CS23710 C Programming (and UNIX) Batch Six

David Price
Computer Science

Reminder - Structures

Like records in other languages…

```
struct mystruct {  int c;
                   float y;
                } z ;
```

Type is **struct mystruct** and z is a variable of this type.

Element (member) access ……

```
z.c = 7;
```

Typedef

New names for `types'
Common Example:-

```
typedef struct mystruct
      mynew;

mynew  r;
```

rather than

```
struct mystruct r;
```

Typedef - Continued

Often combine typedef and the struct definition

```
typedef struct s_tag {
                   int c;
                   float x;
                   struct s_tag * s_ptr;
                 }  newstruct ;
```

and the just use

```
newstruct
```

Structures and Arguments

K&R and ANSI both permit pointers to structures as arguments and return values for functions.

ANSI also permits structures themselves as argumens and return values.

Dynamic Data Structures structs are often used to build linked lists, trees, networks etc...

Potential Problems with Pointers

```
int * myfun()
{   int i;   /* A local variable created
                automatically as the function
                starts */

     return  &i;
}
/* function returns a pointer to an integer
   which has now been destroyed !!
*/
```

Unions

A single area of memory that can be used in one of a choice of ways.

```
union myun{
            int x;
            float y;
            char z;
}
union myun p,q; /* Two Variables */
```

Note: at any time, p&q can each only contain an **int** or a **float** or a **char**.

Bit Fields

A means to use one location and split it's BITs into separate accessible items.

```
struct mys{
            unsigned p1:3,p2:4,p3:1;
            int x;
           } z;
/* Z is a structure of type struct mys. It has
  four members, p1,p2,p3 and x. p1, p2, p3 are
  bit fields and x is an integer.
  CANNOT ask for address of bit fields.
  i.e.   &Z.p2     is ILLEGAL  */
```

FILES

stdio.h conatins a definition of a structure **FILE**.

typedef struct {

…….

} FILE ;

Files are often accessed via FILE *

functions …. fopen, fprintf, fscanf, fclose,
feof, fgets, fputs, fseek, fread, fwrite

Some `standard' FILE *
(streams) stdin, stdout, stderr

Program Parameters

main(int argc, char * argv[])
{

argc ----- no. of arguments
argv[0] ----- pointer to the first argument (string)

Pointers to Functions

ANSI **float (*p) (int I, int j) ;**
K&R **float (*p)();**

p is a pointer to a function that return a float.
and with the ANSI syntax, it has two paramters
of type integer.

(*p)(x,y)

is a call of the function
ANSI says you can also write p(x,y)
See Ammeraal p.129

Functions with variable parameter lists

If often proves useful to define a function
that may take different numbers of parameters
of different types on each call.

Example:

printf()

ANSI

```
#include <stdarg.h>
int myfun(int p, float x, …)
{ va_list my_arg_ptr;
   int i;
   va_start(my_arg_ptr, x);
   i = va_arg(my_arg_ptr, int);
   va_end;
   return i;
}
```

Other variants of ….

```
int myfun(p, x, va_alist)
int p; float x;
va_dcl
{  va_list  my_arg_ptr;
    int i;
    va_start(my_arg_ptr);
    I  = va_arg(my_arg_ptr, int);
    va_end ;
    return ;
}
```