comp20005 Engineering Computation

Additional Notes

More C

Lecture slides prepared by Alistair Moffat

Chapter 10 – Dynamic Structures

The `malloc` function allows variable-sized segments of memory to be allocated. When no longer required, the memory can be returned back to the system using `free`.

So pointers can get their underlying arrays allocated at runtime.

Even better, if a structure type contains a pointer of its own type, can create recursive data structures.

They can be both one-dimensional (lists and stacks) and two-dimensional (trees).

To create and use a dynamic array:

```
stuff_t *A;
int n;
    ....
if ((A = malloc(n * sizeof(*A))) == NULL) {
    error_message("No memory available\n");
    exit(EXIT_FAILURE);
}
/* can now use A[i] and etc, up to A[n-1] */
    ....
free(A);
A = NULL;
```

To create a dynamic/linked structure:

```
typedef struct node node_t;
struct node {
    stuff_t stuff;
    node_t *next;
};

node_t *head=NULL, *new;
    ....
if ((new = malloc(sizeof(node_t)) == NULL) {
    ....
}
new->stuff = ....;
new->next = head;
head = new;
```

Then, to process the collection of stuff:

```
p = head;
while (p) {
    do_something_with(p->stuff);
    p = p->next;
}
```

Note that this example "stacks" the entries into a list – the one pointed to by `head` is the most recent insertion. A linked list can also be a "queue" if new items are inserted at end.

When done, and space is needed in different form for next phase of computation, free up the linked list node by node:

```
while ((p=head) != NULL)) {
    head = p->next;
    free(p);
}
```

Even more exciting:

```
typedef struct node node_t;
struct node {
    stuff_t stuff;
    node_t *left;
    node_t *rght;
};
```

This now allows the construction of a dynamic two-dimensional data structure, called a tree.

And recursive functions become the most natural thing in the world...

Chapter 10 – Function Pointers

Calling a function and supplying arguments into the call is an operation that makes use of an implicit function pointer.

Can have function pointer variables, arrays of function pointers, and also function arguments, as a higher-order of abstraction.

For example, a general purpose sorting routine needs a comparison function argument:

```
void
qsort(void *base, size_t nmemb, size_t size,
      int(*compar)(const void *, const void *));
```

Chapter 11 – File Operations

The type `FILE*` is defined in `stdio.h`.

The functions `fread` and `fwrite` allow input and output of structured data such as whole arrays and structs. Function `fopen` is used to connect a file directly to a location in the file system, for reading, writing, appending, etc.

The on-disk version of these binary files is not text, and cannot be read by humans or text editors.

But binary file operations are more efficient and manageable than via `scanf` and `printf`.

Can also have arrays of files, and file parameters.

```
FILE *fin;
char filename1[..];
int n;
stuff_t *A;
    ....
if ((fin = fopen(filename1,"r")) == NULL) {
    error_message("Unable to read from %s\n",
        filename1);
    exit(EXIT_FAILURE);
}
    ....
if ((fread(A, sizeof(*A), n, fin)) != n) {
    error_message("Failed to read %d values\n", n);
    exit(EXIT_FAILURE);
}
```

```
FILE *fout;
char filename2[..];
int n;
stuff_t *A;
    ....
if ((fout = fopen(filename2,"w")) == NULL) {
    error_message("Unable to write to %s\n",
        filename2);
    exit(EXIT_FAILURE);
}
if ((fwrite(A, sizeof(*A), n, fout)) != n) {
    error_message("Failed to write %d values\n", n);
    exit(EXIT_FAILURE);
}
fclose(fout);
```

Chapter 12 – Algorithms

This is where the fun really starts – clever techniques for processing problems.

Elegant sorting techniques based on recursion that take time that grows as $n \log n$ rather than the n^2 growth of the sorting methods in Chapter 7. This makes a BIG difference to performance.

Dynamic structures that allow combinations of item insertion, item search, and item deletion, in time that is logarithmic (or even less) in the size of the set.

Methods that rely on random numbers to assure good average performance.

Chapter 13 – Everything Else

Binary number systems, and floating-point representations (already included in comp20005).

Bit operations, rather than whole-of-word operations.

And there is more to the C preprocessor than constant `#define`'s.

Summary

C is a powerful programming language that is used in a wide variety of systems programs and embedded applications, including in engineering disciplines.

The more you practice, the better you'll get.

The better you get, the more you'll want to practice.

And remember, Programming is Fun!