- Review
- Variables and data types
- Operators
- Epilogue

Review: C Programming language

- C is a fast, small,general-purpose,platform independent programming language.
- C is used for systems programming (*e.g.,* compilers and interpreters, operating systems, database systems, microcontrollers *etc.*)
- C is static (compiled), typed, structured and imperative.
- "C is quirky, flawed, and an enormous success."–Ritchie

Review: Basics

- Variable declarations: `int i; float f;`
- Intialization: `char c='A'; int x=y=10;`
- Operators: `+,-,*,/,%`
- Expressions: `int x,y,z; x=y*2+z*3;`
- Function: `int factorial (int n); /*function takes int, returns int*/`

- Review
- Variables and data types
- Operators
- Epilogue

Definitions

Datatypes:

- The **datatype** of an object in memory determines the set of values it can have and what operations that can be performed on it.
- C is a *weakly* typed language. It allows implicit conversions as well as forced (potentially dangerous) casting.

Operators:

- **Operators** specify how an object can be manipulated (*e.g.*,, numeric vs. string operations).
- operators can be unary(*e.g.,* -,++),binary (*e.g.,* +,-,*,/),ternary (?:)

Definitions (contd.)

Expressions:

- An expression in a programming language is a combination of values, variables, operators, and functions

Variables:

- A variable is as named link/reference to a value stored in the system's memory or an expression that can be evaluated.

Consider: `int x=0,y=0; y=x+2;`.

- x, y are variables
- $y = x + 2$ is an expression
- $+$ is an operator.

Variable names

Naming rules:

- Variable names can contain letters,digits and _
- Variable names should start with letters.
- Keywords (*e.g.,* for,while *etc.*) cannot be used as variable names
- Variable names are case sensitive. `int x; int X` declares two different variables.

Pop quiz (correct/incorrect):

- `int money$owed;` (incorrect: cannot contain $)
- `int total_count` (correct)
- `int score2` (correct)
- `int 2ndscore` (incorrect: must start with a letter)
- `int long` (incorrect: cannot use keyword)

Data types and sizes

C has a small family of datatypes.

- Numeric (int,float,double)
- Character (char)
- User defined (struct,union)

Numeric data types

Depending on the precision and range required, you can use one of the following datatypes.

	signed	unsigned
short	**short int** x;**short** y;	**unsigned short** x;**unsigned short int** y;
default	**int** x;	**unsigned int** x;
long	**long** x;	**unsigned long** x;
float	**float** x;	N/A
double	**double** x;	N/A
char	**char** x; **signed char** x;	**unsigned char** x;

- The unsigned version has roughly double the range of its signed counterparts.
- Signed and unsigned characters differ only when used in arithmetic expressions.
- Titbit: Flickr changed from unsigned long $(2^{32} - 1)$ to string two years ago.

Big endian vs. little endian

The individual sizes are machine/compiler dependent. However, the following is guaranteed:

sizeof(**char**)<**sizeof**(**short**)<=**sizeof**(**int**)<=**sizeof**(**long**) and

sizeof(**char**)<**sizeof**(**short**)<=**sizeof**(**float**)<=**sizeof**(**double**)

"NUXI" problem: For numeric data types that span multiple bytes, the order of arrangement of the individual bytes is important. Depending on the device architecture, we have "big endian" and "little endian" formats.

Big endian vs. little endian (cont.)

- Big endian: the **most** significant bits (MSBs) occupy the lower address. This representation is used in the powerpc processor. Networks generally use big-endian order, and thus it is called **network order**.
- Little endian : the **least** signficant bits (LSBs) occupy the lower address. This representation is used on all x86 compatible processors.

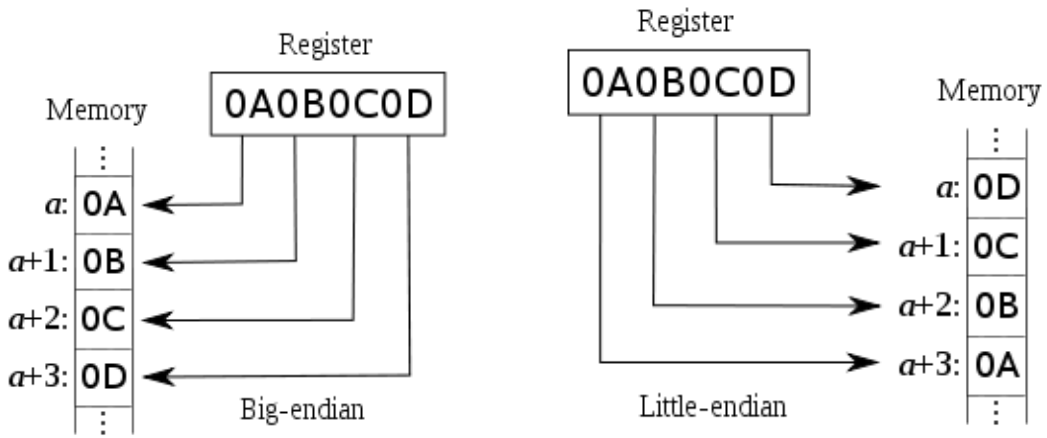

Figure: (from `http://en.wikipedia.org/wiki/Little_endian`)

Constants

Constants are literal/fixed values assigned to variables or used directly in expressions.

Datatype	example	meaning
integer	**int** i=3;	integer
	long l=3;	long integer
	unsigned long ul= 3UL;	unsigned long
	int i=0xA;	hexadecimal
	int i=012;	octal number
floating point	**float** pi=3.14159	float
	float pi=3.141F	float
	double pi=3.14159265358979323 84L	double

Constants (contd.)

Datatype	example	meaning

character	`'A'`
	`'\x41'`
	`'\0101'`
		character
		specified in hex
		specified in octal

string	`"hello world"` `"hello""world"`	string literal same as "hello world"
enumeration	**enum** BOOL {NO,YES} **enum** COLOR {R=1,G,B,Y=10}	NO=0,YES=1 G=2,B=3

Declarations

The general format for a declaration is
type variable-name [=*value*]
Examples:

- **char** x; /* uninitialized */
- **char** x='A'; /* intialized to 'A' */
- **char** x='A',y='B'; /*multiple variables initialized */
- **char** x=y='Z';/*multiple initializations */

Pop quiz II

- **int** x=017;**int** y=12; /∗is x>y?∗/
- **short int** s=0xFFFF12; /∗correct?∗/
- **char** c=−1;**unsigned char** uc=−1; /∗correct?∗/
- puts("hel"+"lo");puts("hel""lo");/∗which is correct?∗/
- **enum** sz{S=0,L=3,XL}; /∗what is the value of XL?∗/
- **enum** sz{S=0,L=−3,XL}; /∗what is the value of XL?∗/

- Review
- Variables and data types
- Operators
- Epilogue

Arithmetic operators

operator	meaning	examples
+	addition	x=3+2; /*constants*/ y+z; /*variables*/ x+y+2; /*both*/
-	subtraction	3−2; /*constants*/ **int** x=y−z; /*variables*/ y−2−z; /*both*/
*	multiplication	**int** x=3*2; /*constants*/ **int** x=y*z; /*variables*/ x*y*2; /*both*/

Arithmetic operators (contd.)

operator	meaning	examples
/	division	**float** x=3/2; /∗produces x=1 (int /) ∗/ **float** x=3.0/2 /∗produces x=1.5 (float /) ∗/ **int** x=3.0/2; /∗produces x=1 (int conversion)∗/
%	modulus (remainder)	**int** x=3%2; /∗produces x=1∗/ **int** y=7;**int** x=y%4; /∗produces 3∗/ **int** y=7;**int** x=y%10; /∗produces 7∗/

Relational Operators

Relational operators compare two operands to produce a 'boolean' result. In C any non-zero value (1 by convention) is considered to be 'true' and 0 is considered to be false.

operator	meaning	examples
>	greater than	3>2; /*evaluates to 1 */ 2.99>3 /*evaluates to 0 */
>=	greater than or equal to	3>=3; /*evaluates to 1 */ 2.99>=3 /*evaluates to 0 */
<	lesser than	3<3; /*evaluates to 0 */ 'A'<'B' /*evaluates to 1*/
<=	lesser than or equal to	3<=3; /*evaluates to 1 */ 3.99<3 /*evaluates to 0 */

Relational Operators

Testing equality is one of the most commonly used relational operator.

operator	meaning	examples
==	equal to	3==3; /*evaluates to 1 */ 'A'=='a' /*evaluates to 0 */
!=	not equal to	3!=3; /*evaluates to 0 */ 2.99!=3 /*evaluates to 1 */

Gotchas:

- Note that the "==" equality operator is different from the "=", assignment operator.
- Note that the "==" operator on float variables is tricky because of finite precision.

Logical operators

operator	meaning	examples
&&	AND	((9/3)==3) && (2*3==6); /*evaluates to 1 */ ('A'=='a') && (3==3) /*evaluates to 0 */
||	OR	2==3 || 'A'=='A'; /*evaluates to 1 */ 2.99>=3 || 0 /*evaluates to 0 */
!	NOT	!(3==3); /*evaluates to 0 */ !(2.99>=3) /*evaluates to 1 */

Short circuit: The evaluation of an expression is discontinued if the value of a conditional expression can be determined early. Be careful of any side effects in the code.

Examples:

- (3==3) || ((c=getchar())=='y'). The second expression is not evaluated.
- (0) && ((x=x+1)>0) . The second expression is not evaluated.

Increment and decrement operators

Increment and decrement are common arithmetic operation. C provides two short cuts for the same.

Postfix

- x++ is a short cut for x=x+1
- x−− is a short cut for x=x−1
- y=x++ is a short cut for y=x;x=x+1. x is evaluated **before** it is incremented.
- y=x−− is a short cut for y=x;x=x−1. x is evaluated **before** it is decremented.

Increment and decrement operators

Prefix:

- ++x is a short cut for x=x+1
- −−x is a short cut for x=x−1
- y=++x is a short cut for x=x+1;y=x;. x is evaluate **after** it is incremented.
- y=−−x is a short cut for x=x−1;y=x;. x is evaluate **after** it is decremented.

Bitwise Operators

operator	meaning	examples
&	AND	0x77 & 0x3; /*evaluates to 0x3 */ 0x77 & 0x0; /*evaluates to 0 */
|	OR	0x700 | 0x33; /*evaluates to 0x733 */ 0x070 | 0 /*evaluates to 0x070 */
^	XOR	0x770 ^ 0x773; /*evaluates to 0x3 */ 0x33 ^ 0x33; /*evaluates to 0 */
«	left shift	0x01<<4; /*evaluates to 0x10 */ 1<<2; /*evaluates to 4 */
»	right shift	0x010>>4; /*evaluates to 0x01 */ 4>>1 /*evaluates to 2 */

Notes:

- AND is true only if **both** operands are true.
- OR is true if **any** operand is true.
- XOR is true if **only one** of the operand is true.

Assignment Operators

Another common expression type found while programming in C is of the type var = var (op) expr

- x=x+1
- x=x∗10
- x=x/2

C provides compact assignment operators that can be used instead.

- x+=1 /∗is the same as x=x+1∗/
- x−=1 /∗is the same as x=x−1∗/
- x∗=10 /∗is the same as x=x∗10 ∗/
- x/=2 /∗ is the same as x=x/2
- x%=2 /∗is the same as x=x%2

Conditional Expression

A common pattern in C (and in most programming) languages is the following:

```
if (cond)
   x=<expra>;
else
   x=<exprb>;
```

C provides *syntactic sugar* to express the same using the ternary operator '?:'

```
sign=x>0?1:-1;
```

```
if (x>0)
   sign=1
else
   sign=-1
```

```
isodd=x%2==1?1:0;
```

```
if (x%2==1)
   isodd=1
else
   isodd=0
```

Notice how the ternary operator makes the code shorter and easier to understand (syntactic sugar).

- Review
- Variables and data types
- Operators
- Epilogue

Type Conversions

When variables are promoted to higher precision, data is preserved. This is automatically done by the compiler for mixed data type expressions.

```
int i;
float f;
f=i+3.14159; /*i is promoted to float, f=(float)i+3.14159*/
```

Another conversion done automatically by the compiler is 'char' $\rightarrow$ 'int'. This allows comparisons as well as manupilations of character variables.

```
isupper=(c>='A' && c<='Z')?1:0; /*c and literal constants
                                  are converted to int*/
if(!isupper)
  c=c-'a'+'A'; /*subtraction is possible
                 because of integer conversion*/
```

As a rule (with exceptions), the compiler promotes each term in an binary expression to the highest precision operand.

Precedence and Order of Evaluation

- ++,–,(cast),sizeof have the highest priority
- *,/,% have higher priority than +,-
- ==,!= have higher priority than &&,||
- assignment operators have very low priority

Use () generously to avoid ambiguities or side effects associated with precendence of operators.

- y=x∗3+2 /∗same as y=(x∗3)+2∗/
- x!=0 && y==0 /∗same as (x!=0) && (y==0)∗/
- d= c>='0' && c<='9' /∗same as d=(c>='0') && (c<='9')∗/