CSSE2010/CSSE7201
Learning Lab 9

C programming

School of Information Technology and Electrical Engineering
The University of Queensland

Today

- C operator precedence
- C if statements and iteration
- C arrays
- C programming exercises
    - Look at C tutorials/documentation on Blackboard as required
    - Use Virtual-C IDE to test your code

Postfix/Prefix Increment and Decrement

- Example:

```
int a,b,c,d,e;
a = 4;
b = a++;      /* b=a; a=a+1; */
c = --a;      /* a=a-1; c=a; */
d = ++a;      /* a=a+1; d=a; */
e = a--;      /* e=a; a=a-1; */
```

Postfix – change happens after the value used

Prefix – change happens before the value used

- After these statements, values are
a=4, b=4, c=4, d=5, e=5

Operator Precedence

- Consider a + b * c
- C has strict operator precedence to disambiguate expressions like the above
- Above expression means   a + (b * c)
- Some operators associate right to left, e.g.
  - ~ ++ a means ~ (++ a)
- Most associate left to right:
  - a - b - c   means (a - b) - c not a - (b - c)

Operator Precedence and Associativity

Operators	**Associativity**
( ) [ ] -> .	Left to right
! ~ + - ++ - - & * (unary versions)	Right to left
* / %	Left to right
+ -	Left to right
<< >>	Left to right
< <= > >=	Left to right
== !=	Left to right
& (bitwise and)	Left to right
^ (bitwise xor)	Left to right
| (bitwise or)	Left to right
&& (logical and)	Left to right
|| (logical or)	Left to right
?:	Right to left
= *= /= %= += -= &= ^= |= <<= >>= (assignment)	Right to left
,	Left to right

Increasing Precedence ↑

Bit-shifting and assignment

`a << b`	means a shifted left by b bits
`a >> b`	means a shifted right by b bits
`a = b`	means a is assigned the value of b
`a += b`	is shorthand for **`a=a+b`**

- Similarly **`-=, *=, /=, %=, &=, |=, ^=, <<=, >>=`**
- Examples

`1 << 5`	is 1 * $2^5 = 32$
`3 << 4`	is 3 * $2^4 = 48$
`a += 1`	same as a++ or a=a+1

if Statement

- **`if (`**`*expression*`**`)`** *`stmt`* **`else`** *`stmt`*

- **`else`** clause is optional
- Note on expressions:
  - C interprets any 0 value as false, anything else as true
  - `if(a)` means `if(a != 0)`
  - `if(!a)` means `if(a == 0)`
- *`stmt`* can be replaced by multiple statements enclosed in braces { }

What will the code below print?

- one three
- one four
- two three
- two four

```
int a=6;
int b=4;
if(a-b)
   printf("one ");
else
   printf("two ");
if(a=b)
   printf("three\n");
else
   printf("four\n");
```

Loops

- **while (***expression***)** *stmt*
- **do** stmt **while (***expression***)**
- **for(***init_expr; test_expr; end_expr***)** *stmt*

- for loop equivalence:

```
for (init_expr; test_expr; end_expr) statement1;
```

is equivalent to:

```
init_expr;
while (test_expr) {
    statement1;
    end_expr;
}
```

What will be printed if the code below is compiled and executed?

A. 0 6
B. 0 7
C. 4 6
D. 4 7
E. 5 6
F. 5 7
G. 6 6
H. 6 7

```
int a,i;
a=0;
for(i=1;i<6;i++) {
    ++a;
}
printf(”%d %d\n”, a, i);
```

What will be printed if the code below is compiled and executed?

A. 0 6
B. 0 8
C. 1 8
D. 3 6
E. 3 8
F. 4 6
G. 4 8
H. 6 6
I. 6 8

```
int a,i;
a=0;
for(i=0;i<=6;i+=2);
{
    a++;
}
printf("%d %d\n", a, i);
```

Arrays

- Declaring an array

```
type variable_name[size];
```

  - Examples:

```
char message[16];
int values[10];
```

- Accessing elements within an array

```
variable_name[index]
```

  - index = 0 … size-1
    - This is called zero-based indexing
  - Examples:

```
message[0] = 'c';
values[9] = values[8]++;
```

Strings

- A string in C is an array of characters
  - End of string indicated by null character

Array Initialisation

- Arrays can be initialised at declaration, e.g.
  - `int values[9] = {3, 1, 4, 1, 5};`
    - if variable is global (static) – remaining elements initialised to 0
    - if variable is local (automatic) – remaining elements are uninitialised
- Size can be omitted if array is initialised, e.g.
  - `int a[] = {2,3,5,7};`
    - length is 4 in this case

Initialising String Arrays

- `char str[] = "Hello";`

Same as

- `char str[] = {'H','e','l','l','o','\0'};`

  - Array will have length 6.

C Exercises – Task 1

- Complete the CSSE2010/7201 C Programming Tutorials (1 to 3). These can be found on Blackboard under "Learning Resources" then "C Programming". You will find other C programming resources there also. The Virtual-C IDE has a built-in tutorial that helps you get started with this IDE (Integrated Development Environment)
- Virtual-C IDE – a link can be found on Blackboard Learning Resources → Software

C Exercises – Task 2(a)

Write and test a C program that calculates and prints the number of and sum of the integers between 1 and 100 (inclusive) that are **not** divisible by 3 or 5. (Hint: See the % (modulus) operator in C Tutorial 2.)

C Exercises – Task 2(b)

Start from the provided file **lab09.c**. (This program can be found on Blackboard. The program prompts for a string (max 80 characters) and converts uppercase characters to lowercase.) Modify the program to perform the following actions. Test your program after each step.

i. Convert lowercase characters to uppercase instead.
ii. Count the number of each type of letter (A to Z) and print a table showing these counts. Ignore non-alphabetic characters.
iii. Accept 20 lines of text (each up to 80 characters) instead of 1 line, then print a count of each letter seen (the total over all lines).
iv. As per (iii), but the program accepts any number of lines of text, up to when it sees a line containing only the text “END” (without the quotes).