

601.220 Intermediate Programming

C basics

Outline

- A few C basics
 - variables, assignment, data types
 - collecting input
 - arithmetic operators & precedence
- Exercise 1-1

Hello world

```
// hello_world.c:  
#include <stdio.h>  
  
// Print "Hello, world!" followed by newline and exit  
int main(void) {  
    printf("Hello, world!\n");  
    return 0;  
}
```

```
$ gcc hello_world.c -std=c99 -pedantic -Wall -Wextra  
$ ./a.out  
Hello, world!
```

We've seen `printf` to output a literal string, as in `hello_world.c`

Printing in C

- We've seen printf to output a literal string, as in `hello_world.c`
- printf allows for formatted printing of values, using placeholders in the format string
 - `printf("There are %d students in class.", 36);`
- placeholders begin with '%' and then may contain additional format information regarding field size and precision, and lastly contains a character indicating the type of data to be inserted
- the actual values corresponding to place holders are listed after the format string, 36 in this case

Printing in C

- some of the most common data type place holders:
 - d - decimal (integer type, ld for long int)
 - u - unsigned (integer type that disallows negatives, lu for long unsigned)
 - f - floating point (float, lf for double)
 - c - character
 - s - string (we'll learn more about these next week)

Variables

- `int num_students;`
- When declared, a variable gets a *type* (`int`) and *name* (`num_students`)
- A variable also has a *value* that may change throughout the program's lifetime
- To print out the value, we can use `printf`
 - `printf("There are %d students in class.", num_students);`

Types

- Integer types
 - `int`: signed integer, usually stored in 32 bits
 - `unsigned`: unsigned integer
 - `long`: signed integer with significantly greater capacity than a plain `int`
- Floating-point (decimal) types
 - `float`: single-precision floating point number
 - `double`: double-precision floating point number
- More details here:
https://en.wikipedia.org/wiki/C_data_types

Types

- Character type
 - `char`: holds a 1-byte character, `'A'`, `'B'`, `'$'`, ...
 - `chars` are basically integers, as we'll see
- Boolean type
 - `#include <stdbool.h>` to use this
 - `bool`: value can be `true` or `false`
 - Integer types can also function as bools, where 0 means `false`, non-0 means `true`
 - This is quite common, since `bool` was only introduced in C99
 - Generally, C mindset is “Booleans are just integers”

Assignment

- `num_students = 32;`
- `=` is the *assignment operator*, which modifies a variable's value

Assignment

- It is *very good practice* to declare and assign *at the same time*:
 - `int num_students = 32;`
- Generally, a variable that has been declared but not yet assigned has an “undefined” value

Aside

- **"Undefined" should strike fear into your heart**
- Programs with undefined behavior or data can (and often do) fail in mysterious ways
- Manner in which they fail might change from run to run
- We will always learn practices that avoid “undefined”

Operators

- `3 + 4`
 - 3 and 4 are operands, + is operator
 - 3 and 4 are *constants* (not variables)
- `num_students + 4`
 - `num_students` and 4 are operands, + is operator
 - `num_students` is a variable
 - A two-word variable in C such as `num_students` is often written using underscores rather than in camel case:
`numStudents`

Arithmetic operators

C operation	Arithmetic operator	Algebraic expression	C expression
Addition	+	$f + 7$	<code>f + 7</code>
Subtraction	-	$p - c$	<code>p - c</code>
Multiplication	*	bm	<code>b * m</code>
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	<code>x / y</code>
Remainder	%	$r \bmod s$	<code>r % s</code>

Fig. 2.9 | Arithmetic operators.

- Beware of integer division!
 - `7 / 2` yields 3, not 3.5

Next few examples

- Reinforce what we learned about types & operators
- Demonstrate good variable naming, operator precedence, `const`

Mysterious program

```
// mysterious.c:
#include <stdio.h>

int main(void) {
    int x = 75;
    float y = 5.0 / 9.0 * (x - 32);
    printf("%.2f", y); // print up to 2 decimal places
    return 0;
}

$ gcc mysterious.c -std=c99 -pedantic -Wall -Wextra
$ ./a.out
23.89
```

Less mysterious program

```
// convert_fc.c:
#include <stdio.h>

// Convert 75 degrees fahrenheit to celsius, print result
int main(void) {
    int fahrenheit = 75;
    float celsius = 5.0 / 9.0 * (fahrenheit - 32);
    printf("%.2f", celsius); // print up to 2 decimal places
    return 0;
}
```

- Output is correct, meaningful variable names improve readability

```
$ gcc convert_fc.c -std=c99 -pedantic -Wall -Wextra
$ ./a.out
23.89
```


Mistake?

```
// convert_fc_badprec.c:  
#include <stdio.h>  
  
// Convert 75 degrees fahrenheit to celsius, print result  
int main(void) {  
    int fahrenheit = 75;  
    float celsius = 5.0 / 9.0 * fahrenheit - 32;  
    printf("%.2f", celsius); // print up to 2 decimal places  
    return 0;  
}
```

Mistake?

```
// convert_fc_badprec.c:
#include <stdio.h>

// Convert 75 degrees fahrenheit to celsius, print result
int main(void) {
    int fahrenheit = 75;
    float celsius = 5.0 / 9.0 * fahrenheit - 32; // removed parentheses
    printf("%0.2f", celsius); // print up to 2 decimal places
    return 0;
}
```

```
$ gcc convert_fc_badprec.c -std=c99 -pedantic -Wall -Wextra
$ ./a.out
9.67
```

- Mistake because multiplication & division have higher *precedence* than subtraction

Operator precedence

C Operator	Type	Associativity
()	parentheses (function call operator)	left to right
[]	array subscript	
.	member selection via object	
->	member selection via pointer	
++	unary postincrement	
--	unary postdecrement	right to left
++	unary preincrement	
--	unary predecrement	
+	unary plus	
-	unary minus	
!	unary logical negation	
~	unary bitwise complement	
(type)	C-style unary cast	
*	dereference	
&	address	
sizeof	determine size in bytes	left to right
*	multiplication	
/	division	
%	modulus	left to right
+	addition	
-	subtraction	left to right
<<	bitwise left shift	
>>	bitwise right shift	left to right
<	relational less than	
<=	relational less than or equal to	
>	relational greater than	
>=	relational greater than or equal to	

Fig. A.1 | C operator precedence chart. (Part 1 of 2.)

Operator precedence

- More here:
`en.cppreference.com/w/c/language/operator_precedence`
- Know where to look up the rules; use parentheses when in doubt

Checkpoint Question

What output is printed by the following C program? (Note that mathematically, $9/5 = 1.8$ and $9/6 = 1.5$.)

```
#include <stdio.h>
int main(void) {
    float x = 9 / 5 + 1.0;
    printf("x = %.1f\n", x);
    return 0;
}
```

- A. x = 1.5
- B. x = 1.8
- C. x = 2.0
- D. x = 2.5
- E. x = 2.8

Using const

- Put const before the type to say a variable cannot be modified
 - `const int base = 32;`
- Compiler will catch accidental modifications

```
// convert_fc_var2.c:
```

```
#include <stdio.h>
```

```
// Convert 75 degrees fahrenheit to celsius, print result
```

```
int main(void) {
```

```
    int fahrenheit = 75;
```

```
    const int base = 32; // can't be modified
```

```
    const float factor = 5.0 / 9.0; // can't be modified
```

```
    float celsius = factor * (fahrenheit - base);
```

```
    printf("%0.2f", celsius); // print up to 2 decimal places
```

```
    return 0;
```

```
}
```

Formatted input with scanf

- The `scanf` function works similarly to the `printf` output function for reading formatted input: use a format string followed by the memory location(s) we are reading into

```
// scanf_d.c:  
int i;  
printf("Please enter an integer: ");  
scanf("%d", &i);  
printf("The value you entered is %d", i);
```

Common scanf format options (we'll see more soon)

- Use whichever code matches the type of value you want to collect
 - integer: %d
 - char: %c
 - float (real number type): %f
- The memory location you indicate you want to fill should be able to accommodate this type

Function `scanf` returns a value

- The number returned is the number of input items assigned
 - Zero typically indicates that even though input was available, the input was invalid for the specified type
 - A return value of EOF (which is -1) indicates that no input at all was available (i.e. “end of file” was reached)
 - Checking the return value can help you determine success of the scan

Live coding

- write a C program that reads two integer numbers as input and prints the sum of them

Exercise

- On the course website in “Course Materials”: find link for Exercise 1-1 and follow it
- Follow the instructions; raise your hand if you get stuck