Control Flow and Functions

1 Arrays

- You can use arrays to store multiple values of the same type

```
int a[6] = {2,4,7,1,2,4}
```

2 True/False Comparison

- Traditionally, C did not have boolean types and just used ints
- Comparisons will evaluate to 1 if they hold and 0 if they don't
- C99 introduced bool, which is defined in `stdbool.h`

3 Statements and compound statements

- A statement in C is a single instruction terminated with a semicolon

```
printf("Hello World!\n");
```

- A compound statement is a set of statements surrounded by a pair of curly brackets {}

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

- You can always replace a statement with a compound statement
- C doesn't care about formatting - but we need it!

4 Some Style Conventions

Compound statement:

- Curly brackets on own lines
- Indent body with 2/4 characters or a tab

Variable names:

- Constants - all capitals: MAX, PI
- #defines - all capitals: DEBUG
- Normal variables
  - camel case: myAge
  - or snake cake: my_age

Comments

```
/* This is a comment
that can go on multiple lines */
// This is a single line comment
```

5 Iteration Statements

- C's iteration statements are used to create loops
- A loop is a statement whose job is to repeatedly execute other statements: the loop body
- In C, every loop has a controlling expression
- Each time the loop body is executes, the expression is evaluated
- If the expression is true (has a non zero value), the loop continues to execute

C provides three iteration statements

- The while statement is used for loops whose controlling expression is tested before the loop body is executed

  ```
  while (a> 100) {...}
  ```

- The do statement is used if the expression is tested after the loop body is executed

  ```
  do {...} while (a>100);
  ```

- The for statement is convenient for loops that increment or decrement a counting variable

  ```
  for (a = 199; a>100; a = a-1) {...}
  ```

6 The break and continue statements

- The break; statement causes the innermost enclosing loop (or switch) to be exited immediately

  ```
  for(n = 10; n<=10; n++){
      statement(s)
      break; // or continue;
  }
  ```

- continue; causes the next iteration for the loop to begin (it does not apply to switch)
- In the case of a while or do loop, the test part is executed immediately; in the case of a for loop, control first passes to the increment step
- In a do while, continue will evaluate before looping round

7 The if-else statement

```
if (expr1){
    statement1
}else{
    statement2
}
```

7.1 Cascaded if statements

Allows testing of a series of conditions

```
if(boolean_expression1) {
/* Executes when the boolean expression 1 is true */
} else if( boolean_expression2) {
/* Executes when the boolean expression 2 is true */
} else if( boolean_expression3) {
/* Executes when the boolean expression 3 is true */
} else {
/* executes when the none of the above condition is true */
}
```

8 Static program checking

Using the wall (all warnings) flag on the compiler will make static checks e.g. for

```
#include <stdio.h>
int main(){
    int x = 0;
    if(x=0){ printf("x is 0\n");}
    return 0;
}
```

This doesn't give the correct output as there is an assignment in the if statement, which equates to zero, which is equivalent to false, and so the if statement passes

9 The switch statement

This has the form

```
switch(expression){
    case const-expr: statements
    case const-expr: statements
    default: statements
}
```

Warning: if there is no break statement, execution falls through - all the cases will be executed

10 Incrementing etc

C has many methods of incrementing and decrementing

```
++
--
+=
-=
*=
/=
%=
```

x++: Evaluates to x, then adds one
++x: Adds one, then evaluates to x (so +1)

In a for loop it doesn't matter which one you use

11 Functions

11.1 Declaration

- Functions encapsulate code in a convenient way
- Analogous to methods in an O-O language
- Functions can be declared before they are defined, as a function declaration:

  ```
  return-type function-name (paramenters);
  ```

- E.g. to calculate base raise to the power n

  ```
  int power(int base, int n);
  ```

- Often we put these in a header file (.h)

11.2 Definition

Functions can be defined anywhere in a program wile, if the declaration precedes use of the function

```
int power (int base, int n) {
    int p;
    for ( p=1; n>0; n--)
        p=p*base
    return p;
}
```

11.3 Call by value

- Function parameters in C are passed using a call by value semantic

  ```
  result = power(x,y);
  ```

- Here when x and y are passed through to `power()`, the values of x & y are copied to the base and n variables in the function
- A function cannot affect the value of its arguments
- `swap (x,y)` example