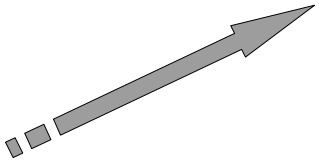
Lecture 6



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

Arithmetic Operators

- C has a number of arithmetic operators which are used to combine variables and constants into expressions
  - Unary operators +, -
    - e.g. +x, -x
  - Binary operators +, -, *, /, %
    - e.g. x+y*z
- Integer expression examples

  1+2 = 3
  3*4 = 12
  17/5 = 3
  17%5 = 2
- Note that integer division discards any fractional part. The modulus operator can be useful for many things e.g. x%2 is 0 if x is even, 1 if x is odd

Operator Precedence

- If we have several operators in an expression, what order are they processed in

  e.g. does 1+2*3 evaluate to 9 or 7 ?
- C has fixed rules for this, see pg 22 of notes

  ( ), *, /, %, +, -
- We can force the order using parentheses which have top priority
- For operators of equal precedence the evaluation is from left to right

intarith.c

```
/*  Example: integer arithmetic operators */

/* Old Imperial measures: miles, furlongs, chains,
yards, feet and inches.

1 mi = 8 fur; 1 fur = 10 ch; 1 ch = 22 yd;
1 yd = 3 ft; 1 ft = 12 in */

#include <stdio.h>

#define FUR_PER_MI 8L
#define CH_PER_FUR 10L
#define YD_PER_CH 22L
#define FT_PER_YD 3L
#define IN_PER_FT 12L

/* Note: these are longs to avoid overflow later.
(The suffix L means the value should be treated
as a long.) */

main()
{
  int a = 5, b = 3;
  long x, remainder;
  int inches, feet, yards, chains, furlongs, miles;

  printf("a = %i, b = %i\n\n", a, b);

  printf("a + b = %i\n", a + b);
  printf("a - b = %i\n", a - b);
  printf("a * b = %i\n", a * b);
  printf("a / b = %i\n", a / b);
  printf("a %% b = %i\n", a % b);

  printf("\nEnter a number of inches: ");
  scanf("%li", &x);
```

```
  miles = x / IN_PER_FT / FT_PER_YD
            / YD_PER_CH / CH_PER_FUR
            / FUR_PER_MI;
  remainder = x % (IN_PER_FT * FT_PER_YD
                   * YD_PER_CH * CH_PER_FUR
                   * FUR_PER_MI);

/* Note: if the above constants are defined as ints,
overflow might occur when their product is computed.
This is because the product of ints is itself an int. */

  furlongs = remainder / IN_PER_FT / FT_PER_YD
                       / YD_PER_CH / CH_PER_FUR;
  remainder %= (IN_PER_FT * FT_PER_YD
                * YD_PER_CH * CH_PER_FUR);

  chains = remainder / IN_PER_FT / FT_PER_YD
                     / YD_PER_CH;
  remainder %= (IN_PER_FT * FT_PER_YD * YD_PER_CH);

  yards = remainder / IN_PER_FT / FT_PER_YD;
  remainder %= (IN_PER_FT * FT_PER_YD);

  feet = remainder / IN_PER_FT;
  inches = remainder % IN_PER_FT;

  puts("\nConverted to old fashioned units:");
  printf("%li inches = %i mi, ", x, miles);
  printf("%i fur, %i ch, ", furlongs, chains);
  printf("%i yd, %i ft, %i in\n", yards, feet, inches);
}
```

Traps for the unwary

```
/* BUG ZONE!!!
Example: integer arithmetic */

#include <stdio.h>

main()
{
  int r1 = 22000;
  int r2 = 10000;
  int r3 = 15000;
  int r;

  puts("Three resistors in parallel");
  printf("%i || %i || %i\n\n", r1, r2, r3);

  puts("Method 1: r = (r1 * r2 * r3)/(r1*r2 + r2*r3 + r3*r1)\n");
  r = (r1 * r2 * r3)/(r1*r2 + r2*r3 + r3*r1);  /* BUG */
  printf("Total resistance = %i\n\n", r);

  puts("Method 2: r = 1/(1/r1 + 1/r2 + 1/r3)\n");
  r = 1/(1/r1 + 1/r2 + 1/r3);  /* BUG */
  printf("Total resistance = %i\n", r);
}
```

```
/* BUG ZONE!!!
Example: integer arithmetic */

#include <stdio.h>

main()
{
  int StudentsInClass = 116;
  int MaxGroupSize = 5;
  int NumberOfGroups;
  int passes;
  int GirlsWithBrownEyes;

  NumberOfGroups = StudentsInClass/MaxGroupSize; /* BUG */

  printf("%i students at %i max. per group means %i groups\n\n",
    StudentsInClass, MaxGroupSize, NumberOfGroups);

  /* Three-quarters of the class pass the exam: */

  passes = 3/4 * StudentsInClass;  /* BUG */
  printf("%i students passed the exam\n\n", passes);

  /* Half the students are female, and
  half of these have brown eyes */

  GirlsWithBrownEyes = StudentsInClass / 2*2;  /* BUG */

  printf("There are %i brown-eyed girls in the class\n",
    GirlsWithBrownEyes);
}
```

Floating point expressions

- arithmetic with floats or doubles works more or less as expected, however we cant use % operator

```
/* Example: floating point arithmetic */

#include <stdio.h>

main()
{
  float r1 = 22000;
  float r2 = 10000;
  float r3 = 15000;
  float r;

  puts("Three resistors in parallel");
  printf("%5.0f || %5.0f || %5.0f\n\n", r1, r2, r3);

  puts("Method 1: r = (r1 * r2 * r3)/(r1*r2 + r2*r3 + r3*r1)\n");
  r = (r1 * r2 * r3)/(r1*r2 + r2*r3 + r3*r1);
  printf("Total resistance = %7.2f\n\n", r);

  puts("Method 2: r = 1/(1/r1 + 1/r2 + 1/r3)\n");
  r = 1/(1/r1 + 1/r2 + 1/r3);
  printf("Total resistance = %7.2f\n", r);
}
```

Type Casting

- Conversion between data types (eg `int` to `float`) is called casting. It can be :
  - implicit (done automatically)
    `int x=1.34;`
  - explicit (we force it to happen)
    `int x=(int)1.34;`
- Note:
  `int -> float` is straightforward, e.g. 32 -> 32.000
  `float -> int` involves truncation, e.g. 32.73 -> 32

typecast.c

```
/* Example: type casting */

#include <stdio.h>

main()
{
  float x = 34.256, y;
  int n = 27, m;
  char c = 'A';

  /* Implicit casting: */

  m = x * n; /* implicit (int) cast */
  y = n/2; /* implicit (float) cast */
  printf("m = %i, y = %6.2f\n\n", m, y);

  m = c * n; /* implicit (int) cast */
  printf("m = %i\n\n", m);

  /* Explicit casting: */

  m = (int)x * n; /* explict (int) cast */
  y = (float)n/2; /* explicit (float) cast */
  printf("m = %i, y = %6.2f\n\n", m, y);

  y = (float)c + x;
  printf("y = %6.3f\n\n", y);

  /* Explicit and implicit casting: */

  m = (float)n/2;
  y = ((int)x + 1)/2 + x;
  printf("m = %i, y = %6.3f\n\n", m, y);
}
```

Increment and Decrement Operators

- We could write `x=x+1` (this is valid C)
- But shorthand versions are:
  - pre increment : `++x`
  - post increment : `x++`
- The difference is that with `x++` the value of x is used first then incremented; the reverse for `++x`

eg

```
int a=5, b;
b=a++;
/*Now a is 6, b is 5*/
```

```
int a=5, b;
b=++a;
/*Now a is 6, b is 6*/
```

- There is also a decrement operator -- which follows the same rules

Assignment Operators

- We have already seen the basic assignment operator '=' e.g. `x=y`
- C also has some useful shorthand's

`a+=b`	means	`a=a+b`
`a-=b`	means	`a=a-b`
`a*=b`	means	`a=a*b`
`a/=b`	means	`a=a/b`
`a%=b`	means	`a=a%b`

- Something that can be used on the LHS of an assignment is called an lvalue e.g.
  x is an lvalue
  `x/2, 3+y, a+b` are not
- Because assignments are operators several can be combined in a single statement:
  `x=y=z=0;`

Relational Operators

- They are >, <, >=, <=, ==, !=
- Expressions involving these operators evaluate to true or false, e.g.
  - 27>21 is true
  - 27<=3 is false
- In C there is no logical or boolean data type, but integers can be used
  - 0 -> false
  - anything else -> true
- However, these relational operators give 1 for true

relation.c

```
/*  Example: relational and logical operators */

#include <stdio.h>

main()
{
  int a = 5, b = 6, c = 7;

  puts("int a = 5, b = 6, c = 7;\n");

  printf("The value of  a > b  is         \t%i\n\n", a > b);
  printf("The value of  b < c  is         \t%i\n\n", b < c);

  printf("The value of  a + b >= c  is    \t%i\n\n", a + b >= c);
  printf("The value of  a - b <= b - c  is\t%i\n\n", a - b <= b - c);

  printf("The value of  b - a == b - c  is\t%i\n\n", b - a == b - c);
  printf("The value of  a * b != c * c  is\t%i\n\n", a * b < c * c);
}
```

Statements

- Expressions can be made into statements by a suffixing semicolon
- Statements can be simple or complex

e.g.

```
x;
y++;
tall=height>180;
poly=a*x*x+b*x+c;
```

Blocks (or Compound Statements)

- These are simply one or more statements enclosed within braces { } to group them together.

```
{
    x=3;
    y=x+p;
    ++x;
}
```

- The compound statement is treated as a single entity, as well see in the next 2 lectures