Types, Operators, Expressions and Files Handling in C

EECS 2031

Summer 2014

Przemyslaw Pawluk

May 13, 2014

What we will discuss today

Types, Operators and Expressions

Handling files in C

Homework

Table of Contents

Variable names

Name is ...

Combinations of letters, numbers, and underscore character (_) that

- do not start with a number
- are not a keyword

Note

Upper and lower case letters are distinct ($a \neq A$) which means that C is case sensitive

Variable names – Recommendations

- Dont begin variable names with underscore _
- Limit the length of a variable name to 31 characters or less (extraneous characters may be ignored).
- Function names, external variables: the limit may be as low as 6.
- Lower case for variable names.
- Upper case for symbolic constants
  e.g. `#define MAX_SIZE 100`
- Use short names for local variables and long names for external variables.

Data types and sizes

char
Characters (8 bits)

int
Integers (either 16 or 32 bits)

float
Single precision floating point numbers (4 bytes)

double
Double precision floating point numbers (8 bytes)

Qualifiers

- `signed char sc; /* -127  +128 */`
- `unsigned char uc; /* 0  +255 */`
- `short s; /* 16 bits, -32,768 - +32,767 */`
  - `short int s;`
- `long counter; /* 32 bits */`
  - `long int counter;`
  - `int` is either 16 or 32 bits, depending on systems.
  - `signed int sint;/* same as int sint; */`
- `unsigned int uint;`
  - 0 +4,294,967,295; assuming 4-byte int
- `long double ld; /* 12 or 16 bytes */`

Qualifiers cont.

<limits.h> and <float.h>

- symbolic constants for all of the above sizes
- other properties of the machine and compiler

Check the size

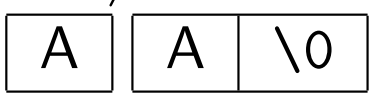
To get the size of a type, use `sizeof()`

Characters

- occupy 8 bits of space in the memory
- denoted by single quotes in the code
  `char x = 'A'`
- character string (array of characters) denoted by double quotes
  `char *str = "This is a string"`
- numerical values of ascii characters can be used
  `char c = '\012'`

Note

`'A'`$\neq$`"A"`



Constants

- Numeric constants
- Character constants
- String constants
- Constant expressions
- Enumeration constants

Numerical constants

- Decimal numbers
  `123487`
- Octal: starts with 0 (zero)
  `0654`
- Hexadecimal: starts with `0x` or `0X`
  `0x4Ab2, 0X1234`
- long int: suffixed by `L` or `l`
  `7L, 106l`
- unsigned int: suffixed by U or u
  `8U, 127u`

Floating-point Constants

```
15.75
1.575E1 /* = 15.75 */
1575e-2 /* = 15.75 */
-2.5e-3 /* = -0.0025 */
25E-4 /* = 0.0025 */
100.0L /* long double */
100.0F /* float */
```

You can omit the integer portion of the floating-point constant.

```
.0075e2
0.075e1
.075e1
75e-2
```

Note

- If there is no suffix, the type is considered **double** (8 bytes).
- To specify **float** (4 bytes), use suffix F or f.
- To specify **long double** (12 or 16 bytes), use suffix L or l.

Numeric Constants

2010
100000
729L or 729l
2010U or 2010u
20628UL or 20628ul
24.7 or 1e-2
24.7F or 24.7f
24.7L or 24.7l
037
0x1f, 0X1f, 0x1F
0XFUL

Numeric Constants

2010	int
100000	
729L or 729l	
2010U or 2010u	
20628UL or 20628ul	
24.7 or 1e-2	
24.7F or 24.7f	
24.7L or 24.7l	
037	
0x1f, 0X1f, 0x1F	
0XFUL	

Numeric Constants

2010	int
100000	taken as long if 16-bit int
729L or 729l	
2010U or 2010u	
20628UL or 20628ul	
24.7 or 1e-2	
24.7F or 24.7f	
24.7L or 24.7l	
037	
0x1f, 0X1f, 0x1F	
0XFUL	

Numeric Constants

2010	int
100000	taken as long if 16-bit int
729L or 729l	long (int)
2010U or 2010u	
20628UL or 20628ul	
24.7 or 1e-2	
24.7F or 24.7f	
24.7L or 24.7l	
037	
0x1f, 0X1f, 0x1F	
0XFUL	

Numeric Constants

2010	int
100000	taken as long if 16-bit int
729L or 729l	long (int)
2010U or 2010u	unsigned
20628UL or 20628ul	
24.7 or 1e-2	
24.7F or 24.7f	
24.7L or 24.7l	
037	
0x1f, 0X1f, 0x1F	
0XFUL	

Numeric Constants

2010	int
100000	taken as long if 16-bit int
729L or 729l	long (int)
2010U or 2010u	unsigned
20628UL or 20628ul	unsigned long
24.7 or 1e-2	
24.7F or 24.7f	
24.7L or 24.7l	
037	
0x1f, 0X1f, 0x1F	
0XFUL	

Numeric Constants

2010	int
100000	taken as long if 16-bit int
729L or 729l	long (int)
2010U or 2010u	unsigned
20628UL or 20628ul	unsigned long
24.7 or 1e-2	double
24.7F or 24.7f	
24.7L or 24.7l	
037	
0x1f, 0X1f, 0x1F	
0XFUL	

Numeric Constants

2010	int
100000	taken as long if 16-bit int
729L or 729l	long (int)
2010U or 2010u	unsigned
20628UL or 20628ul	unsigned long
24.7 or 1e-2	double
24.7F or 24.7f	float
24.7L or 24.7l	
037	
0x1f, 0X1f, 0x1F	
0XFUL	

Numeric Constants

2010	int
100000	taken as long if 16-bit int
729L or 729l	long (int)
2010U or 2010u	unsigned
20628UL or 20628ul	unsigned long
24.7 or 1e-2	double
24.7F or 24.7f	float
24.7L or 24.7l	long double
037	
0x1f, 0X1f, 0x1F	
0XFUL	

Numeric Constants

2010	int
100000	taken as long if 16-bit int
729L or 729l	long (int)
2010U or 2010u	unsigned
20628UL or 20628ul	unsigned long
24.7 or 1e-2	double
24.7F or 24.7f	float
24.7L or 24.7l	long double
037	octal (= 31 decimal)
0x1f, 0X1f, 0x1F	hexadecimal (= 31)
0XFUL	

Numeric Constants

2010	int
100000	taken as long if 16-bit int
729L or 729l	long (int)
2010U or 2010u	unsigned
20628UL or 20628ul	unsigned long
24.7 or 1e-2	double
24.7F or 24.7f	float
24.7L or 24.7l	long double
037	octal (= 31 decimal)
0x1f, 0X1f, 0x1F	hexadecimal (= 31)
0XFUL	long unsigned (= 15)

Character Constants

`'x'`	letter x
`'2'`	numeric value 50
`'\0'`	NULL char, value 0
`#define NEW_LINE '\012'`	octal, 10 in decimal
`#define NEW_LINE '\12'`	'\ooo' 1 to 3 octal digits
`#define SPACE '\x20'`	hex, 32 in decimal

Character Constants

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

String Constants

```
"hello, world\n"
"" /* empty string */
\" /* double quote character */
"hello, " " world" same as "hello, world"
```

- Strings are concatenated at compile time
- It is useful for splitting up long strings across several source lines

Constant Expressions

Expressions evaluated during compilation time. Textual replacement!

```
#define MAXLINE 1000
char line[MAXLINE+1];
```

```
#define LEAP 1 /* in leap years */
int days[31+28+LEAP+31+30+31+30+31+31+30+31+30+31];
```

Enumeration Constant

Ordered labels where first has a value 0

```
enum boolean { NO, YES };

enum months { JAN = 1, FEB, MAR, APR, MAY, JUN, JUL,
      AUG, SEP, OCT, NOV, DEC };
 /* FEB = 2, MAR = 3, etc. */
```

Note

If not all values are specified, unspecified values continue the progression from the last specified value.

const Qualifier

- Indicates that the value of a variable will not be changed
- If used with an array indicates that the elements will not be altered
- Can be used with array arguments, to indicate that the function does not change that array.

Examples:

```
const double pi = 3.1415;
const char msg[] = "alert: ";

int strlen( const char[] );
```

Limits

Why?

Knowledge of limits allows you to avoid overflow situation

Where?

Most popular limits are defined as constants in:

- `limits.h` for int, char, long etc.
- `float.h` for float and double

Arithmetic Operators

Basic operators

$+, -, *, /, \%$

Some shortcuts

$++, --, +=, -=, *=, /=$

Arithmetic Operators

Basic operators

$+, -, *, /, \%$

Some shortcuts

$++, --, +=, -=, *=, /=$

Question

What is the difference between the following expressions?

```
int x = ++i;

int x = i++;
```

Arithmetic Operators

Basic operators

$+, -, *, /, \%$

Some shortcuts

$++, --, +=, -=, *=, /=$

Question

What is the difference between the following expressions?

```
int x = ++i;
```

```
int x = i++;
```

What is the value of x assuming that i=1 in the beginning?

Precedence

Operators in order of precedence	**Associativity**
(), [], $->$, .	left to right
!, ~, ++, -, - (unary), * (indirection), & (address-of), sizeof, casts	right to left
* (multiplication), /, %	left to right
+, - (subtraction)	left to right
$<<$, $>>$	left to right
$<, <=, >=, >$	left to right
$==$, $!=$	left to right
& (bitwise and)	left to right
^	left to right
|	left to right
&&	left to right
||	left to right
?:	right to left
=, +=, -= etc.	right to left
, (comma)	left to right

Type conversion

Convert narrower to wider type

Operands	Result
int int	int
double double	double
int double	double

Examples:

$17/3 =$

$17.0/3 =$

$9/2/3.0/4 =$

Type conversion

Convert narrower to wider type

Operands	**Result**
int int	int
double double	double
int double	double

Examples:

$17/3 = 5$

$17.0/3 =$

$9/2/3.0/4 =$

Type conversion

Convert narrower to wider type

Operands	Result
int int	int
double double	double
int double	double

Examples:

$17/3 = 5$

$17.0/3 = 3.5$

$9/2/3.0/4 =$

Type conversion

Convert narrower to wider type

Operands	Result
int int	int
double double	double
int double	double

Examples:

$17/3 = 5$

$17.0/3 = 3.5$

$9/2/3.0/4 = 4/3.0/4 = 1.333/4 = 0.333$

Type conversion–Rules

Assignment type

The value of the right side is converted to the type of the left, which is the type of the result.

Truncation

Wider types are truncated e.g. `float` looses fractional part if converted into `int`

Integer truncation

Longer integers are converted to shorter ones or to chars by dropping the excess high-order bits.

Casting

The **cast** operation does not change the value, just changes the type

The **cast** operator has the same high precedence as other unary operators.

```
int A = 9, B = 2;
double x;
x = A / B; /* x is 4.0 */
x = A / (double)B; /* x is 4.5 */
```

Boolean operators

- 0 is False
- Anything else is True

You can write

Example:

```
int valid;

...

if(!valid)/*instead of valid==0*/
{
 ...
}
```

Bitwise operators

- They work on individual bits
- Useful when each bit has a different meaning
- Handy when memory was at a premium
- Handy when working with masks

&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement

Bit shifting

Shift left

`x<<y`

It is equivalent to multiplication of x by 2^y

Shift right

`x>>y`

It is equivalent to division of x by 2^y

Table of Contents

Reading from and writing to files in C

- Include `stdio.h`
- There are functions similar to those used for standard I/O
- Names usually start with f e.g. `fopen`, `fgets`, `fprintf` etc.
- The library defines a type `FILE`
- `FILE *f` is a stream that could be `stdin`, `stdout`, `stderr` or a file

Working with streams

You can read and write

```
int fprintf(FILE *f,char *fmt,);
int fscanf(FILE *f, char *fmt,);
```

Note:

`printf("hello");`
is the same as
`fprintf(stdout, "hello");`

Streams may be read-only e.g. stdin or write-only e.g. stderr

Calling `fprintf(stdin,will not work);` is illegal and will cause your program to crash!

Line-based I/O

```
char *fgets(char *s,int n,FILE *f)
```

Reads at most one less than the number of characters specified by `n` from the given stream and stores them in the string `s`

If `\n` is read, it is also stored in the string

Note:
There is no guarantee that a full line will be read!

Opening and closing files

Open

```
FILE *fopen(char *f,char *mode);
```

Creates a new stream by opening a file whose name is in `f`. If it fails, returns `NULL`

Close

```
int fclose(FILE *f);
```

Note:

When a program exits all open files are automatically closed but it is a good practice to close files that are not used anymore.
Number of file handlers in the system is limited!

Modes

The mode tells us whether we are reading or writing (it is a string)

- `"r"` - read-only file must exist
- `"w"` - write-only file is created if necessary, contents are destroyed on open
- `"a"` - append (write-only) file is created if necessary, contents preserved (write at the end of the file)

Modes

The mode tells us whether we are reading or writing (it is a string)

- `"r"` - read-only file must exist
- `"w"` - write-only file is created if necessary, contents are destroyed on open
- `"a"` - append (write-only) file is created if necessary, contents preserved (write at the end of the file)

Modes

The mode tells us whether we are reading or writing (it is a string)

- `"r"` - read-only file must exist
- `"w"` - write-only file is created if necessary, contents are destroyed on open
- `"a"` - append (write-only) file is created if necessary, contents preserved (write at the end of the file)

Modes

The mode tells us whether we are reading or writing (it is a string)

- `"r"` - read-only file must exist
- `"w"` - write-only file is created if necessary, contents are destroyed on open
- `"a"` - append (write-only) file is created if necessary, contents preserved (write at the end of the file)

Example

```
#include <stdio.h>
int main() {
 FILE *f, *g;
 char str[100];
 f = fopen("filecopy.c","r");
 g = fopen("output.c","w");
 while(fgets(str,100,f)) {
 fprintf(g,"%s",str);
 }
 fclose(f);
}
```

Stream status

- `int feof(FILE *f)` - returns non-zero if EOF has been reached on `f`, 0 otherwise
- `int feof(FILE *f)` - returns non-zero if EOF has been reached on `f`, 0 otherwise

Table of Contents

Homework

Create a simple C program that:

- Reads a file called `input.txt`
- Converts the input according to the following rules
  - Converts all lower case letters into upper case letters
  - If the character is a digit does not print it to output but computes a sum of all digits
  - If the character is a new line, prints a new line
  - For any other character it prints a '.' into output
  - For `EOF` prints new line followed by sum of digits and new line
- Writes output to a file called `output.txt`