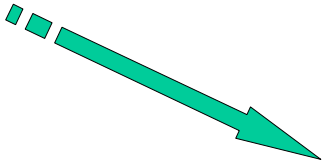
Lecture 12



1. Introduction
2. Binary Representation
3. Hardware and Software
4. High Level Languages
5. Standard input and output
6. Operators, expression and statements
7. Making Decisions
8. Looping
9. Arrays
10. Basics of pointers
11. Strings
12. Basics of functions
13. More about functions
14. Files
14. Data Structures
16. Case study: lottery number generator

Basics of Functions

- We should all be familiar with mathematical functions

  $f(x) = x^2 - 3x + 5$

  – here f denotes the function
  – x is called its argument
  – and the expression $x^2 - 3x + 5$ is an algorithm which describes how to calculate the functions value
- Note that x is merely a placeholder and we could write $f(z) = z^2 - 3z + 5$ *or* $f(p+1) = (p+1)^2 - 3(p+1) + 5$

Basics of Functions

- We can also have functions of several variables

  $g(x,y,z) = x^2 + y^2 + z^2$

  – where there are several variables x,y,z
- Some mathematical functions have special symbols e.g. e^x, *log x, sin x,* $|x|$ *etc* but the principle remains the same
- In C we have a similar idea
- Functions take the form of

```
[return type] [function name] ( [argument list…] )
{
    /* Body of the function where calculations are made*/
    return ; /* functions returns some value*/
}
```

Basics of Functions

- So we can write a function which calculates the square of a number as

```
float square(float x)
{
   return x*x;
}
```

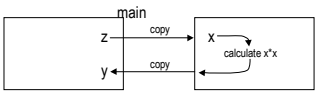
- which is like $f(x) = x^2$
  – this takes a float parameter x (which is the place holder) and returns another float value

Basics of Functions

- The function is "called" as follows

```
float y, z=3.0f;
y=square(z);
```

- Notice that we have used Z as the argument and have assigned the return value to y.
- Diagrammatically:



The argument z is copied to the parameter (placeholder) x

The return value is copied to y

funct1.c

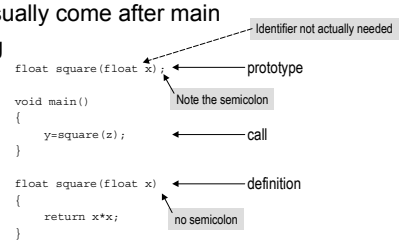
```
/* Example: simple functions */
/* Functions taking one or more arguments and returning a value */
#include <stdio.h>

/* Function prototypes: */
float square(float x);      /* computes x squared */
float cube(float x);        /* computes x cubed */
float power(float x, int n); /* computes x to power n >= 0 */

main()
{
  float X;
  int N;

  printf("Enter a floating point number: ");
  scanf("%f", &X);
  printf("Enter an integer >= 0: ");
  scanf("%i", &N);

  printf("The square of %f is %f\n", X, square(X));

  printf("The cube of %f is %f\n", X, cube(X));

  printf("%f to the power %i is %f\n", X, N, power(X, N));
}

/* Function definitions: */
float cube(float x)  /* computes x cubed */
{ return x * x * x;
}

float square(float x) /* computes x squared */
{ return x * x;
}

float power(float x, int n)  /* computes x to power n >= 0*/
{ /* Note: no check is made for n >= 0! */
  float t = 1;

  while (n > 0)
  {
    t *= x;
    n--;
  }
  return t;
}
```

Function Prototypes

- Recall that we must declare variables (primarily to give them a type) before using them.
- Similarly we must declare functions before using them
- We could do this by putting all functions at the start of the program (before main)
- However this has disadvantages
  - all the details are given before the general outline
  - two function could call each other so it would be impossible to get them in the right order

Function Prototypes

- To avoid this C has function prototypes which go near the start, while the function definitions usually come after main
- eg



Functions with no Arguments

- Sometimes a function has no argument, but returns a value
- To cope with this the void keyword is used
- VOID is used like a data type but means "absence of type" or "no type"

```
int rand(void);
……
int i;
i=rand();
```

- The empty parentheses ( ) are needed!

```
/* Example: simple functions */
/* Function taking no arguments but returning a value */

#include <stdio.h>
#include <stdlib.h> /* for rand */

/* The prototype of rand is int rand(void); */

main()
{
  int n;

  puts("Some random numbers:");
  for (n = 0; n < 10; n++)
    printf("%i ", rand());
  putchar('\n');
}
```

funct2.c

Functions with no Return Value

- Again, void is used to denote a lack of return type for a function.
- The keyword return is used without a following expression or can be omitted if your lazy/sloppy

```
void stars(int n);
……
stars(3);
```

```
/* Example: simple functions */
/* Function taking an argument but returning no value */

#include <stdio.h>

void stars(int n);  /* print a line of n stars */

main()
{
  int n;

  puts("How many stars?");
  scanf("%i", &n);

  stars(n);
}

void stars(int n)  /* print a line of n stars */
{
  while (n-- > 0)
    putchar('*');
  putchar('\n');
  return;
}
```

funct3.c

Functions with no Arguments and no Return Value

```
void print20stars(void);
……
print20stars();
```

```
/* Example: simple functions */
/* Function taking no arguments and returning no value */

#include <stdio.h>

void print20stars(void);  /* print a line of 20 stars */

main()
{
   int k;

  for (k = 0; k < 10; k++)
     print20stars();
}


void print20stars(void)  /* print a line of 20 stars */
{
  puts("********************");
  return;
}
```

funct4.c

Multiple Return Statements

- It is sometimes convenient to have multiple returns
  - e.g. in an if…else or switch statement

```
/* Example: simple functions */
/* Function with multiple return statements */

#include <stdio.h>

int pins(int tn);  /* IC pins lookup */

main()
{
  int ic, p;

  printf("Enter an IC type number: ");
  scanf("%i", &ic);

  p = pins(ic);

  if (!p)
    printf("IC type %i is not recognised.\n", ic);
  else
    printf("IC type %i has %i pins.\n", ic, p);
}
```

```
int pins(int tn)  /* IC pins lookup */
{
  /* Returns the number of pins for an IC of type
number tn. If the type is not recognised, returns
zero. */

  switch (tn)
  {
    case 555: case 741: case 748:
      return 8;
    case 556: case 7400: case 7414:
      return 14;
    case 7448: case 7485:
      return 16;
    case 6502: case 6800: case 8085: case 8088:
      return 40;
    default:
      return 0; /* easily tested */
  }
}
```

funct5.c

Functions Can Call Functions

- Functions need not be called directly from main
- Functions can be called from anywhere, even each other
  - This allows us to break up a job into manageable parts

```
          *
         ***
        *****
       *******
      *********
     ***********
    *************
   ***************
  *****************
          *
          *
       *******
        *****
         ***

  Merry Christmas!
```

xmascard.c

```
/* Example: draws a Christmas card */
/* Function calling another function */

#include <stdio.h>

void draw_tree(void);          /* draws a tree */
void starline(int sp, int st); /* prints sp spaces then st stars */

main()
{
  draw_tree();
  puts("* Merry Christmas!");
}

void draw_tree(void)  /* draws a tree */
{
  int i;

  for (i = 1; i < 19; i += 2)
     starline(9 - i/2, i);
  for (i = 1; i < 3; i++)
     starline(9, 1);
  for (i = 7; i > 1; i -= 2)
     starline(9 - i/2, i);
  putchar('\n');
  return;
}

void starline(int sp, int st)  /* prints sp spaces then st stars */
{
  while (sp-- > 0)
     putchar(' ');
  while (st-- > 0)
     putchar('*');
  putchar('\n');
  return;
}
```

Function Libraries

- Its very useful to have libraries of common functions.
  eg <stdio.h> functions which weve used extensively
- The headerfile (*.h) contains the prototypes
  - stdio.h standard input and output functions
  - string.h string manipulation functions
  - math.h common mathematical functions

math.c

```
/* Example - mathematical functions in the <math.h> library */
/* At the Unix prompt, enter 'man math' for details */

#include <stdio.h>
#include <math.h>  /* contains function prototypes etc. */

main()
{
  printf("fabs(-2.6) = %f\n", fabs(-2.6));   /* absolute value */
  printf("ceil(7.5) = %f\n", ceil(7.5));     /* round up */
  printf("floor(7.5) = %f\n\n", floor(7.5)); /* round down */

  printf("HUGE_VAL = %lf\n\n", HUGE_VAL);    /* "infinity" */
  printf("sqrt(0.5) = %f\n", sqrt(0.5));     /* square root */
  printf("fmod(2.7, 0.5) = %f\n\n", fmod(2.7, 0.5)); /* remainder */

  printf("sin(0.5) = %f\n", sin(0.5));   /* sine */
  printf("cos(0.5) = %f\n", cos(0.5));   /* cosine */
  printf("tan(0.5) = %f\n\n", tan(0.5)); /* tangent */

  printf("acos(0.5) = %f\n", acos(0.5));     /* arccosine */
  printf("asin(0.5) = %f\n", asin(0.5));     /* arcsine */
  printf("atan(0.5) = %f\n", atan(0.5));     /* arctangent */
  printf("atan2(1, 2) = %f\n\n", atan2(1, 2)); /* arctangent */

  printf("exp(0.5) = %f\n", exp(0.5));   /* exponential */
  printf("sinh(0.5) = %f\n", sinh(0.5)); /* hyperbolic sine */
  printf("cosh(0.5) = %f\n", cosh(0.5)); /* hyperbolic cosine */
  printf("tanh(0.5) = %f\n\n", tanh(0.5)); /* hyperbolic tangent */

  printf("log(0.5) = %f\n", log(0.5));     /* natural logarithm */
  printf("log10(0.5) = %f\n\n", log10(0.5)); /* base 10 logarithm */

  printf("pow(0.5, 2.4) = %f\n\n", pow(0.5, 2.4)); /* power */
}
```

Why doesn't this work

funct6.c

```
#include <stdio.h>

void square_it(int num);

main()
{
  int x = 6;

  printf("[main] x = %i\n", x);
  square_it(x);
  printf("\n[main] x squared = %i\n", x);

  return;
}


void square_it(int num)
{
  printf("\n[square_it] num = %i\n", num);
  num *= num; /* multiply num by itself */
  printf("\n[square_it] num squared = %i\n", num);

  return;
}
```