CS23710 C Programming (and UNIX) Batch Four

David Price

Computer Science

Arithmetic Conversion

X operator **Y**

Integer Promotion

char or **short** promoted to **integer**
unsigned short promoted to **integer** or **unsigned**

Arithmetic Conversion

X operator Y

Type Conversion

either **Long Double**, other to **LD**
either **Double**, other to **Double**
either **Float**, other to **Float**
Long & **Unsigned**, **Unsigned** to **Long**
or both -> **Unsigned Long**
Either **Long**, other -> **Long**
Either **Unsigned**, other to **Unsigned**

ONLY FIRST MATCHING RULE APPLIED

Cast Operator

Forced type conversions, “coercion”

(type name) expression

e.g.

```
float x;
 j=8 ; k=3;
 x = j / k;      /* then x is equal to 2.0 */

x= (float) j / (float) k ;   /* then X is 2.66666 */
```

Note: cast operator has second highest precedence

& and * operators

The & operator allows us to get the address of a variable, I.e. gets us a pointer to a data area. Conversely, the ***** operator lets us get at the data pointed to by a pointer.

```
main()
{ int x, y, * intptr;
        x = 5;
        intptr = &x;
        y = * intptr;
}
```

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;
```

Pointers to Structures and -> operator

struct mystruct new_struct ;
struct mystruct * struct_ptr;
struct_ptr = & new_struct;
then we can either write
new_struct.c = 23;
or
(*struct_ptr).c = 23;
or
struct_ptr -> c = 23;
and all are the same !

More on Structures

struct mystruct p, q;

We are allowed to copy complete structures, even if they contain arrays !

Program Organisation - Functions etc.

Program

is a collection of functions and global variables etc.

No nested functions.

Program Organisation

Declarations and Definitions

Two types of declarations
(can be local or global)

1/. Old (simple) K&R (with no parameter checking)

2/. New ANSI function prototypes

(but you can still use type 1)

Old Version

```
/* simple K&R version */
#include <stdio.h>
main()
{ float p;
  float triple();
  p = triple(2.7);
  printf("p = %f\n", p);
}
float triple(x)
float x;
{    return 3.0 * x ;
}
```

New Version

```
/* ANSI version */
#include <stdio.h>
main()
{ float p;
  float triple(float x);
  p = triple(2.7);
  printf("p = %f\n", p);
}
float triple(float x)
{    return 3.0 * x ;
}
```

Function Local Variables

1/. variables defined in a function

2/. parameters - initialised as the function is called.
“call by value” mechanism
“actual” parameters not altered by function

Argument Types

1/. if K&R then the programmer must get it correct
(but float / double, char/int)

2/. if ANSI then automatic type conversions
if possible

Altering Variables from Functions

1/. Remember “call by value”
2/. Need to pass an address (pointer) to the location to be altered - I.e. use &

definition might be like….

```
int fun (iptr, jptr)
int * iptr, *jptr;
{  ………….
}
```

Default Function Type

int is the default type of all functions if not specified otherwise.

Function return values are discarded if not used.

New void type in ANSI C

a). to say functions have no return value

b) used in prototypes if no parameters

```
void   newfun (void);
```

Variable Initialisation

Global and static initialised to zero
(both have permanent storage)

Automatic - only initialised if you specify
- done on each creation

```
int fred;
myfun()
{ int xxx=0;       /* automatic - initialised each call */
   static int  yyy=0;     /* static - initialised once */
…..
}
```

Variable Initialisation

Initialising arrays, o.k. on newer compilers

```
int a[10]={1,2,7,43};
```

(rest get set to zero)

Initialising Automatic Variables

Simple automatic variables can be initialised
by any expression.

```
int fun(x);
int x;
{ int y = 3.2*x;
         y= y*2.7
         return y;
}
```

Separate Compilation - Multiple Files / Modules

Functions by default are external (visible to the linker)

Declare (note not define) external variables by

```
extern int x;
```

can then use x

All global variables are external (visible to the linker)

Private Variables

The keyword **static** has another use.

Global variables and (all) functions can be made ***private*** to the file in which they are declared by prefixing their definition by the keyword static.