Admin

For this week:

- lectures: variables, if, double, functions
- VLab: don’t use the web one, use via VNC
- make sure any email forwarding works ok

Variables

Variables are objects used in computation

Each variable has

- a name e.g. x, y, i, j, sum, myValue, …
- a type e.g. char, int, double, array, struct, …
- a current value e.g. 1, 3.14, ‘a’, “hello”

Variables (cont)

Variables are *declared* by specifying

- the type, the name, an initial value (optional)

E.g. int i; char ch; int x = 0; double y = 2.5;

If no initial value is given, assume random value

Integers

- often need to deal with integer values
- C provides the int type
- e.g. int count = 0; // how many …
- operations on ints: arithmetic, comparison

Integers (cont)

- reading ints: scanf(“%d”, &x);
- writing ints: printf(“%d”, x);
- %d can be qualified e.g. %10d, %4d
- %*W*d … *W* = width
- if number shorter than *W*, blank pad on left
- if number longer than *W*, write in full, no blank padding

Writing C Programs (cont)

Another problem to solve in C:

- add two numbers
  - print a message asking for the first number
  - read the first number
  - print a message asking for the second number
  - read the second number
  - add the numbers and print the sum on a line by itself

Writing C Programs (cont)

Another problem to solve in C:

- sum of squares
  - print a message asking for the first number
  - read the first number *x*
  - print a message asking for the second number
  - read the second number *y*
  - print the value of $x^2 + y^2$ on a line by itself

Writing C Programs (cont)

Another problem to solve in C:

- divide two numbers
  - print a message asking for the first number
  - read the first number
  - print a message asking for the second number
  - read the second number
  - divide the numbers and print the result

Writing C Programs (cont)

Another problem to solve in C:

- convert temperature in fahrenheit to celsius
  - print a message asking for the temperature in F
  - read in the temperature
  - convert to C $= \frac{5}{9} \times (F - 32)$
  - print value of C

Making Choices

Programs need to make choices

```
if ( some condition holds ) {
   do something
}
else {
   do something else
}
```

Making Choices (cont)

Choices can be multiway …

```
if ( Condition1 ) {
   Statements1 …
} else if ( Condition2 ) {
   Statements2 …
} else {
   StatementsN+1 …
}
```

Writing C Programs (cont)

Another problem to solve in C:

- comparing numbers
  - prompt for and read in two numbers
  - if first > second, print appropriate message
  - if first < second, print appropriate message
  - if first = second, print appropriate message

Writing C Programs (cont)

Another problem to solve in C:

- ask for the meaning of life, universe, …
  - print a message asking for The Answer
  - read in the answer
  - if 42, then print “Ahhhh! … so that’s it”
  - if non-zero, then print “Are you sure”?
  - if zero, then print “What’s that supposed to mean?”

Making Choices (cont)

Choices can be nested …

```
if ( $Condition_1$ ) {
   if ( $Condition_{1a}$ ) {
      $Statements_{1a}$ …
   else {
      $Statements_{1b}$ …
   }
   …
}
else {
   $Statements_{N+1}$ …
}
```

Writing C Programs (cont)

Another problem to solve in C:

- classifying numbers
  - prompt for and read in a number
  - if it’s < 0
    - if < 100 then big else small, and definitely negative
  - if it’s > 0
    - if > 100 then big else small, and definitely positive
  - print the number’s classification

Doubles

- often need to deal with real numbers
- C provides two types: float and double
- double is more accurate, so use it
- e.g. double height = 1.97; // metres
- operations on doubles: arithmetic, comparison

Doubles (cont)

- reading doubles: scanf(“%lf”, &x);
- writing doubles: printf(“%lf”, x);
- %lf can be qualified e.g. %6.2lf, %0.1lf
- %*W*.*P*lf … *W* = width, *P* = precision
- if number shorter than *W*, blank pad on left
- if number longer than *W*, write in full, no blank padding

Writing C Programs (cont)

Another problem to solve in C:

- convert temperature in fahrenheit to celsius
  - print a message asking for the temperature in F
  - read in the temperature
  - convert to C = $\frac{5}{9} \times (F - 32)$
  - print value of C
- this time use double rather than int

Doubles (cont)

- doubles can represent only a very small subset of the real numbers
- some real values cannot be represented exactly as a double
- arithmetic on doubles is approximate

Writing C Programs (cont)

Another problem to solve in C:

- precision check:
  - the expression

    1.0 - (0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1)

    should have the value 0
  - write a C program to check this

Admin

- no more C Lecture Stream after today
  - attend the A Stream lectures in CLB7
    (same timeslots … Tue 1-3, Wed 2-4)
  - given by Andrew Taylor, COMP1511 LiC
- VLab: don't use the web one, use via VNC
- too many lab exercises each week?
  - just do as many as possible

Writing C Programs (cont)

Another problem to solve in C:

- pythagorean identity
  - geometry tells us that $\sin^2(t) + \cos^2(t) = 1.0$
  - write a C program to check this
  - read a value for *t*
  - check the identity (e.g. $1.0 - \sin^2(t) + \cos^2(t)$ is zero)
  - write out whether the identity holds

Recap

- **int** type for counters, indexes, …
  - read int values using e.g. scanf("%d", &x);
  - write int values using e.g. printf("%5d", x);
- **double** type for measurements, …
  - read double values using e.g. scanf("%lf", &y);
  - write int values using e.g. printf("%6.2lf", y);
- scanf() returns how many %*X* were *satisfied*

#define

- #define allows us to give meaningful names to expressions and constants
- usage: #define *Name* *Expression*
- effect:
  - everywhere *Name* appears in the program
  - *Name* is replaced by *Expression*

#define (cont)

- used well, makes programs more readable
  - good usage: #define MAX_STR 100
  - poor usage: #define ONE 1
- if *Name* used multiple times in program
  - changing it only needs change in one place

Functions

A function packages a small computation

- provides ways to pass data in (parameters)
- provides ways to get data out (return value)
- contains code and *local* data objects

Examples (that you've already seen):

- scanf(), printf(), sin(), cos()

Functions provide abstraction (a VIP concept)

Functions (cont)

Functions are defined by giving

- return type, function name, parameter names/types
- and, of course, code to compute and return result

Example: function to return sum of two ints

```
int add(int x, int y) {
   return x+y;
}
```

Functions (cont)

Function signatures are defined by giving

- return type, function name, parameter types

Example: function to return sum of two ints

```
int add(int, int);
```

Users of the function need to know at least the signature, before they can use it.

Functions (cont)

Most functions return a result (of type)

- each function contains a return statement
- usage: return *Expression*;
- the result returned by the function is the value of the *Expression*

Functions (cont)

Functions are typically used like x = fun(y);

- this assigns the function result to variable x
- x's type must match function's return type

If a function does not return any result

- declare return type as void
- e.g. void vfun(int a) {…} // returns no result
- the function call is a statement: vfun(y);

Writing C Programs (cont)

Problem: print 100 messages:

- approach 1:
  - write 100 printf statements
- approach 2: use functions
  - one function prints 10 messages
  - another function calls this one 10 times
- approach 3: use a loop (next week)

Writing C Programs (cont)

Problem: Adding two numbers (again):

- get the first number
- get the second number
- print the sum of the two numbers
- use functions for the above operations

Writing C Programs (cont)

Problem: giving speeding tickets

- get the type of licence (L, P, Full)
- get the recorded speed
- get the local speed limit
- work out whether a speeding ticket is given

Writing C Programs (cont)

Additional info for the speeding problem:

- L-Plate drivers limited to max 80kmh
  - or the local speed limit, whichever is the lower
- P-Plate drivers limited to max 100kmh
  - or the local speed limit, whichever is the lower
- for Full drivers, allow +5km above limit
  - except in School Zones (40kmh hard limit)

Functions (cont)

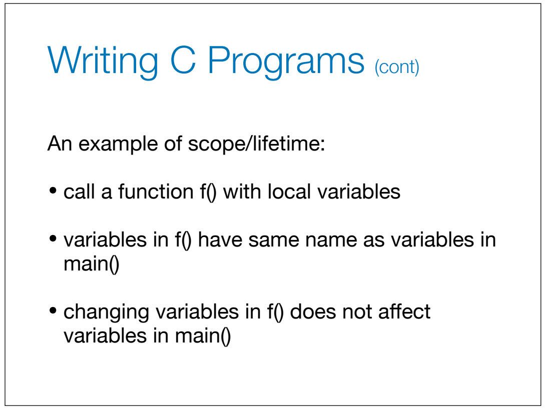
Functions introduce notions of

- scope … where is an object visible?
- lifetime … how long does an object exist?

Functions (cont)

Variables defined within a function …

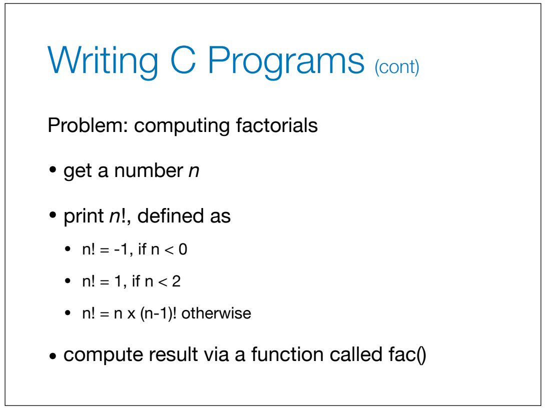
- are only visible within that function
- only exist while the function is executing
  - are created when the function is called
  - are removed when the function returns



Writing C Programs (cont)

An example of scope/lifetime:

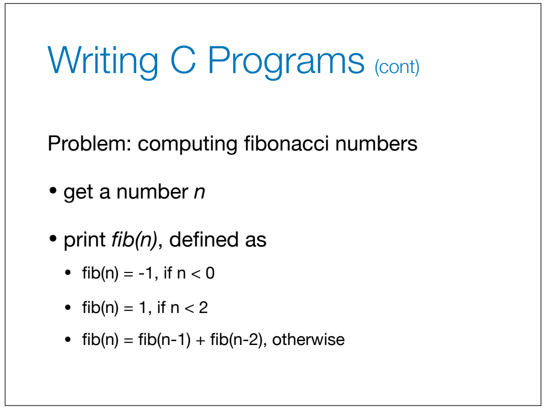
- call a function f() with local variables
- variables in f() have same name as variables in main()
- changing variables in f() does not affect variables in main()



Writing C Programs (cont)

Problem: computing factorials

- get a number *n*
- print *n*!, defined as
  - n! = -1, if n < 0
  - n! = 1, if n < 2
  - n! = n x (n-1)! otherwise
- compute result via a function called fac()



Writing C Programs (cont)

Problem: computing fibonacci numbers

- get a number *n*
- print *fib(n)*, defined as
  - fib(n) = -1, if n < 0
  - fib(n) = 1, if n < 2
  - fib(n) = fib(n-1) + fib(n-2), otherwise