Fundamental Data Types

CSE 130: Introduction to Programming in C
Stony Brook University

Program Organization in C

The C System

- C consists of several parts:
  - The C language
  - The preprocessor
  - The compiler
  - The library
  - Other tools (editors, debuggers, etc.)

The Preprocessor

- The preprocessor is a program that scans a source file before it is compiled
  - The preprocessor makes substitutions in the source file
- Preprocessor directives (instructions) begin with #
  - For example, `#include “stdio.h”` tells the preprocessor to replace that line with a copy of the referenced file
  - Quotes search in the current directory and other system-dependent places; < > only search in the “other places”

The Standard Library

- The standard library contains many useful functions that you can include in your C programs
  - For example, math functions, random numbers, etc.
- The C compiler knows where to find the (pre-compiled) definitions of these functions
- However, your program must still include function prototypes for any library functions that you use
  - This is generally done by `#include`-ing the appropriate header (.h) files

Example: Random Numbers

- Use the `rand()` function (found in `stdlib.h`) to generate random integer values

  ```
  printf(“%7d”, rand());
  ```

- If you put this into a program, you’ll find that your program generates the same “random” values each time it runs
- To fix this, you must “seed” the random number generator with an ever-changing value from `time.h`:

  ```
  srand(time(NULL)); /* goes at start of code block */
  ```

Fundamental Data Types

Variables

- Remember that variables are named blocks of memory
- Variables have two properties:
  - *name* — a unique identifier
  - *type* — what sort of value is stored

Identifiers

- Identifiers give unique names to various objects in a program
- An identifier may contain letters, digits, and the underscore character (‘_’)
- An identifier must begin with a letter or _
- Identifiers should be *meaningful* (and nouns)
- Style convention: the second and subsequent words in an identifier are capitalized

Identifier Examples

- Good Identifiers

```
tax_rate
taxRate
level4score
```

- Bad Identifiers

```
1stName   /* starts with a digit */
%discount /* contains invalid character */
```

Keywords

- Some words may not be used as identifiers
- These words have special meaning in C
  - C has 32 reserved words
  - Ex. for, if, while, switch

Reserved Words in C

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

Data Types

- `int` — stores integer values (ex. 5)
- `float` — stores decimal values (ex. 3.14)
- `double` — stores larger decimal values than `float` (double the precision of a float)
- `char` — stores an integer representing a character (ex.‘A’)
- Also `short`, `unsigned`, and `long`

The `char` Data Type

- C variables of any integer type (typically `char` and `int`) may be used to represent characters
  - In some cases, an `int` is required for technical reasons
  - Character constants (literals) like ‘a’ and ‘+’ are of type `int`, not `char`
- The `char` type can also hold small integers
  - `char` is stored in 1 byte (8 bits) of memory

Manipulating Characters

- Because characters are inherently integers, we can compare them using the standard relational operators
  - e.g., to test for a lowercase letter:

```
if (input >= ‘a’ && input <= ‘z’)
```

- We can also perform arithmetic on them:

```
/* convert lowercase letter to equivalent
uppercase letter */
c = c – ‘a’ + ‘A’;
```

Escape Sequences

- We can use ***escape sequences*** to print some hard-to-print characters
  - A backslash (\) changes the meaning of the character that follows it
    - e.g., \n means newline, and \t means tab

Interchangeable ints and chars

- Consider the following code fragment:

```
char c = ‘a’;
printf(“%c”, c); /* produces a */
printf(“%d”, c); /* produces 97 */

printf(“%c%c%c”, c, c+1, c+2); /* produces
abc */
```

Memory Representation

- Computer data is stored as sequences of bits (1s and 0s)
- Just like in decimal (base 10), each bit position represents a power of the base (in this case, 2):

  $2^n 2^{n-1} \ldots 2^2 2^1 2^0$

- Consider the character ‘a’, whose memory representation is 01100001:

  $0x2^7 + 1x2^6 + 1x2^5 + 0x2^4 + 0x2^3 + 0x2^2 + 0x2^1 + 1x2^0$

The `int` Data Type

- Integers are stored in different sizes of memory blocks on different platforms
  - e.g., 2 bytes (16 bit systems) or 4 bytes (32-bit systems)
  - This affects the number of values that can be stored
  - Storing too large a value can cause ***overflow***
- Beware of integer values that begin with a leading 0!
  - 0x precedes a hexadecimal value; 0 precedes an octal value

Floating-Point Types

- Use `float, double,` and `long double` to store real numbers like 0.001 and 3.14159
- Use a suffix (`f` for `float`, `l` for `long double`) to specify the type of a floating constant; otherwise, it's a `double` by default
  - e.g., `3.19f` or `4.62l`
- Exponential notation is also available, e.g. `1.234e5`

Character and Integer Types

Type	Size	Value Range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned	4 bytes	0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	8 bytes	--9223372036854775808 to 9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615

Floating-point Types

Type	Storage Size	Value Range	Precision
float	4 bytes	1.2E-38 to 3.4E+38	6 decimal
double	8 bytes	1.2E-38 to 3.4E+38	15 decimal
long double	16 bytes	3.4E-49321 to 1.2E+1049321	20 decimal

typedef

- Use **typedef** to associate a type with a mnemonic identifier

```
typedef int INCHES;
typedef char uppercase;
```

- You can then use the identifier to declare a variable or function
- **typedef** lets you abbreviate long declarations or easily redefine types when porting code to different machines

The `sizeof` Operator

- `sizeof()` returns the number of bytes needed to store an object (a type or an expression)
  - parentheses are only required when applied to a type
- sizeof(char) is always 1
- sizeof(char) <= sizeof(short) <= sizeof(int) <= sizeof(long)
- sizeof(signed) == sizeof(unsigned) == sizeof(int)
- sizeof(float) <= sizeof(double) <= sizeof(long double)

`getchar()` and `putchar()`

- These are macros from `stdio.h` that are used to read and print characters one at a time
    - They work with `int` values, not `char` values!
- `stdio.h` defines a symbolic constant named `EOF` that represents an end-of-file mark
- For example, to read one character at a time from the keyboard:

```
while ( (c = getchar() ) != EOF) { ... }
```

Mathematical Functions

- These are generally defined in **`math.h`**
  - **`sqrt()`**, **`pow()`**, **`exp()`**, **`log()`**, **`sin()`**, **`cos()`**, **`tan()`**, etc.
- Most of these functions take one argument of type **`double`**, and return a **`double`** result
- **`pow()`** takes two **`double`** arguments (base and exponent) instead
- You can use **`abs()`** (integer absolute value) and **`fabs()`** (floating-point absolute value) as well

Operators

<table>
<tr><th>Types</th><th>Operators</th></tr>
<tr><td>Arithmetic</td><td>+ - * / %</td></tr>
<tr><td>Increment/ Decrement</td><td>++
--</td></tr>
<tr><td>Assignment</td><td>= += -= *= /= %=</td></tr>
<tr><td>Relational</td><td>== < > <= >= !=</td></tr>
<tr><td>Logical</td><td>&&(AND) ||(OR) !(NOT)</td></tr>
<tr><td>Bitwise</td><td>&(AND) |(OR) ^(XOR) ~(complement) << (left shift) >> (right shift)</td></tr>
<tr><td>Ternary</td><td>:? (conditionalExpression ? expr1 : expr2)</td></tr>
</table>

Operator Precedence and Associativity

Operators	Associativity
() ++(postfix) --(postfix)	left to right
+(unary) –(unary) ++(prefix) --(prefix)	right to left
* / %	left to right
+ -	left to right
= += -= *= /= %=	right to left