CS23710 C Programming (and UNIX) Batch Five

David Price
Computer Science

Maths Functions

Lots of Maths Functions….

Check the MAN Pages

Read Ammeraal Page 82

Arrays, Pointers and Structures

(Ammeraal Chapter 6)

type name [size];

"The name of an array is a pointer to its first element"

Therefore, if ….

int s[10];

then

s equivalent to **&(s[0])**
&s[0]

(Check precedence rules)

More ……

&s[i]

is the address of the ith element (starting from zero)

s+i is equivalent to **&s[i]**

i.e. 1/. can do arithmetic with addresses
2/. arithmetic works in terms of the size of the elements

if we have …. **int * x;** as a definition

then ***x** means what **x** points at

therefore ***s** is the contents of **s[0]**

& s[6] - 3 is equivalent to **& s[3]**

Arrays as arguments to functions

```
int a[50];
…..
myfun(a);

void myfun(int *p)
 {      can now use
            p[0]
       or
            *(p+2)
 }
```

Or ……

```
void myfun(int p[])
{ ……   }
```

or K&R

```
int myfun(ptr)
int * ptr;
{  ……..   }
```

or K&R

```
int myfun(ptr)
int ptr[];
{ …… }
```

More about Pointers….

```
int * p;
```

P can be used to point to an integer. This allocates space for the pointer, but NOT for what it might point at.

```
main()
{ char *p;
      *p = 'A';
}
```

WRONG .. P points to nowhere...

```
main()
{ char *p, ch;
      p = &ch;
      *p = 'A';
}
```

O.K…..

Pointer Types

```
int i;
char * p_char;
p_char = &I ;              /* WRONG */
p_char = (char *) &i;    /*     OK     */
```

New generic pointer type...

void * p_void;

generic pointer…
no support for address arithmetic

STRINGS

"ABC" is a string stored somewhere….

***"ABC"** is equiavlent to **'A'**
"ABC"[0] is equivalent to **'A'**
"ABC"[2] is equiavlent to **'C'**

String Processing Functions

#include <string.h>

strcpy strncpy strlen strcat

strncat strcmp strncmp

Dynamic Memory Allocation

```
char   s[n];   /* WRONG ..
                  where n is a variable… */
char *s;

s = malloc (n);  /* request an area of n bytes
                    and sets s to point at it     */
float *f;

f = malloc( 100 *  sizeof(float) );
```

malloc returns **NULL ≡ 0** if failure

I/O Strings

```
gets(s)  /* gets text into s[0] … s[…] */

puts(s)
```

2D Arrays

```
int X[5][7];
```

stored first row all columns
2nd row all columns

```
int X[5][4] = {{0,3,7,9},
               {7,4,2,1}};
```

C PreProcessor -- # lines

```
#define  symbol  value

#define symbol(x,y)  (x+x)*y

z = symbol(2,3);
```

equivalent to

```
z = (2+2)*3;
```

Problems if the actual x and y are expressions

…. Because ...

#define symbol(x,y) (x+x)*y

z = symbol(2,5+7);

z =(2+2)*5+7;

therefore z = 4*5 +7 = 27 … NOT
z = (2+2) * 12 = 48 !!

therefore

#define symbol(x,y) ((x)+(x))*(y)

More macro features --

#

#define mymac(x) #x

together with … **mymac(fred)**

causes **"fred"**

i.e. actual argument in quotes

##

#define newmac(x,y) x##y

togther with … **newmac(bus,7)**

causes

bus7

i.e. concatenates tokens...

Conditional Compilation

Two variants of the code in one file, perhaps for different operating systems…..

#if constant_expression
…….
#else
…….
#endif

also an **#elif**

One use (trick)

Sometimes used to comment old code while testing..

#if 0
.. the old code sits here….
#endif

More …..

#if defined(name)
#if !defined(name)

equivalent….

#ifdef name
#ifndef name
#define fred
#undef
#line …..
#error …..