Data Type

Syntax Rules Recap

- keywords

break	double	if	sizeof	void
case	else	int	static	.....

- Identifiers

not#me	123th	
scanf	printf	
_id	so_am_i	gedd007

- Constant

122.72	'a'	'+'

- String Constants

"a string of text"	"a"

- Operators

```
() []
+  -  *  /  %
&&  ||  !            /* logic */
^  ~  &  |         /* bitwise */
Sizeof
?:         /* (n > 0) ? f : n */
* & ->
>> <<                 /* shift */
<   >   <=  >=  ==  !=
    /* relational operators */
++ --
 /* increment and decrement */
=  += - = *= /= %= >>= <<= &=
   ^= |n
```

Operators

Operator precedence (order from top to down)	Associativity
() [] . ->	left to right
! ~ ++ -- + - * & (type) sizeof	right to left
* / % (binary)	left to right
+ - (binary)	left to right
* / %	left to right
<< >>	left to right
< <= > >=	left to right
== !=	left to right
&	left to right
^	left to right
|	left to right
&&	left to right
||	left to right
?:	right to left
= += -= *= /= %= &= ^= |= <<= >>=	right to left
,	left to right

Declarations, Expressions, and Assignment

```
#include <stdio.h>

int main(void)
{
   int   a, b, c;                      /* declaration */
   float x, y = 3.3, z = -7.7;          /* declaration with
                                           initializations */

   printf("Input two integers: "); /* function call */
   scanf("%d%d", &b, &c);           /* function call */
   a = b + c;                       /* assignment */
   x = y + z;                       /* assignment */
   ……
```

Fundamental Data Types

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

- all variables must be declared before they are used
- other types (array, pointer, structure, union) are derived from the fundamental data types

Data Types and Sizes

- sizes are machine dependant
  - short and int are at least 16 bits
  - long is at least 32 bits
  - short <= int <= long
- float
  - typically 4 bytes (32bits)
  - double is 8 bytes
  - floating arithmetic is NOT always exact
    - refer <float.h> <limits.h>

Characters

- assume a single byte for a character even though it is represented as int
  - 256 distinct characters are possible

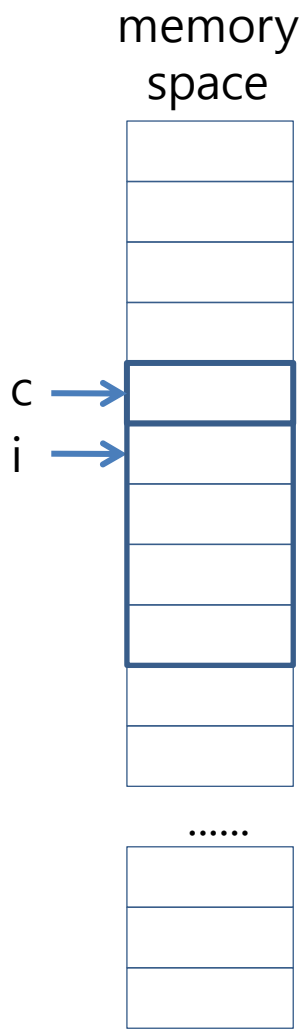
'a'	'b'	'c'	...	'z'
97	98	99		112
'A'	'B'	'C'	...	'Z'
65	66	67		90
'0'	'1'	'2'	...	'9'
48	49	50		57
'&'	'*'	'+'		
38	42	43		

name of character	written in C with \	corresponding integer value
alert (bell)	\a	7
backslash	\\	92
backspace	\b	8
carriage return	\r	13
double quote	\"	34
formfeed	\f	12
horizontal tab	\t	9
newline	\n	10
null character	\0	0
single quote	\'	39
vertical tab	\v	11
question mark	\?	63

```
char  c = 'a';
  /* ASCII code for 'a' is 01100001 */
printf("%c", c);
                    /* a is printed    */
printf("%d",c);
                    /* 97 is printed   */
printf("%c%c%c",c,c+1,c+2);
                    /* abc is printed  */
```

```
char c;
int  i;

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



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

Character Types

- ANSI C provides three types of char
  - char is either one of the followings
  - signed char -128~127
  - unsigned char 0~255
- int
  - 16 bits for small/old computers
  - 32 bit for your computers
  - what if overflow occurs
    - depends on the CPU

Suffix	Type	Example
u or U	unsigned	37U, 127u
l or L	long	37L
ul or UL	unsigned long	37UL

- Suffixes can be appended to an integer constant to specify its type
- The type of an unsuffixed integer constant is either `int`, `long`, or `unsigned long` depending how large is integer number

- ANSI C provides the three floating types to represent real numbers: `float`, `double` (working floating type), and `long double`
- A suffix can be appended to a floating constant to specify its type (without suffix, by default it will be double)

Suffix	Type	Example
f or F	float	3.7F
l or L	long double	3.7L

- Examples of floating constants

```
3.14159
314.159e-2F  /* of type float */
0e0
/* floating point zero 0.0 of type double */
1.           /* double 1.0 */
```

- Incorrect syntax for floating constants

```
3.14,159 /* comma not allowed */
314159 /* no decimal point or exponent */
.e4 /* only decimal point not allowed */
-3.14159 /* constant expression not a
constant */
```

Floating Numbers

- IEEE 754 floating point standard:
  - Single precision: (sign)(significand)*2^{exp}
    - **8** bit exponent (0~127) = (-63~64)
    - **23** bit significand
    - **1** bit sign
  - Double precision: **(11, 52, 1)**

Float: precision 6 sig. figures; range 10^{-38} to 10^{+38}

- $0.d_1d_2d_3d_4d_5d_6 * 10^n$

Double: 15, 10^{-308} to 10^{308}

- $0.d_1d_2d_3d_4d_5d_6d_7d_8d_9d_{10}d_{11}d_{12}d_{13}d_{14}d_{15} * 10^n$

Data Type Definition typedef

```
typedef     char    uppercase;
typedef     int     Inches, Feet;

uppercase   FirstChar;
Inches      length, width;
```

sizeof( ) operator

- returns the number of bytes
  - because some sizes are machine dependent
- guaranteed

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

getchar( ) and putchar( )

- defined in <stdio.h>
  - getchar() reads in a character
  - putchar() writes out a character
  - to/from the standard device

```
#include <stdio.h>

int main(void)
{
   int   c;

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

```c
#include <stdio.h>

int main(void)
{
   int   c;

   while ((c = getchar()) != EOF)
      if (c >= 'a' && c <= 'z')
         putchar(c + 'A' - 'a');
      else
         putchar(c);
   return 0;
}
```

capitalize.c

Mathematical Functions

```
#include <math.h>
#include <stdio.h>

int main(void)
{
   double   x;

   printf("₩n%s₩n%s₩n%s₩n₩n",
      "The square root of x and x raised",
      "to the x power will be computed.",
      "---");
   while (1) {                    /* do it forever */
      printf("Input x:  ");
      scanf("%lf", &x);
      if (x >= 0.0)
         printf("₩n %15s %22.15e ₩n %15s %22.15e ₩n %15s %22.15e ₩n₩n",
            "x = ", x,
            "sqrt(x) = ", sqrt(x),
            "pow(x, x) = ", pow(x, x));
      else
         printf("₩n Sorry, your number must be nonnegative.₩n₩n");
   }
   return 0;
}
```

- many mathematical functions are available from the math library
  - include <math.h>
  - link with the library " gcc –lm code.c"

Arithmetic Conversions

- Some data types are converted automatically in an expression and on an assignment

  int op int

  short op short => int

  int op float => float

- Some rules
  - small one is converted to a large one

    float op long

    long op double

    int op float

Automatic Conversions

- on an assignment d = i; i is converted to the type of d

Declarations			
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*

Cast

- you can force explicit conversions
  - (double) i
  - (long) ('A' + 1.0)
  - f = (float) ((int) d + 1) * (double)(x = 77));