C

About the crash course

Cover sufficient C for simple programs:

- variables and statements
- control
- functions
- arrays and strings
- pointers

Slides and captured lecture (video and sound) are available at:

`www.cim.mcgill.ca/~jer/C/`

Your first C program

```
/* A simple example program */

#include <stdio.h>

main ()
{
   printf ("Hello, world!\n");
}
```

Declarations

Example:

```
float x;
double d = 5;
int *p, i, a[100];
char s[21];
```

Syntax:

```
type variable_name, ... [= value];
```

Rules:

- declarations must precede executable statements
- int type may be modified by: long, short, unsigned

Changing Variable Values

Example:

```
int x, y, z;
x = 2;
y = z = 4 + 5;
x = x + 1;
```

Getting Fancy

```
x += 1;
x++;
++x;
y = x--;
```

Note:

- assignment statements return value, which may or may not be ignored; same goes for increment statements

Formatted Output

Example:

```
#include <stdio.h>
int i = 10;
float f = 2.5;
char s [] = "hi";
printf ("integer i = %d", i);
printf ("float f = %f", f);
printf ("string s = %s", s);
```

Syntax:

```
printf (string_with_formatting, var1, var2,...);
```

- Formats: %d integer, %f float, %c character, %s string, …
- #include <stdio.h> is compulsory; more about it later

Formatted Input

Example:

```
#include <stdio.h>
int i;
float f;
scanf ("%d %f\n", &i, &f);
   /* inputs an integer and a float */
```

Syntax:

```
scanf (string_with_formatting, &var1, &var2,...);
```

Note:

- the ampersand (&) is necessary because scanf modifies the variable values; more about this later

I/O Example

What does this print?

```
#include <stdio.h>

main ()
{
   int n;
   float x;
   char mark;
   scanf ("%d %f %c", &n, &x, &mark);
   printf ("Of %d %s,\n%f got %c\'s\n",
      n, "students", x, mark);
}
```

Note:

- Escape sequences: \n newline, \' quote, \0, etc.

I/O Example

```
#include <stdio.h>

main ()
{
   int n;
   float x;
   char mark;
   scanf ("%d %f %c", &n, &x, &mark);
   printf ("Of %d %s,\n%f got %c\'s\n",
      n, "students", x, mark);
}
```

- input:

```
86 85.999 A
```

- output:

```
Of 86 students,
85.999001 got A's
```

Conditional Statements

Example:

```
if (age < 0)
{
   printf ("warning: negative age\n");
   age = -age;
}
```

Syntax:

```
if (condition) statement
if (condition) statement else statement
```

Rules:

- the condition is an int ! (no booleans)
- parentheses required around conditional expression
- use curly braces to make a compound statement

More Conditionals

Example:

```
if (x < 0)
  printf ("x is less than 0\n");
else if (x == 0)
  printf ("x is equal to 0\n");
else
  printf ("x is greater than 0\n");
```

What's wrong with this?

```
if (x < 0)
  if (y < z)
    printf ("y is less than z\n");
else
  printf ("x not less than 0\n");
```

While Loops

Example:

```
/* print "hi" forever */
while (1)
  printf ("hi");
```

Syntax:

```
while (condition)
 statement
```

Rules (again):

- the condition is an int ! (no booleans)
- parentheses required around conditional expression
- use curly braces to make a compound statement

For Loops

Example:

```
/* print "hi" three times */
int i; /* i continues to exist when loop ends */
for (i = 0; i < 3 ; i++)
  printf ("hi");
```

Syntax:

```
for (statement1; condition; statement2)
  statement3;
```

Equivalent to:

```
statement1;
while (condition) {
  statement3;
  statement2;
}
```

Loop Example

```
/* print squares up to 100 */
main ( )
{
   int j, up = 100;
   for (j = 0; j * j <= up; j++)
     printf ("%d \n", j * j);
}
```

Note:

- can't do: `for (int j = 0;` ...
- waste of one multiplication per iteration
- can you make it more efficient?

Example (cont'd)

```
/* print squares up to 100 */
void main ( )
{
   int j, up = 100, sq;
   for (j = 0; (sq = j * j) <= up; j++)
     printf ("%d \n", sq);
}
```

Note:

- recall equivalence to a while loop: condition is evaluated before the loop body

Functions

```
/* Increment; takes an integer argument and
 * returns the argument plus one.
 */
int incr (int i)
{
  int j;
  j = i + 1;
  return j;
}

main ()
{
  int k, m = 4;
  k = incr(m);
  printf ("k = %d, m = %d\n", k, m);
}
```

- output: `k = 5, m = 4`

More about Functions

- might have no return type, and no return statement:

```
void printhi ()
{
  printf ("hi\n");
}
```

- parameters are copied and can be modified

```
int incr (int i)
{
  i++;
  return i;
}
```

- default (unspecified) return type is int

Variables within Functions

But this does not work:

```
void no_incr (int i)
{
   i++;
}

void main () {
   int x = 5;
   no_incr(x);
   printf ("%d\n", x);
}
```

- beware that modifications are on internal copies of the parameters.
- note: **`void main()`** since main does not return a value

Arrays

```
int years[45];
float temperatures [11];
void main () {
  years[0] = 2000;
  temperatures[11] = -45.67;
}
```

Rules:

- indices start at zero
- maximum valid index is the size of the array minus 1
- but C lets you go beyond the declared boundaries
- `temperatures[11]` is illegal

Characters

Characters

```
char a, b, c1, c2;
a = '0';  b = '\037';  c1 = 'K';  c2 = c1 + 1;
```

- assigns values: 48, 31, 75, 76

Casting

```
printf ("%c %d\n",  c1, (int) c1);
```

- Outputs: **K 75**

Strings

Strings are '\0'-terminated arrays of char:

```
char s[3] = "hi"; /* invisible '\0' */
char t[3] = {'h', 'i', '\0'};
```

String operations

```
#include <string.h>
strlen ("there"); /* returns 5 */
strcpy (s, t);    /* copy t to s */
strcmp (s, t)     /* alphabetical comparison */
```