1 Arithmetic Operators

There are five arithmetic operators in C, which are shown in the table below:

Symbol	Operation
+	Addition
-	Subraction
*	Multiplication
/	Division
%	Modulus

These are all binary operators; they are supposed to be placed between two numbers. The only exception is the “−”, which can also be used as a unary negation operator to denote negative numbers.

All these operators can be applied to all data types we have seen, with the exception of the modulus operator “%”, which can only be used for integers.

As far as priority goes, you should know that multiplication and division are done before addition and subtraction. For anything else, you should use parentheses. As implied here, it is possible to use parentheses to group operations together in C.

2 Making Decisions

An important part of programming is making decisions within the program. Otherwise, all statements would be executed one after the other in a way that can be said to be “too straightforward”. One of the main methods of making a decision in a C program is to use the `if` statement. The general form of an if statement is as follows:

```
if (<condition>)
    <if-statement>;
else
    <else-statement>;
```

If the `<condition>` is true, the `<if-statement>` is executed. Otherwise, the `<else-statement>` is executed. It is possible to leave out the `else` part completely, in which case no statement will be executed if the `<condition>` is false.

The `<condition>` can be any valid C expression. In C, the value of zero means false, and any non-zero value means true.

Note that either or both of `<if-statement>` and `<else-statement>` can be blocks of code rather than single statements. In this case the format becomes:

```
if (<condition>) {
    <if-block>;
} else {
    <else-block>;
}
```

I personally always prefer the latter form with the braces included even if the block of code consists of a single statement. Firstly this helps the readability of the program, and it also makes it easier to add code into the block should you later need to.

Here is an example for the use of the `if` statement:

```
#include <stdio.h>

int main(void)
{
  int a=3;
  int b=2;

  if (a > b) {
    printf("%d is greater than %d\n", a, b);
  } else {
    printf("%d is less than %d\n", a, b);
  }
}
```

You should be able to understand what this program does on its own. The only new thing here is the use of the “greater than” operator in the condition, which we will get to now.

3 Relational Operators

The table below shows the relational operators in C and their meanings:

Symbol	Meaning
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

These should all be obvious, but here is one warning: Do not confuse the assignment operator “=” and the operator used for testing for equality “==”. That very often leads to very nasty results in a program.

4 Logical Operators

C also has logical operators that can operate on logic values. As mentioned before, in C zero means false and any non-zero value means true. In addition, when a logical or relational operator is used on two values, the result is always zero or one, regardless of the value of the operands. Here are the logical operators you can use in a C program:

Symbol	Meaning
!	Not
&&	And
||	Or

5 An Example Program

```
#include <stdio.h>

int main(void) {
  int a=2;
  int b=3;

  if (a > b) {
    printf("%d > %d\n", a, b);
  } else if (a < b) {
    printf("%d < %d\n", a, b);
  } else {
    printf("%d = %d\n", a, b);
  }

  return 0;
}
```

You should be able to figure out clearly what will happen when you run this program. Here, we have a construct that has not been mentioned explicitly, but was rather implied: The if-else ladder. Examine it carefully; make sure you do understand how it works. Note that in any case only one of the blocks of code will ever execute in such a ladder.

6 Getting Input from the Console

The example code above is very bland, since what will happen is very obvious right from the beginning. Now we will learn a way to receive user input through the keyboard. This is the `scanf()` function. I think showing the example first is the way to go:

```
#include <stdio.h>

int main(void) {
  int a;
  int b;

  printf("Enter a number:");
  scanf("%d", &a);

  printf("Enter another number:");
  scanf("%d", &b);

  if (a > b) {
    printf("%d > %d\n", a, b);
  } else if (a < b) {
    printf("%d < %d\n", a, b);
  } else {
    printf("%d = %d\n", a, b);
```

```
  }

  return 0;
}
```

As you can see in the example, `scanf()` is quite simple to use. The first argument is the format string, which should contain conversion specifiers. Any following arguments are variables which will receive the results of the conversion. The most typical use is to use a single conversion specifier and a single variable (since users usually can not be trusted to provide well-formatted input). The conversion specifiers are `%d` for integers, `%ld` for long integers, `%f` for floating-point numbers, and `%lf` for doubles. For now, regard the `&` in front of the variables as part of the `scanf()` command, we will elaborate on its meaning in coming lectures.