An Interesting Operator

++i adds one to i, and returns the new value of i
--i subtracts one from i, and returns the new value of i
i++ adds one to i, and returns the old value of i
i-- subtracts one from i, and returns the old value of i
A program to illustrate the ++ operator (and initializing variables)

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
    int count=1;
    int new_count;
    char my_string[20];
    char my_char;
    char answer='y';

    printf("This program shows what can happen if you forget to initialise variables.\n");
    printf("The variables my_string and my_char are not initialised.\n");
    printf("The value of my_string is %s, while the value of my_char is %c.\n", my_string, my_char);
    printf("The integer new_count was not initialised either. Its current value is %d.\n", new_count);

    printf("Would you like to continue while I explain incrementing and decrementing in C?\n");

    scanf("%c", &answer);
```
if(answer == 'y')
{
    printf("We will increment the variable count using
count++.\n");
    printf("count is currently %d\n", count);
    count++;
    printf("count is now %d\n", count);
    printf("It seems fairly simple, but we can also use ++count,
which makes a difference.\n");

    printf("count is currently %d\n", count);
    new_count=count++; 
    printf("we will assign new_count = count++. The value of new_count is
now %d\n", new_count);
    printf("but count is now %d\n", count);
    new_count=++count;
    printf("Now we will assign new_count = ++count. The value of
new_count is %d this time\n", new_count);
    printf("++count increments the count variable before assigning it to
new_count\n");
    printf("whereas count++ assigns the variable and then
increments it.\n");
}

```c
else
{
    printf("Perhaps another time...\n");
}

return(0);
}

More Unary operators

Unary operators are operators that only take one argument.

- `sizeof(i)` the number of bytes of storage allocated to i
- `+123` positive 123
- `-123` negative 123
- `~i` one's complement (bitwise complement)
- `!i` logical negation (i.e., 1 if i is zero, 0 otherwise)
- `*i` returns the value stored at the address pointed to by i
- `&i` returns the address in memory of i
- `i[j]` array indexing
- `i (j)` calling the function i with argument j
- `i.j` returns member j of structure i
- `i->j` returns member j of structure pointed to by i
The "for" loop

- The basic looping construct in C is the "for" loop:
  
  ```c
  for (initial_expression; loop_condition; loop_expression)
  statement;
  ```

- this is precisely equivalent to the following code fragment:
  ```c
  initial_expression;
  while (loop_condition)
  {
    statement;
    loop_expression;
  }
  ```

- "loop expression " is the control variable e.g. answer = 'y'

An example:

```c
for (i = 0; i< 100; i++)
    printf("%i\n", i);
```

which simply prints the first 100 integers. If you want to include more that one statement in the loop, use curly brackets to delimit the body of the loop, e.g.,

```c
for (i = 0; i < 100; i++) {
    j = i * i;
    printf("i = %i; j = %i\n", i, j);
}
```
The do-while loop

Similar to the “for” loop

```
initial_expression;
    do{statement;} while (loop_condition)
```

For example:

```
i = 0;
do
  {
    printf("The value of i is now %d\n", i);
    i = i + 1;
  }
while (i < 5);
```

Break and continue

These two statements are used in loop control.

• “break” exits the innermost current loop.
• “continue” starts the next iteration of the loop.
Break and continue example (program continues over two pages)

#include <stdio.h>
int main()
{
    int xx;
    for(xx = 5 ; xx < 15 ; xx = xx + 1)
    {
        if (xx == 8)
            break;
        printf("In the break loop, xx is now %d\n", xx);
    }

    for(xx = 5 ; xx < 15 ; xx = xx + 1)
    {
        if (xx == 8)
            continue;
        printf("In the continue loop, xx is now %d\n", xx);
    }

    return 0;
}
Structures

Structures are C variables that store aggregate data of different types, unlike arrays that store aggregate data of the same type.

You define a structure as follows:

The structure "collection" is defined to store a catalogue, where each item in the catalogue has a catalogue number and a description.

```c
#include <stdio.h>
int main()
/*The structure collection is defined so that it will store a catalogue number, name and description for each item in the collection.*/
    struct collection
    {
        int cat_number;
        char name[20]
        char description[20];
    }
```

/* If you want to use a "collection" structure in your program you will need to declare a variable as a structure of type collection and give it a name*/

/* I want to create an itemised list of the dresses in my wardrobe, so I will declare a structure of type collection nad name it my_dresses*/

```c
struct collection my_dresses;
```

/*I then want to put some data into my structure.*/

```c
    printf("Enter the catalogue number for this item:\n");
    scanf(%d, &my_dresses.cat_number);
    printf("Enter the name of this item:\n");
    scanf(%d, &my_dresses.name);
    printf("Enter the catalogue number for this item:\n");
    scanf(%d, &my_dresses.description);
```
/* Remember that the & in front of the variable refers (points) to the address at which the input data is to be stored. This is fine for one item, but how can I store many items? Simply define an array of structures, like so:*/

    struct collection my_dresses[20];

/* An array of 20 collection structures named my_dresses is created.*/

/*To put access a particular structure in the array you proceed as for any other array:*/

/*If I want to print out the name for the 4th dress stored I specify this as */

    printf("Dress %d is %s\n", my_dresses[3].cat_number,
            my_dresses[3].name); /*Don't forget that in C, the first item in an array is 0, not 1.*/

A structure can store variables of different types, and this is its primary purpose. A structure is like a record in a database. For example, if you store names, addresses and ages of customers in a database, then the record for one customer would be a structure.