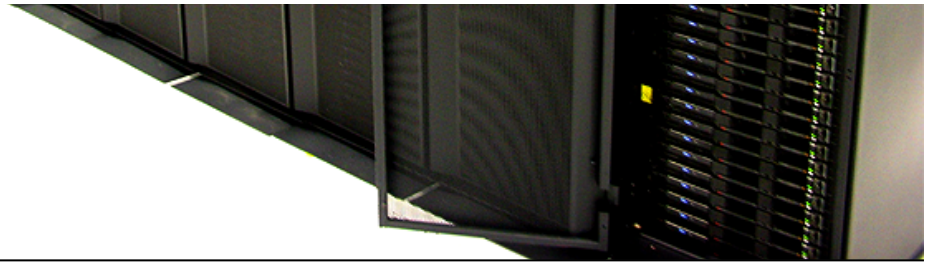
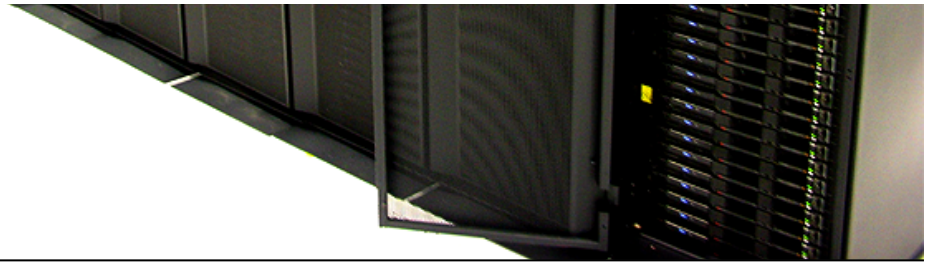
ICHEC
Irish Centre for High-End Computing



Nanoscale Simulators of Ireland (NSI) Graduate Lectures

Introduction to Programming

ICHEC
Irish Centre for High-End Computing



Irish Centre for High-End Computing

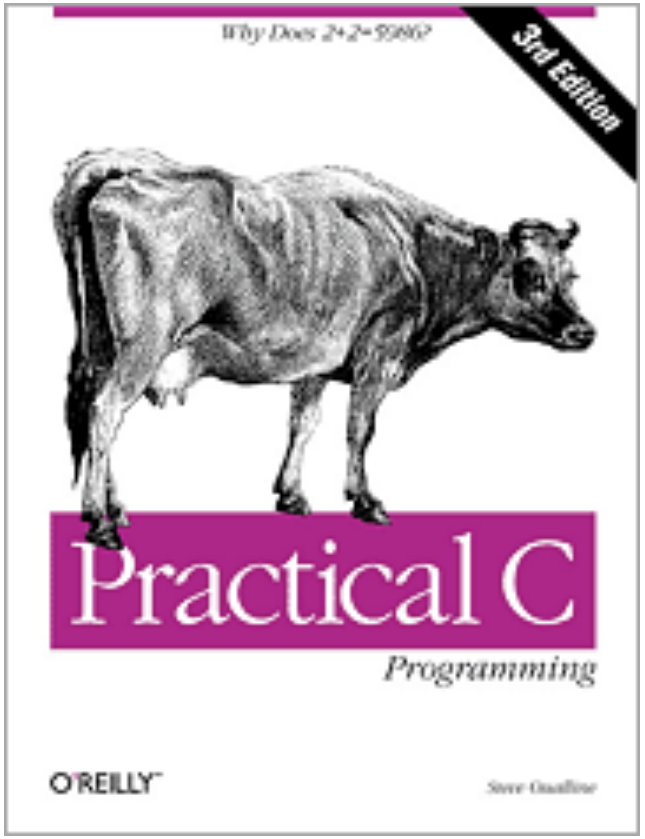
- Michael Browne
- ICHEC Support Scientist
- http://www.ichec.ie

Recommended Text

Virtually any introduction to a language book.

Practical C Programming, ISBN: 1-56592-306-5

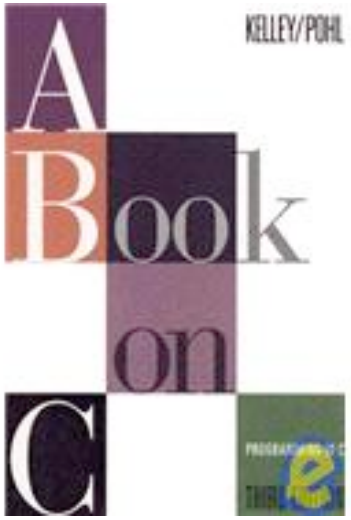
- For dedicated High Performance Computing (HPC) material see: http://www.ichec.ie/training_support/

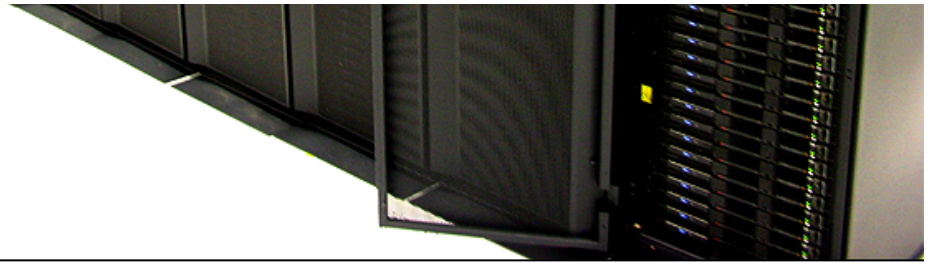


Recommended Text

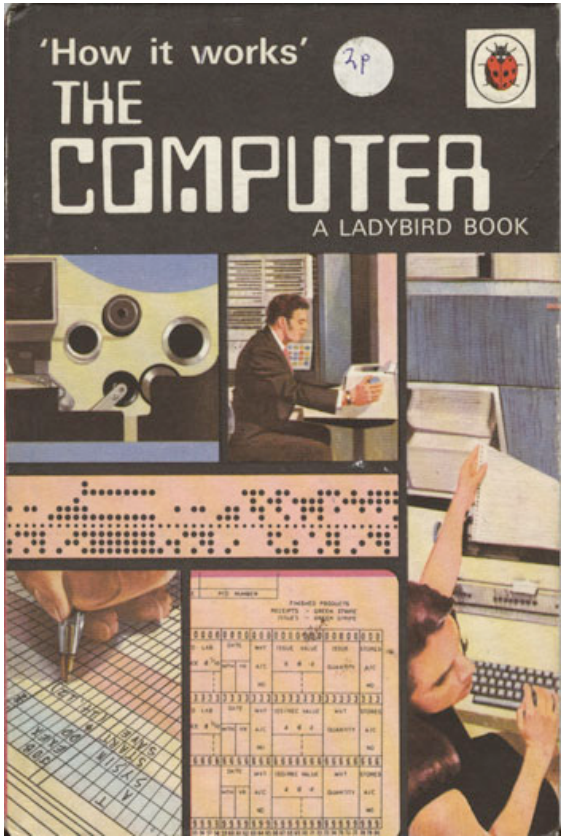
"A Book on C" by:

Al Kelley and Ira Pohl



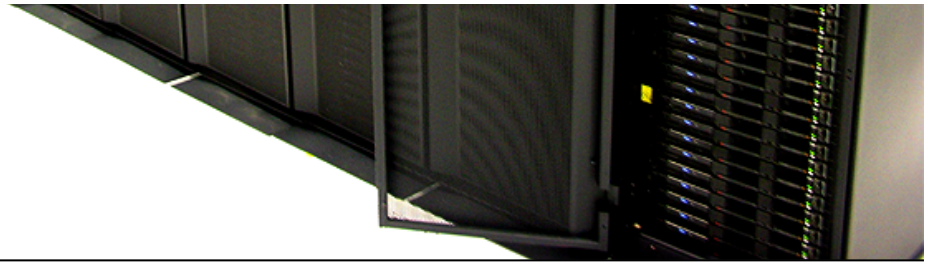


What is a Program?



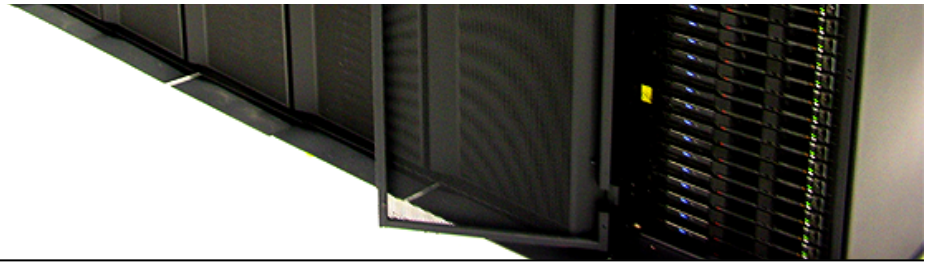
Published in 1971 & 79

"Programming in a high level language is something most people can do provided they are given time to learn the rules that must be followed."



What is a Program?

- A series of instructions to be carried out
- Written in human readable form
- Yet executed by computer
- The human readable form is written in one of many 'languages'
- How that is turned into a machine readable form depends on the language



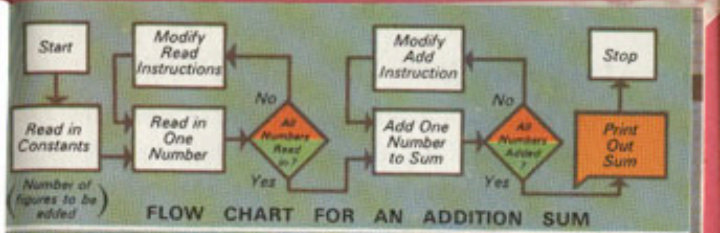
Programming

A set of instructions given to a computer is known as a *program*. The first step in preparing such a program is to draw a *flow chart*, two examples of which are shown opposite. This is built up from a number of connected boxes, the label attached to each box showing the job or calculation which is to be done at each step. One very important type of box is the *decision box*, in which a question is asked. The only answers allowed to the computer are 'yes' or 'no' and although this may seem too simple to be of use in a complicated problem, remember that a computer can ask nearly half-a-million questions a second!

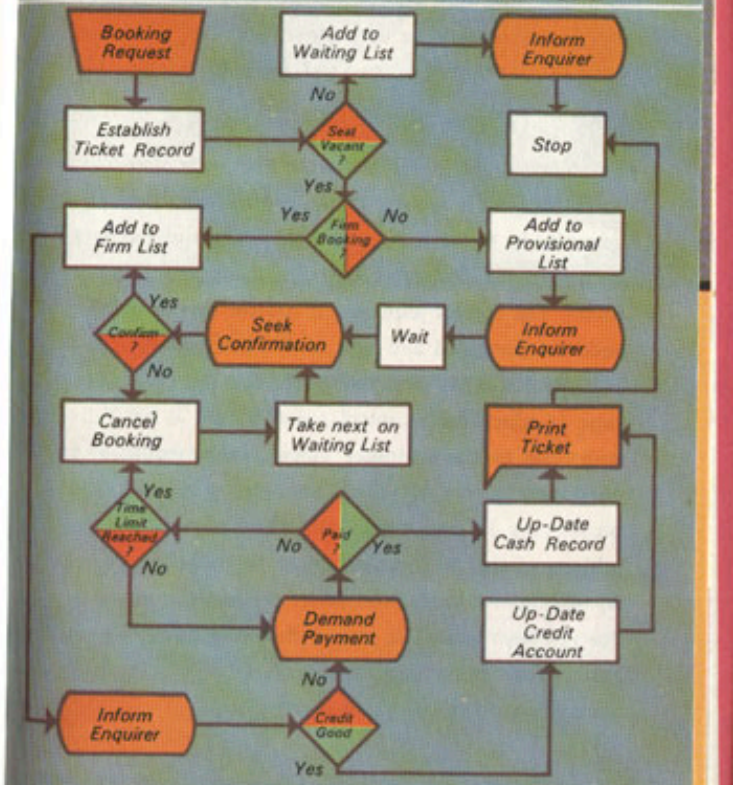
Transferring the job of each box into number form (*machine code*) is very laborious, and the programmer is helped in this task by an intermediate language (*high level language*) which is then translated by a master program (*the compiler*) into machine code. There are many high level languages in use, the most common being FORTRAN (Formula Translation), ALGOL (Algorithmic Language) and COBOL (Common Business Orientated Language).

Programming in machine code is a job for a highly trained person, whereas programming in a high level language is something most people can do provided they are given time to learn the rules that must be followed.

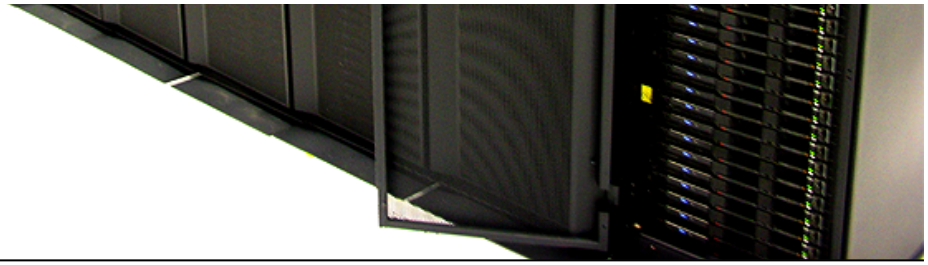
38



FLOW CHART FOR AN ADDITION SUM

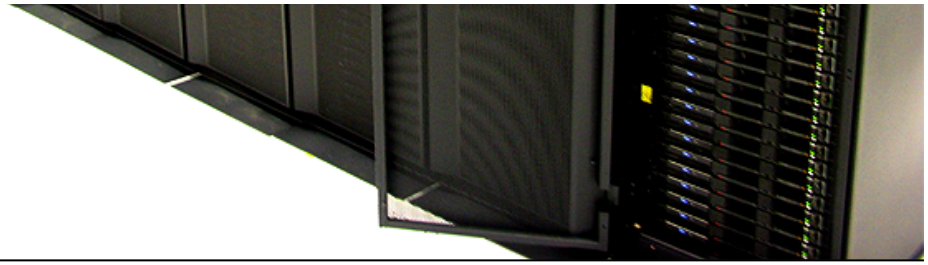


FLOW CHART FOR A FLIGHT BOOKING



Sample Program – in c

```
#include <stdio.h>

int main() {
   printf ("Hello World!\n");
   return 0;
}
```

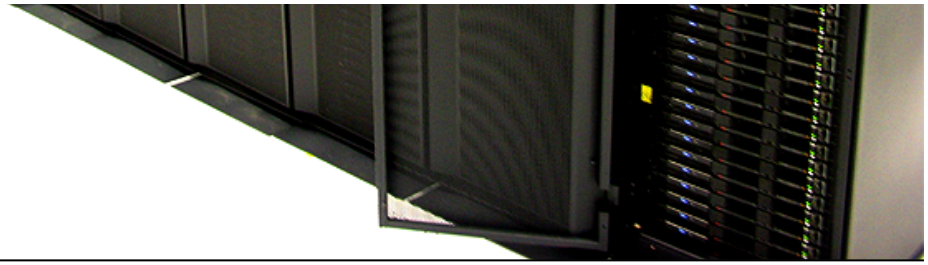


Data Types

- In a language data be it a number, a sentence, table of numbers etc.
- Some languages place more emphasis on the type of data than others
- The terms scalar, vector, array are commonly used.

e.g from C:

```
int mynumber=5;        int values[4]={1, 3, 5};
```

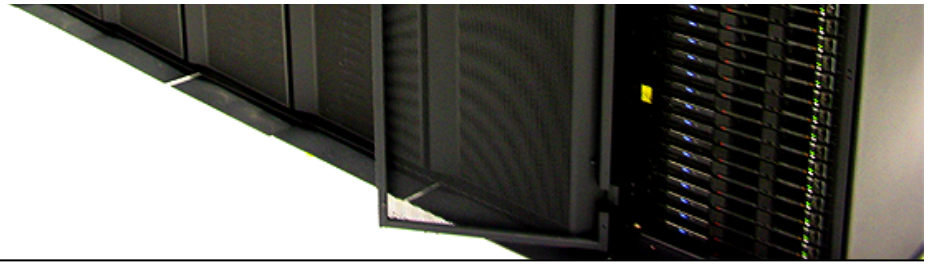


Arithmetic operations

Arithmetic operators are straightforward in C:

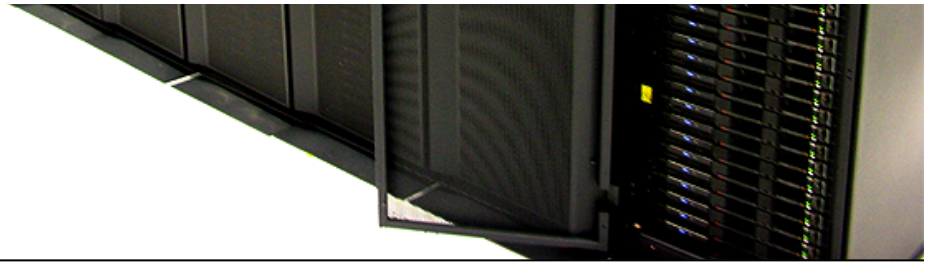
+ - * / % (modulus)

However in object orientated languages e.g. C ++ these operators can be “overload” to work with non arithmetic data.



Loops

- Unlike us computers are REALLY good at do the same thing over and over again!
- Many programs especially scientific programs are built on loops.
- There are different types of loops, the simplest is probably the “for loop”

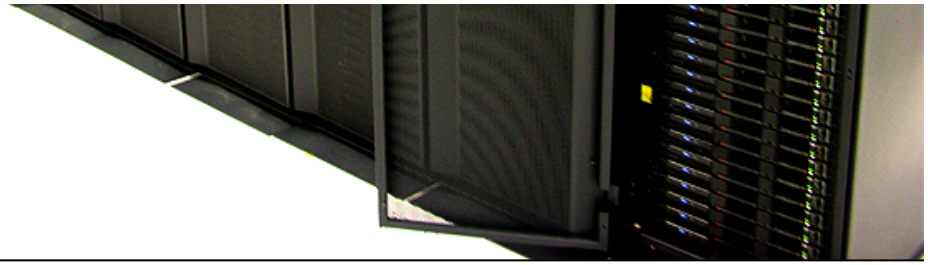


The for loop or for statement

```
for (i=0, i<n; i++) {
   printf("i is %d\n",i);
}
```

Prints a list integers from 0 to n-1

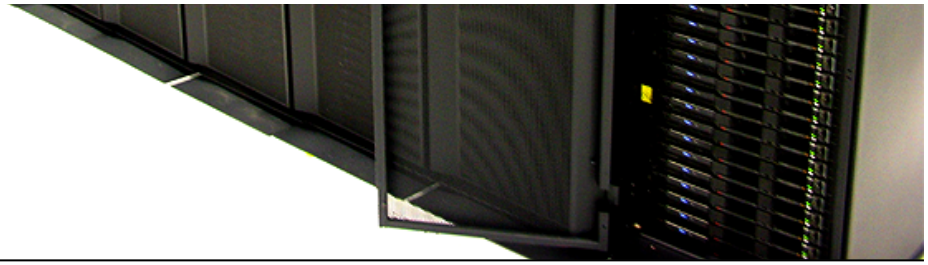
Note: i is set to 0 as the first operation of the loop & after each completed operation i is incremented.



While loop

```
while ((c=getchar()) != EOF) {
   printf("i is %c\n", c);
}
```

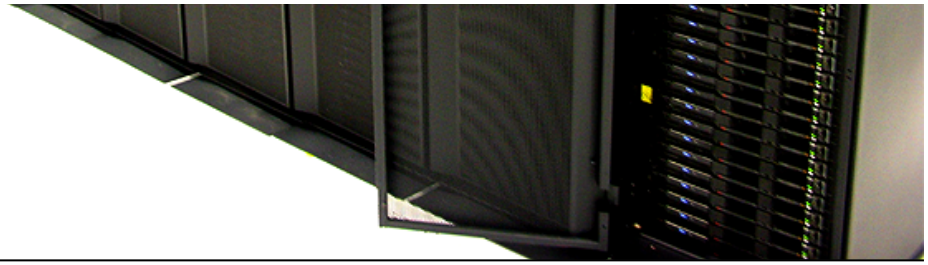
The while loop is useful for dealing with situations where we don't always know how many iterations will be required.



Do loop

```
do {
   some statements
} while (expression);
```

- Do loop is a variant of the while loop where the controlling expression is evaluated at the end on an iteration. The statements inside the loop are executed at least once.

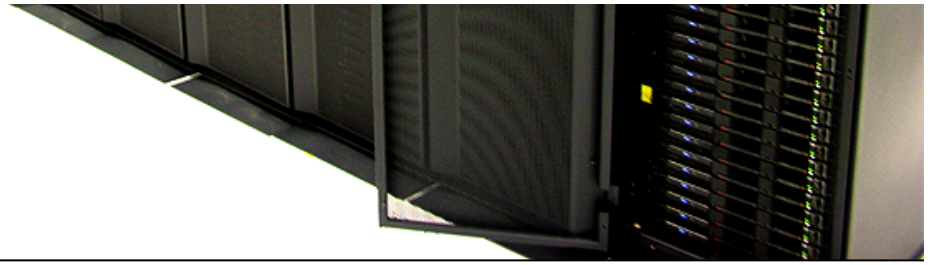


Conditional Statements

- The if statement:

```
if (expression)
        statement1;
else
        statement2;
```

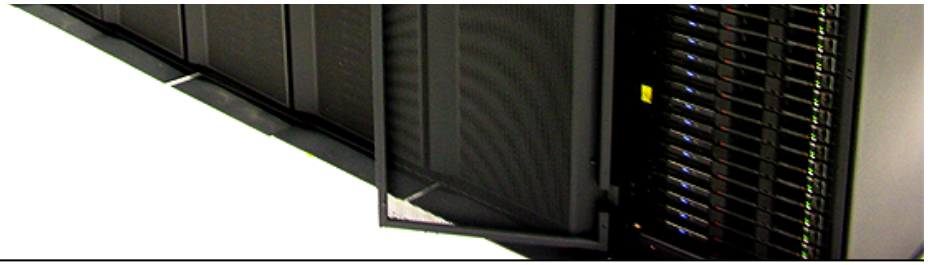
```
if (time == 0) {
        x = 0;
        y = 0;
} else {
        x = 1;
        y = 1;
}
```



Switch Statement

Akin to a structured if else statement:

```
switch (c) {
    case 'a':
        x++;
        break;
    case 'b':
    case 'B':
        y++;
        break;
    default;
        z++;
}
```

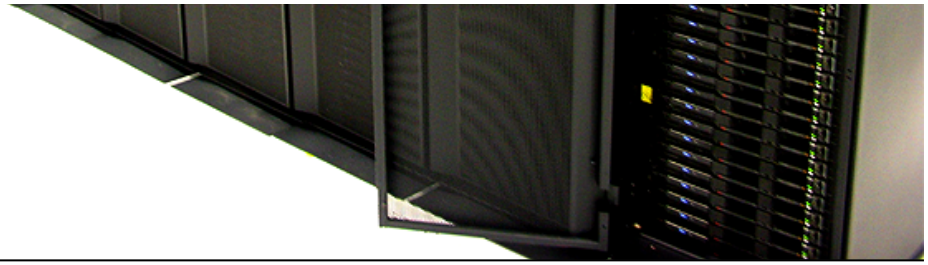


Standard Libraries

- The operating system provides a variety of functionality to the programmer. These functions are termed the "standard libraries".
- One includes the relevant header files for parts of the standard library as follows:

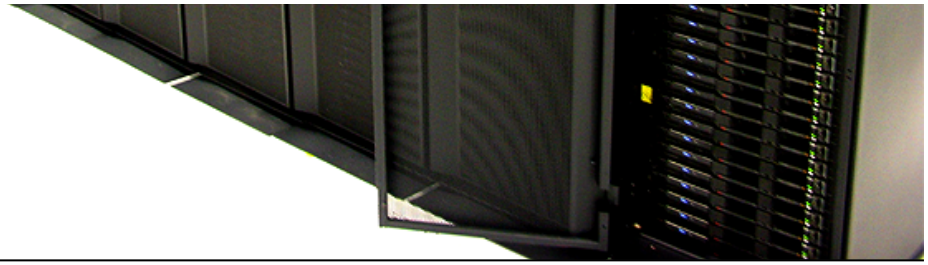
e.g: #include <stdio.h>

#include <stdlib.h>

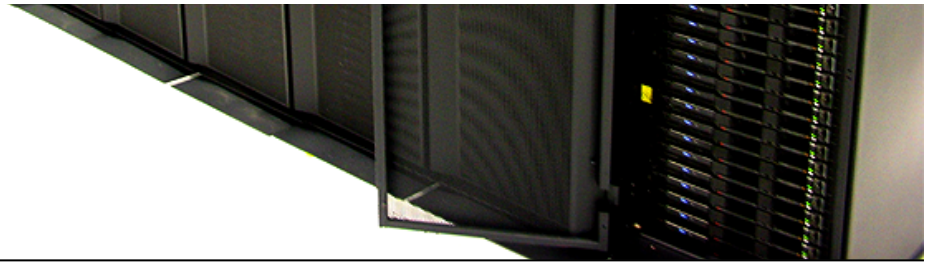


File IO

- Reading and writing to a file is handled much like reading and writing to the console.
- The key difference is the need to open and close files and attach these to file handles.
- There is scope for errors beyond the programmers control when dealing with files so error checking really matters.



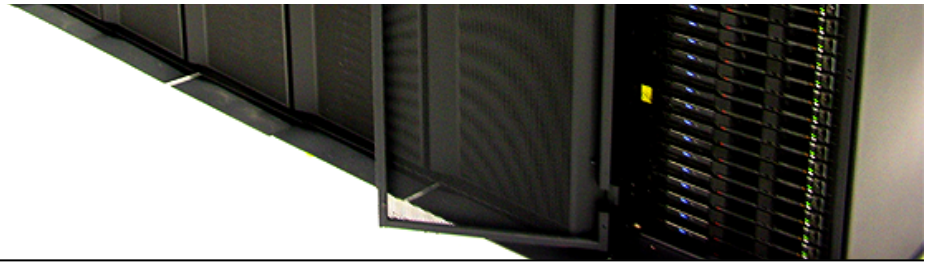
```
FILE *ifp;
ifp = fopen("myfile.txt","r");

while ((c = getc(ifp)) != EOF)
    printf("%c",c);

fclose(ifp);
```



The most common file io calls are:

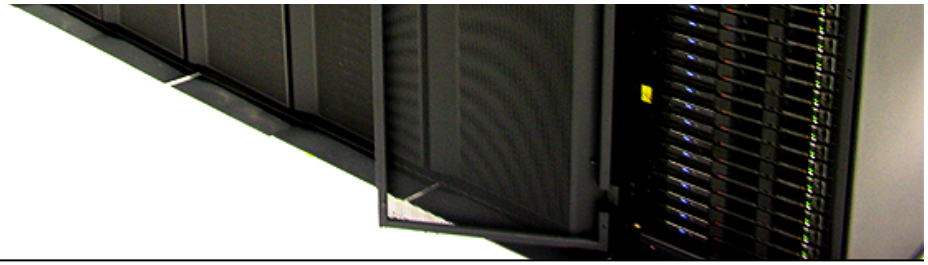
```
int fscanf(FILE *fp, const char *format, …);

int fprintf(FILE *fp, const char *format, …);
```

Differ from regular scanf() and printf() by the addition of the *fp parameter.



Compilation

- Compilation is the process of turning source code into executable code.
- It is carried out by a special program called a compiler.
- e.g. $cc myprog.c
- Produces an executable called a.out
- $cc –o myprog myprog.c
- Produces an executable: myprog

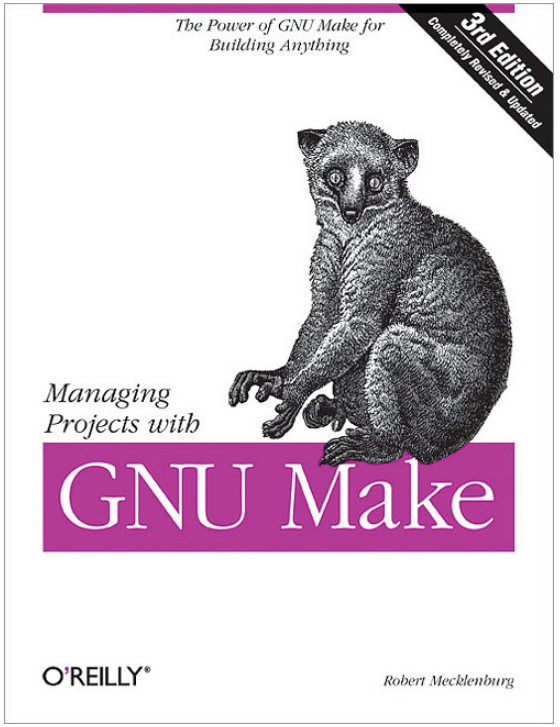


Introduction to Makefiles

- Keeping an entire program's code in one file, is inefficient so multiple files are normally required.
- Make files are a common way to manage the build process.
- Makefiles have two core functions expressing dependencies and setting build settings.

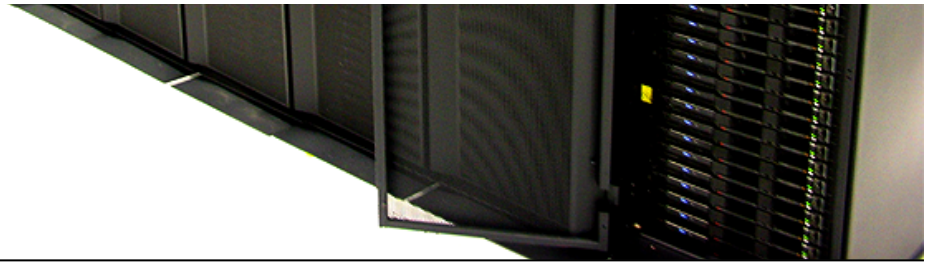
Recommend Text

- Useful for software engineering in general not just C coding!
- Managing Projects with GNU Make, Third Edition, By Robert Mecklenburg
- ISBN 10: 0-596-00610-1



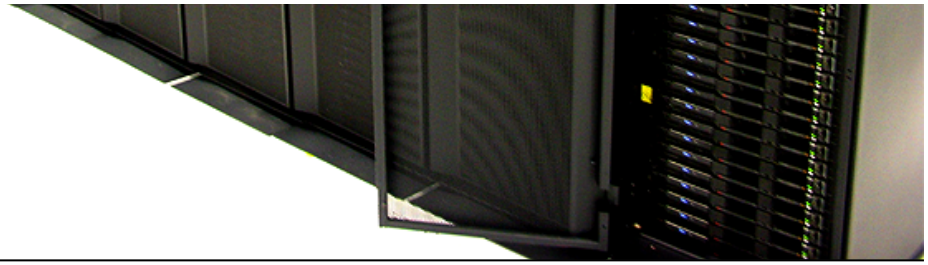
Also see

- Building and porting, software engineering for scientists, by Alastair McKinstry
- http://www.ichec.ie/training_support/tutorials/build.pdf



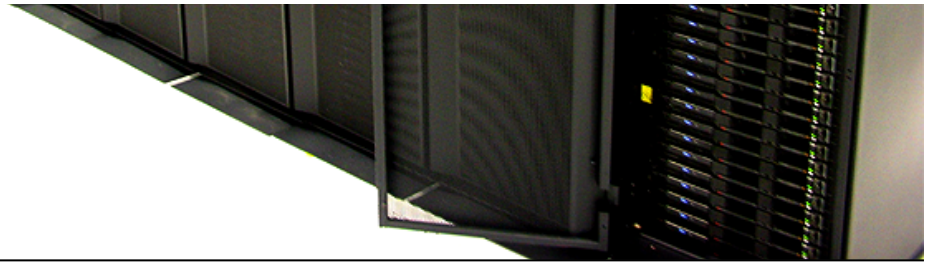
Dependencies example 1:

```
calc: main.o calc.o
   cc –o calc main.o calc.o
main.o: main.c calc.h
   cc –c main.c
calc.o: calc.c calc.h
   cc –c calc.c
```



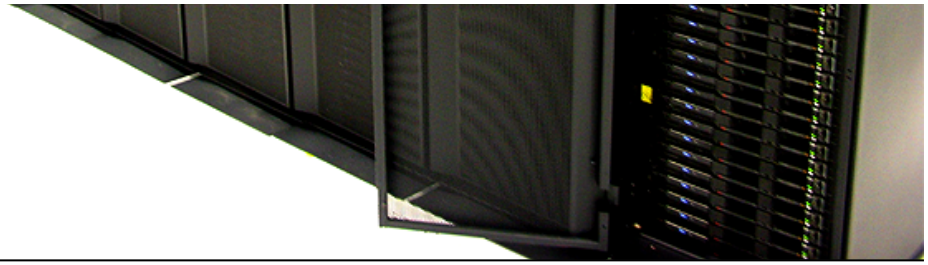
Dependencies example 2, make looks after the dependency of xyz.o on zyx.c:

```
calc: main.o calc.o
   cc –o calc main.o calc.o
main.o: calc.h
   cc –c main.c
calc.o: calc.h
   cc –c calc.c
```



Dependencies example 3 using $*.c built in variable $* is equivalent to main when main.o is being compiled and calc when calc.o is being compiled:

```
calc: main.o calc.o
    cc –o calc main.o calc.o
main.o calc.o: calc.h
    cc –c $*.c
```



Makefiles continued

- Makefiles often begin with the setting of a series of variables e.g:

```
BASE      = /users/home/jblog/mycode
CC        = gcc
CFLAGS    = -O3 -Wall
```