Fundamental data types

Chapter 3

Beginning of a program

```
#include <stdio.h>
int main(void)
{
   int a, b, c;                        /* declaration */
   float x, y = 3.3f, z = -7.7f;       /* with initialization*/
   printf( "Input two ints: " );       /* function call */
   scanf( "%d%d", &b, &c );            /* function call */
   a = b + c;
   x = y + z;
   return 0;
}
```

Fundamental Data Types

Short form

`char`	`signed char`	`unsigned char`
`short`	`int`	`long`
`unsigned short`	`unsigned`	`unsigned long`
`float`	`double`	`long double`

The long form includes e.g. `signed short int`, `unsigned long int`, `signed int` etc, but this form is rarely used.

Functionality Groups

Integral Types:

`char`	`signed char`	`unsigned char`
`short`	`int`	`long`
`usigned short`	`unsigned`	`unsigned long`

Floating Types:

`float`	`double`	`long double`

Arithmetic Types:

integral types + floating types

Constants

`char`	`'a', 'x', '0', '&', '\n'`
`int`	`1, 0, -54, 4234567`
`long`	`0l, 123l, -77661`
`unsigned`	`0u, 23u, 3000000000u`
`float`	`1.2f, .2f, 1.f, 3.14159f`
`double`	`1.0, -3.1412`
`long double`	`1.0l, -2.44331`

Characters and the data type `char`

Some Character Constants and their Integer Values

In C, variables of any integral type can be used to represent characters. In particular **`char`** and **`int`** are used.

character:	`'a'`	`'b'`	`'c'`	...	`'z'`
integer value:	97	98	99	...	112
character:	`'A'`	`'B'`	`'C'`	...	`'Z'`
integer value:	65	66	67	...	90
character:	`'0'`	`'1'`	`'2'`	...	`'9'`
integer value:	48	49	50	...	57
character:	`'&'`	`'*'`	`'+'`		
integer value:	38	42	43		

Some Special Character Constants and their Integer Values

Name of Character	Written in C	Integer Value
alert	`\a`	7
backspace	`\b`	8
backslash	`\\`	92
horizontal tab	`\t`	9
newline	`\n`	10
null character	`\0`	0
double quote	`\"`	39

Some Character Constants and their Integer Values

```
char   c = 'a';
```

This variable can be printed both as character or integer

```
printf("%c", c);    /* a is printed */
printf("%d", c);   /* 97 is printed */
printf("%c%c%c", c, c+1, c+2);  /* abc is printed */
```

Some Character Constants and their Integer Values

character:	'A'	'B'	'C'	...	'Z'
integer value:	65	66	67	...	90

```
char c = 0;
int i = 0;

for ( i = 'a'; i <= 'z'; ++i )
  printf( "%c", i ); /* ab ... z is printed */
for ( c = 65; c <= 90; ++c )
  printf( "%c", c ); /* AB ... Z is printed */
for ( c = '0'; c <= '9'; ++c )
  printf( "%d", c );  /* 48 49 ... 57 is printed */
```

The integer data types: `int`, `short`, `unsigned` and `long`

`int` is typically 2 or 4 bytes.

`short` is typically 2 or 4 bytes.

`long` typically 4 bytes

`unsigned` has the size of **`int`**.

Going over the Limit

```
#include <stdio.h>
#include <limits.h>
int main( void )
{
   int i = 0;
   unsigned u = UINT_MAX; /*Typically 4294967295 */

   for ( i = 0; i < 5; ++i )
       printf("%u + %d = %u\n", u, i, u + i );
   for ( i = 0; i < 5; ++i )
       printf("%u * %d = %u\n", u, i, u * i );
   return 0;
}
```

Going over the Limit

Output:

```
4294967295 + 0 = 4294967295
4294967295 + 1 = 0
4294967295 + 2 = 1
4294967295 + 3 = 2
4294967295 + 4 = 3

4294967295 * 0 = 0
4294967295 * 1 = 4294967295
4294967295 * 2 = 4294967294
4294967295 * 3 = 4294967293
4294967295 * 4 = 4294967292
```

The floating data types

- `float`
- `double`
- `long double`

Examples:

```
3.14159
314.159e-2f
0e0
1.
```

Floating data types can be described by:

- precision
- range

Limited precision

Floats has limited precision hence strange phenomena may occur:

```
#include <stdio.h>
int main()
{
   int i = 0;
   float f = 0;
   for ( i = 0; i < 100; ++i )
       f += 0.01f;
   printf( "%f\n", f );
   return 0;
}
```

Output:

```
0.999999
```

Special Float Values

NaN – Not a Number - represents an illegal value

```
printf("%f\n", sqrt(-1));
```

will print

`-1.#IND00` or `NAN`

INF – infinity

```
printf("%f\n",1.0/0);
```

or

```
printf("%f\n",-log(0));
```

will print

`1.#INF00` or `INF`

The use of `typedef`

The `typedef` mechanism allows to associate a type with an identifier:

```
typedef  char              uppercase;
typedef  int               INCHES;
typedef  unsigned long     size_t;
```

Each of these identifiers can be used later to declare variables, e.g.

```
uppercase u;
INCHES    length, width, height;
```

The `sizeof` Operator

Find the number of bytes needed to store an object.

```
#include <stdio.h>
int main(void)
{
   printf( "The size of some fundamental types is
   computed.\n\n");
   printf( "char: %3d byte \n", sizeof(char) );
   printf( "short: %3d bytes\n", sizeof(short) );
   printf( "int: %3d bytes\n", sizeof(int) );
   printf( "long: %3d bytes\n", sizeof(long) );
   printf( "unsigned: %3d bytes\n", sizeof(unsigned) );
   printf( "float: %3d bytes\n", sizeof(float) );
   printf( "double: %3d bytes\n", sizeof(double) );
   printf( "long double: %3d bytes\n",sizeof(long double));
   return 0;
}
```

compute the size of some fundamental types

run on this laptop, using cygwin:

```
The size of some fundamental types is computed.

char: 1 byte
short:  2 bytes
int:  4 bytes
long:  4 bytes
unsigned:  4 bytes
float:  4 bytes
double:  8 bytes
long double:   8 bytes
```

Guarantees about storage of fundamental types

```
sizeof(char) == 1
sizeof(short) <= sizeof(int) <= sizeof(long)
sizeof(signed) == sizeof(unsigned) == sizeof(int)
sizeof(float) <= sizeof(double) <= sizeof(long double)
```

getchar and putchar

```
#include <stdio.h>

int main(void)
{
  int   c = 0;

  while ( ( c = getchar() ) != EOF )
  {
     putchar( c );
     putchar( c );
  }
  return 0;
}
```

Look at file 02_double_out.c

Mathematical Functions

There are no built mathematical functions in C. Functions such as

`sqrt()` `pow()` `exp()` `log()`

`sin()` `cos()` `tan()`

are part of the math library declared in `<math.h>`.

All the functions use doubles

```
#include <stdio.h>
#include <math.h>
int main(void)
{
   double x = 0;
   printf( "\n%s\n%s\n\n", "The square root of x and x
   raised","to the x power will be computed.");
   while ( 1 )
   {
       printf( "Input x:   " );
       if (scanf( "%lf", &x ) != 1)
          break;
       if ( x >= 0.0 )
          printf("\n%14s%15.8e\n%14s%15.8e\n
   %14s%15.8e\n\n",
             "x = ", x,
             "sqrt(x) = ", sqrt(x),
             "pow(x, x) = ", pow(x, x) );
       else printf( "\nSorry, your number must be
   nonnegative.\n\n" );
   }
   return 0;
}
```

The Result of the Program

```
The square root of x and x raised
to the x power will be computed.


Input x:  2

          x = 2.0000000e+00
    sqrt(x) = 1.4142136e+00
  pow(x, x) = 4.0000000e+00

Input x:
```

Conversions and Casts

Integral promotions

A **`char`** or **`short`** (**`signed`** or **`unsigned`**) can be used in any expression where an **`int`** or **`unsigned int`** is used.

The usual arithmetic conversions

These can occur when operands of binary operators are evaluated. E.g. if **`i`** is an **`int`** and **`f`** is **`float`** , then in the expression **`i + f`**, the operand **`i`** is promoted to float.

The usual arithmetic conversions

If either operand is a **`long double`** the other operand is ***converted*** to **`long double.`**

Otherwise, if either operand is a **`double`** the other operand is ***converted*** to **`double.`**

Otherwise, if either operand is a **`float`** the other operand is ***converted*** to **`float.`**

Otherwise, the "integral promotions" are performed on both operands:

- If either operand is an **`unsigned long`** the other operand is ***converted*** to **`unsigned long.`**
- Otherwise if one operand has type **`long`** and the other operand has type **`unsigned`** then one of two possibilities occurs:
  - If a **`long`** ***can represent*** all the values of an **`unsigned`**, then the operand of type **`unsigned`** is ***converted*** to **`long`**.
  - If a **`long`** ***cannot represent*** all the values of an **`unsigned`**, then ***both*** operands are ***converted*** to **`unsigned long`**.
- Otherwise, if either operand is of type **`long`**, the other operand is ***converted*** to **`long`**.
- Otherwise, if either operand is of type **`unsigned`**, the other operand is ***converted*** to **`unsigned`**.
- Otherwise, both operands have type **`int`**.

Examples for arithmetic conversion

```
char c;       short s;          int  i;
long l        unsigned u;       unsigned long ul;
float  f;     double d;         long double  ld;
```

Expression	Type	Expression	Type
`c - s / i`	`int`	`u * 7 - i`	`unsigned`
`u * 2.0 - i`	`double`	`f * 7 - i`	`float`
`c + 3`	`int`	`7 * s * ul`	`unsigned long`
`c + 5.0`	`double`	`ld + c`	`long double`
`d + s`	`double`	`u - ul`	`unsigned long`
`2 * i / l`	`long`	`u - l`	*system dependent*

Casts

In addition to implicit conversion, there are explicit conversions, called casts. For example if **`i`** is an **`int`**, then **`(double)i`** will cast **`i`** so the expression has a type double.

```
(long)('A' + 1.0);
f = (float) ((int)d + 1);
d = (double) i/3;
(double)(x = 77);
```

The cast operator is unary, and has the same precedence of other unary operators. So for example

`(float) i + 3` is equivalent to **`((float) i) + 3`**

Decimal, Hexadecimal, Octal conversions

```
/* decimal, hexadecimal, octal conversions */
#include <stdio.h>
int main(void)
{
   printf("%d %x %o\n", 19, 19, 19);   /* 19 13 23 */
   printf("%d %x %o\n",0x1c,0x1c,0x1c);/* 28 1c 34 */
   printf("%d %x %o\n",017,017,017);   /* 15  f 17 */
   printf("%d\n", 11 + 0x11 + 011);    /* 37       */
   printf("%x\n", 2097151);            /* 1fffff   */
   printf("%d\n", 0x1FfFFf);           /* 2097151  */
   return 0;
}
```