Examples
  • double: 123.456, 1E-2, -1.23456E4
  • float: 123.456F, 1E-2F, -1.23456E4F
  • long double: 123.456L, 1E-2L, -1.23456E4L

Feature 2: Operators

• A high-level programming language should have operators
• Operators combine with constants and variables to form expressions
Kinds of Operators

• Issue: What kinds of operators should C have?
  • Thought process
    • Should handle typical operations
    • Should handle bit-level programming (“bit fiddling”)
  • Decisions
    • Provide typical arithmetic operators: + – * / %
    • Provide typical relational operators: == != < <= > >=
      • Each evaluates to 0=>FALSE or 1=>TRUE
    • Provide typical logical operators: ! && ||
      • Each interprets 0=>FALSE, non-0=>TRUE
      • Each evaluates to 0=>FALSE or 1=>TRUE
    • Provide bitwise operators: ~ & | ^ >> <<
    • Provide a cast operator: (type)

Assignment Operator

• 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 the assigned value
      • Allows expressions that involve assignment to appear within larger expressions
  • Decisions
    • Provide assignment operator: =
      • Define assignment operator so it changes the value of a variable,
        and also evaluates to that value

Assignment Operator (cont.)

• Examples
  
  i = 0;
  /* Assign 0 to i. Evaluate to 0.
     Discard the 0. */

  i = j = 0;
  /* Assign 0 to j. Evaluate to 0.
     Assign 0 to i. Evaluate to 0.
     Discard the 0. */

  while ((i = getchar()) != EOF) …
  /* Read a character. Assign it to i.
     Evaluate to that character.
     Compare that character to EOF.
     Evaluate to 0 (FALSE) or 1 (TRUE). */

Does the expressiveness affect clarity?
Special-Purpose Operators

• Issue: Should C provide tailored operators?

• Thought process
  • The construct \( a = b + c \) is flexible
  • The construct \( i = i + c \) is somewhat common
  • The construct \( i = i + 1 \) is very common
  • Special-purpose operators make code more expressive
    • May complicate the language and compiler
    • Might reduce some errors

• Decisions
  • Introduce += operator to do things like \( i += c \)
  • Extend to -= *= /= %= <<= >>=
  • Special-case increment and decrement: \( i++ \) \( i-- \)
  • Provide pre-inc/dec form for assignments: \( x = x + 1; y = y + 1; \)

sizeof Operator

• Issue: How can programmers determine the sizes of data?

• Thought process
  • The sizes of most primitive types are unspecified
  • C must provide a way to determine the size of a given data type programmatically

• Decisions
  • Provide a sizeof operator
    • Applied at compile-time
    • Operand can be a data type
    • Operand can be an expression, from which the compiler infers a data type

• Examples, on hats using gcc217
  • sizeof(int) evaluates to 4
  • sizeof(i) evaluates to 4 (where \( i \) is a variable of type int)
  • sizeof(i+1) evaluates to 4 (where \( i \) is a variable of type int)

Other Operators

• Issue: What other operators should C have?

• Decisions
  • Function call operator
    • Should mimic the familiar mathematical notation
    • function(param1, param2, ...)
  • Conditional operator: ?:
    • The only ternary operator
    • See King book
  • Sequence operator: ,
    • See King book
  • Pointer-related operators: & *
    • Described later in the course
  • Structure-related operators (. ->)
    • Described later in the course
Feature 3: Control Statements

• A programming language must provide **statements**
• Some statements must affect flow of control

---

Control Statements

• **Issue**: What control statements should C provide?
• **Thought process**
  • **Boehm** and **Jacopini** proved that any algorithm can be expressed as the nesting of only 3 control structures:

```
Sequence
statement1
statement2
statement1
condition
statement2
TRUE
FALSE
```

```
Selection
statement1
True
statement2
False
```

```
Repetition
statement1
while True
statement2
```

Barry Boehm

---

Control Statements (cont.)

• **Thought Process (cont.)**
  • **Dijkstra** argued that any algorithm should be expressed using only those three control structures (**GOTO Statement Considered Harmful paper**)
  • The ALGOL programming language implemented control statements accordingly

• **Decisions**
  • Provide statements to implement those 3 control structures
  • For convenience, provide a few extras

Edsger Dijkstra
Sequence and Selection

- **Sequence**
  - Compound statement, alias block
    ```
    { statement1;
      statement2;
    ...
    }
    ```
- **Selection**
  - If statement, for one-path or two-path decisions
    ```
    if (integerExpr)
    statement1;
    if (integerExpr)
    statement1;
    else
    statement2;
    ```

Selection Statements (cont.)

- **Multi-way selection**
  - Switch and break statements, for multi-path decisions
    ```
    switch (integerExpr) {
    case integerConstant1:
      ...
    case integerConstant2:
      ...
    default:
      ...
    }
    ```

What if these break statements are omitted?

Note re-use of the word break shows terseness.

Repetition Statements

- **Issue:** How should C implement repetition?
- **Decisions**
  - While statement, for general repetition
    ```
    while (integerExpr)
    statement;
    ```
  - For statement, for counting loops
    ```
    for (initialExpr; integerExpr; incrementExpr)
    statement;
    ```
  - Do-while statement, for loops with test at trailing edge
    ```
    do
    statement;
    while (integerExpr);
    ```
Other Control Statements

• Issue: What other control statements should C provide?

• Decisions
  • break statement (revisited)
    • Breaks out of closest enclosing switch or repetition statement
  • continue statement
    • Skips remainder of current loop iteration
    • Continues with next loop iteration
  • goto statement and labels
    • Avoid in general (as per Dijkstra)
    • Only (grudgingly accepted) use in error handling

Feature 4: Input/Output

• A programming language must provide facilities for reading and writing data
• Alternative: A programming environment must provide such facilities

Input/Output Facilities

• Issue: Should C provide I/O facilities?

• Thought process
  • Unix provides the stream abstraction
    • A stream is a sequence of characters
    • Unix provides 3 standard streams
      • Standard input, standard output, standard error
    • C should be able to use those streams, and others
    • I/O facilities are complex
    • C should be small/simple

• Decisions
  • Do not provide I/O facilities in C
    • Instead provide a standard library containing I/O facilities
      • Constants: EOF
      • Data types: FILE (described later in course)
      • Variables: stdin, stdout, and stderr
    • Functions: ...
Reading Characters

• Issue: What functions should C provide for reading characters from standard input?

• Thought process
  • Need function to read a single character from stdin
  • Function must have a way to indicate failure, that is, to indicate that no characters 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

• Note
  • There is no such thing as "the EOF character"

Writing Characters

• Issue: What functions should C provide for writing a character to standard output?

• Thought process
  • Need function to write a single character to stdout

• Decisions
  • Provide a putchar() function
  • Define putchar() to accept one parameter
  • For symmetry with getchar(), parameter should be an int

Reading Other Data Types

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

• Thought process
  • Must convert external form (sequence of character codes) to internal form
  • Could provide 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 specifications
  • See King book for details
Writing Other Data Types

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

• Thought process
  ▪ Must convert internal form to external form (sequence of character codes)
  ▪ Could provide `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 specifications
  ▪ See King book for details

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

Summary

• C's design goals affected decisions concerning language features:
  ▪ Data types
  ▪ Operators
  ▪ Control statements
  ▪ I/O facilities

• Knowing the design goals and how they affected the design decisions can yield a rich understanding of C

• Reading (required): C Programming (King) 8, 9, 11, 12, 13
  ▪ Reading (recommended): The Practice of Programming (Kernighan & Pike) 5, 6
  ▪ Reading (recommended): Programming with GNU Software (Loukides & Oram) 8, 8