IC150 Lecture 8
Data Types

Timothy A. Gonsalves

Data, Types, Sizes,Values

- `int, char, float, double`
- `char` – single byte, capable of holding one character
  - 2 bytes for Unicode
- Integer qualifiers – `short` and `long`
  - `short int` – 16 bits, `long int` – 32 bits
- Size of integers is compiler dependant, based on the underlying hardware:
  - `int` ≥ 16 bits, `short` ≥ 16 bits, `long` ≥ 32 bits
  - `int` is no larger than `long`, and at least as long as `short`

char, signed and unsigned

- Qualifier `signed` or `unsigned` can be applied to `int` or `char`
- `unsigned` numbers are non-negative
- `signed` char holds numbers between -128 and 127.
  - Whether `char` is signed or unsigned depends on the system. Find out on your system.
  - Print integers between 0 to 255 as characters, and integers between -128 to 127 on your system.

A *Type* is

- Set of data values, and
- Set of operations on the data, and
- Rules for manipulation

E.g. *type* `unsigned char`

- Values: 0-255
- Operations: binary +, -, *, /, %, unary +, -
- Rules: on overflow, wrap-around
  - i.e. 255+1 = 0, 255+2 = 1, …

    1 - 2 = 255, …

Number Systems

- Decimal (base 10 – uses 10 symbols {0..9})
  - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 …
- Unary (base 1)
  - 1, 11, 111, 1111, 11111 …
- Binary (base 2) – uses 2 symbols {0,1})
  - 0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010 …
- Octal (base 8 – start with a 0 in C)
  - 0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13…
  - 00, 01, 02, 03, 04, 05, 06 … C treats them as octal
- Hexadecimal (base 16 – starts with 0x in C)
  - 0, 1, …, 9, A, B, C, D, E, F, 10, 11, … 19, 1A, 1B, …

A funny infinite loop - arithmetic

This program ran into an infinite loop. I only wanted to find which numbers corresponded to which characters, the significance of signed and unsigned, which integers can be printed as characters we recognize.

Why did the infinite loop happen, how to avoid it?

```
#include<stdio.h>
main()
{char c;
  for(c=-128; c <= 127; c++)
        printf("%d -- %c \n", c, c);
}
```

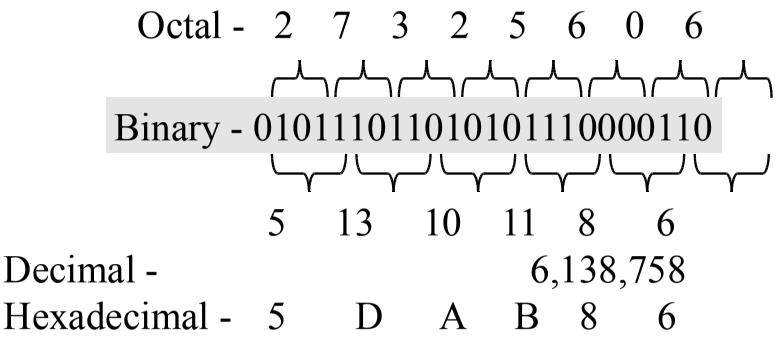
Print it as a decimal number

Print it as a character

Try omitting one parameter…

Binary, Octal and Hexadecimal

The internal representation of binary, octal, and hexadecimal numbers is similar



Float and Double

- Two types, one for single-precision arithmetic, and the other for double precision arithmetic
- `long double` is used for extended-precision arithmetic
- The size of floating pointing variables is implementation defined

Variable Initialization

- Variables may be initialized at the time of declaration:

```
#define MAXLINE 200
char esc = '\\', stop = '.';
int i = 0;
int limit = MAXLINE + 1;
float epsilon = 1.0e-5
```

- Or they may be assigned values by assignment statements in the program

```
int i;
 i = 0;
```

- Otherwise they may contain some arbitrary values

Constants

- At runtime, each variable holds a value, which changes from time to time
- A *constant* has a value that does not change

`1234` is of type `int`

`123456789L` is a `long` constant

`12345678ul` is an `unsigned long` constant

`123.4` is a floating point constant, so is `1e-2` which denotes 0.01. Their type is `double`.
Suffix by `f`, or by `l`, for `float` or `long double` types respectively

Character Constants …

- …are *integers,* written *as one character within single quotes.* Example – `'a'`, `'x'`, `'1'`, `'2'`
- The value of a character constant is the numeric value of the character in the machine's character set. For example, `'1'` has the value 49 in the ASCII character set. That is, number 49, interpreted as a character code stands for `'1'`
- Character constants can participate in arithmetic. What does `'1'+'2'` yield? (not `'3'`!) Understand this distinction. Character arithmetic is used mainly for comparisons.

Characters – escape sequences

\a	alert (bell)	\\	backslash
\b	backspace	\?	question mark
\f	formfeed	\'	single quote
\n	newline	\"	double quote
\r	carriage return	\o73	octal number
\t	horizontal tab	\x3b	hexadecimal
\v	vertical tab		

Non-printable characters

More Constants

Constant numbers, Constant characters, and

Constant Expressions – Expressions all of whose operands are constants. These can be evaluated at compile time.
Examples:

```
#define NUM_ROWS 100
#define NUM_COLS 100
#define NUM_CELLS  (NUM_ROWS * NUM_COLS)
```

`#define` is a *preprocessor directive.* Recall `#include`

Enumerated Constants

By default enum values begin with 0

- `enum boolean {No, Yes}`
  - defines two constants No = 0, and Yes = 1.
- `enum months {jan=1, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec}`
  - (jan=1) explicitly specified so starts counting from 1
- `enum escapes {BELL = '\a', BACKSPACE = '\b', TAB = '\t', NEWLINE ='\n'}`
  - more than one value can be specified

Enum and #define

- Better than `#define`, the constant values are generated for us.
  - Values from 0 onwards unless specified
  - If some values are not specified, they are obtained by increments from the last specified value
  - Variables of enum type may be declared, so the compiler can ensure type compatibility in expressions
    - but the compilers usually do not check that what you store is a valid value for the enum

Declaring Typed Constants

The qualifier `const` applied to a declaration specifies that the value will not be changed

```
const int SQ5 = 25; // SQ5 is a constant throughout
                    //      the program
```

Response to modifying SQ5 depends on the system.
Typically, a warning message is issued during compilation

```
const char MESG[] = "how are you?";
```

The character array *`MESG`* is declared as a constant which will store the string `"how are you?"`

IC150 Lecture 9

Timothy A. Gonsalves

Strings

A *string* is an array of characters terminated by the null character, `'\0'`.

A string is written in double quotes.

Example: `"This is a C string"`

`'This is rejected by C compilers'`

Only a single character can be in single quotes

`""` – empty string

Exercise: What is the difference between `'x'` and `"x"`

32 bit numbers

Internally: 4,294,967,296 (2^{32}) different permutations of 32 bits

Signed 32-bit integers vary from
-2,147,483,648 to 2,147,483,647
-2^{31} to $2^{31}-1$

Unsigned 32-bit integers vary from
0 to 4,294,967,295
0 to $2^{32}-1$

Overflow in integers…

```
#include <stdio.h>
int main()
  {
    int i = 2147483647;
    unsigned int j = 4294967295;
    printf("%d %d %d\n", i, i+1, i+2);
    printf("%u %u %u\n", j, j+1, j+2);
  }
```

Here is the result for some system:

```
2147483647 -2147483648 -2147483647
4294967295 0 1
```

Printing directives

```
#include <stdio.h> int main() {
unsigned int un = 3000000000;
                          /* system with
  32-bit int */
printf("un = %u and not %d\n", un, un);
return 0; }
```

un = 3000000000 and not -1294967296

Both have the *same* internal representation

Printing directives

```
#include <stdio.h> int main() {
short end = 200;    /* 16-bit short */
printf("end = %hd and %d\n", end, end);
return 0; }
```

short decimal

end = 200 and 200

Printing a short decimal as a normal int is okay

Printing directives

```
#include <stdio.h> int main() {
long big = 65537;
printf("big = %ld and not %hd\n", big,
  big);
return 0; }
```

big = 65537 and not 1

When the value 65537 is written in binary format as a 32-bit number, it looks like 00000000000000010000000000000001. Using the %hd specifier persuaded printf() to look at just the last 16 bits; therefore, it displayed the value as 1.

Printing directives

```
#include <stdio.h> int main() {
long long verybig = 12345678908642;
printf("verybig= %lld and not %ld\n",
  verybig, verybig);
return 0; }
```

64 bits

Truncated 32 bits

verybig= 12345678908642 and not 1942899938

Assignment and Arithmetic Rules

- Variables can be initialised when they are declared

Eg: `char c = 'a'; int j = 25;`

- Arrays can be initialised when declared:

```
char mesg[] = "hello";
int numbers[5] = {0,1,2,3,4};
```

- But an assignment to an array in the program is a syntax error: `numbers = {0,1,2,3,4};`

Functions to handle strings

String is not a basic data type => no operators!
Require functions to manipulate them.

Standard functions: `#include <string.h>`
Some examples:

a. Length of a string `len = strlen("iit");`
b. Are two strings equal?
```
if (strcmp(s, "cs101")) …
```
c. Does a given pattern occur as a substring?
d. Concatenate two strings and return the result

```
char winner[20] = "iit ";
strncat(winner, "mandi", 15);
```


Recap

- Variables
- Assignments
- relational operators (comparisons)
- Selection and repetition constructs: control structures
- Data types and their limitations
- Arrays – arrayname[n], single dimensional array
  arrayname[m][n] – 2D array, arrayname[i][j] gives the element in the i^{th} row and j^{th} coloumn

Logical Operators

- Recall relational operators {<=, <, >, >=} to compare values
- `&&` and `||`
  - Called boolean *and,* boolean *or*
  - Expressions involving these as operations take boolean values, and their operands are boolean values.
  - Called truth values also.
- E1 `&&` E2 is true if and only if both E1 and E2 are true
- E1 `||` E2 is true if and only if either E1 or E2 or both are true
- Precedence of `&&` is higher than `||`
  both are lower than relational or equality operators

How to use these in a program

- They are used when composite conditions are to be tested in decision statements.

```
if (marks >= 40 && attend > 75 ||
   marks >= 35 && donation == big)
                passed = TRUE;
```


Exercise

- Write a program which will exit when a certain number of occurrences of any keystroke is read
  1. Decide on data structures (variables)
  2. Design the procedure to solve the problem, using step-wise refinement
     - You need arrays
     - Loops
     - Loops with logical operations and so on.

Concise Operators

- Increment operator: effect is to increment value of a variable
  - `x = j++` // x gets the value of j, and then j is incremented
  - `x = ++j` // j is incremented first, then assigned to x
- Decrement operators – decrements values
  - `x = j--` //x gets the value of j, then j is decremented by 1
  - `x = --j` //j is first decremented, and then assigned to x
- Assignment operator short cut
  - E1 op= E2 is equivalent to the assignment E1 = E1 op E2
  - `x -= 5;` *same as:* `x = x - 5;`
  - Once you learn it, your code is concise
  - Matter of taste