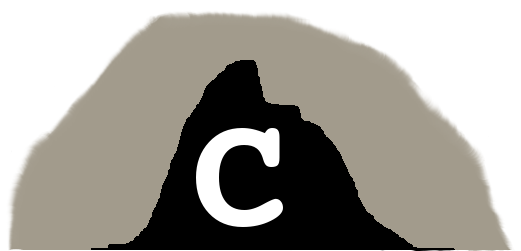
****

**Welcome to the C cave**, a place of basic low-level programming.

Last month’s tutorial concentrated on writing a first program, input and output from the keyboard (I/O) and simple mathematical operations. Before introducing this week’s tutorial, let us take a quick look at the solution to last month’s challenge problem.

**Challenge solution**

#include <stdio.h>

int main() {

int i = 100, j = 9, k;

i = i/10;

k = i – j;

printf("Well done this program compiles.\n");

printf("%d - %d = %d\n",i,j,k);

return 0;

}

Did you get them all right? Rather than trying to find all of the errors by eye, it is often a good idea to use the compiler to find them. If programs have been well designed, the compiler can be a helpful tool to spot typing mistakes.

Tutorial 2 – Variables, conditions and loops

Programming languages offer different ways of writing similar structures. The languages Python, Scratch and C, all have variables, logic conditions, loops and functions.

**Variables**

There are two categories of numeric variables in C, (i) whole numbers (e.g. int) and (ii) floating point numbers (e.g. float). When a variable is declared, a block of memory is allocated to variable. The amount of memory allocated to an int and float is the same, but the way the number is stored is different. An int is stored in memory as a sign bit and a 31bit value. For example,

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |

is 13 in decimal. The first bit on the right is 20, then 21, etc.. The sign bit is on the far left. Therefore, the largest number which can be stored in an int is 231-1, which is 2147483647. If one more is added to this number, the number becomes -2147483648. This happens because the sign bit is set. Here is a simple test program,

#include <stdio.h>

int main() {

int i = 2147483647;

i++;

printf("%d\n",i);

return 0;

}

int.c

An int cannot be used to store a fraction of a number. Fractional or floating-point numbers can be stored in a float. Unlike an int, a float has three pieces: a sign, an exponent and a mantissa. The mantissa contains a fraction from 0. to 2.0 and the exponent contains the power to which the number is multiplied by. For example,

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |

is 3.14159 is decimal. Each mantissa bit is 2-n, where n is the bit position. If a floating-point number becomes too big, it will become not-a-number or infinity. If the precision bits in the mantissa are all used the additional precision will be lost. This is called a floating-point error. In the example,

#include <stdio.h>

int main() {

float f = 3.14159;

f+=0.0000001;

printf("%f\n",f);

return 0;

}

float-error.c

the addition of 0.0000001 to the float f is lost.

Right, lets get back to some simple mathematics. The division of float and int variables behaves differently. An integer division will always be rounded down to the nearest int, whereas a float will be rounded to the precision of the mantissa.

#include <stdio.h>

int main() {

/\* Declare an int and a float and assign them the value three. \*/

int i = 3;

float f = 3.;

/\* Divide both numbers by 2. \*/

i /= 2;

f /= 2.;

/\* Print the results. \*/

printf(“int (3/2) = %d, float (3./2.) = %f\n”,i,f);

return 0;

}

division.c

The result of the program division.c is

int (3/2) = 1, float (3./2.) = 1.500000

Characters are stored in a similar way as an int. A char is a signed 8-bit number. It can be used to print the ASCII character or the position of the ASCII character within the table.

#include <stdio.h>

int main() {

char c = 'a';

printf("The character \'%c\' has the value %d\n",c,c);

return 0;

}

character.c

A string is represented in memory as a series of sequential characters. A series of sequential values of the same type is called an array.

**Arrays**

An array of ten integers can be declared by,

int array[10];

Each element an array can be accessed using the corresponding index. Similar to many other languages, these indices start from zero up to N-1, where N is the size of the array. When an array is declared a sequential block of memory is allocated. The values in each array index may or may not be zero. Therefore, it is important to initialise each array element with a value. Arrays may be declared and assigned values at the same time,

int array[10] = {1,2,3,4,5,6,7,8,9,10}

or each element of the array can be initialised individually.

#include <stdio.h>

int main() {

float fourVector[4]={1.0, 2.0, 0.5, 2.292};

printf("There are %ld elements in the fourVector array\n",sizeof(fourVector)/sizeof(float));

printf("fourVector[3]=%f\n",fourVector[3]);

return 0;

}

array.c

In the example array.c, the sizeof command returns the size of the array as a whole. Therefore, dividing this by the sizeof float returns the number of elements in the array.

A string of nine characters can be stored in a character array of ten characters,

char str[10];

The tenth element is used to store the string terminating character ‘\0’. An array of strings can be stored as an array of an array of characters. Here is a simple string example,

#include <stdio.h>

int main() {

char str[50]="A test string";

printf("\"%s\"\n",str);

printf("The third character = \'%c\'\n",str[2]);

return 0;

}

string.c

**Conditions**

Many different conditions can be expressed in the C language. These conditions can be used within if-else statements, switch, or within loops. A summary of logic conditions is provided in the table below,

|  |  |  |  |
| --- | --- | --- | --- |
| Condition | Meaning | Condition | Meaning |
| a == b | a equal b | **!a** | Not a |
| a != b | a not equal b | **a || b** | a or b |
| a > b | a greater than b | **a ^ b** | a exclusive or b |
| a <= b | A less than or equal b | **a && b** | a and b |

These conditions can be combined together to produce more complicated conditions. When several conditions are combined together it is important to use parentheses, to set the order in which the conditions are tested. For example,

(a || b) && c

is not the same as

a || (b && c)

Conditions can be used within if-else statements to choose a particular outcome,

#include <stdio.h>

int main() {

int apples = 0;

float cost, costPerApple = 0.20;

printf("How many apples would you like to buy?\n");

scanf("%d",&apples);

if(apples > 20) {

printf("Sorry we only have twenty apples left.\n");

}

else {

cost = costPerApple\*apples;

printf("That will be %.2f pounds please.\n",cost);

}

return 0;

}

if-else.c

**Loops**

Repetitive operations, such as accessing array elements or performing the same mathematic operation several times can be more efficiently implemented using loops. C offers three main loop types: for, while, and do-while. Starting with the for loop,

#include <stdio.h>

int main() {

int i;

/\* loop from i=0 and stop when i >= 10 \*/

for(i=0;i<10;i++) {

/\* print the value of i \*/

printf("i = %d\n",i);

}

return 0;

}

for.c

The program for.c prints the numbers from 0 to 9 on the screen. This program can easily modified to add these numbers together,

#include <stdio.h>

int main() {

int i, sum = 0;

/\* loop from i=0 and stop when i >= 10 \*/

for(i=0;i<10;i++) {

/\* Add the value of i to sum and assign the result to sum \*/

sum += i;

}

print(“sum = %d\n”,sum);

return 0;

}

sum.c

It is often useful, to combine loops with arrays. For example, the variable i can also be used to access each element of an array,

#include <stdio.h>

int main(){

int i, array[10];

for(i=0;i<10;i++) {

/\* Assign the value of i to the ith element of the array. \*/

array[i] = i;

}

for(i=0;i<10;i++) {

/\* Print the ith element of the array. \*/

printf("array[%d]=%d\n",i,array[i]);

}

return 0;

}

array-for.c

**Challenge questions**

Not all programming errors will cause the compiler to fail. In the problem below there is an error which prevents the program from finishing.

#include <stdio.h>

int main() {

int i,j,value=2;

for(i=0;i<8;j++) {

value \*= value;

}

printf("2^8 is %d\n",value);

return 0;

}

For the second problem, use a for loop to calculate the first ten triangular numbers. The program should use two variables, a counter counting from 1 to 10 and a variable to hold the sum of the counter values.

The solutions will be given in the next tutorial.

W. H. Bell and D. Shepley