1

Last lecture: C basics

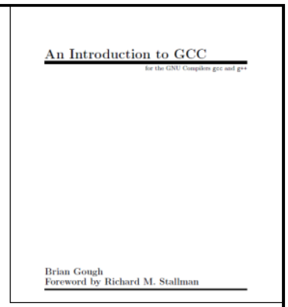
- Compile and running C programs
- Basic syntax
  - Comments
  - Variables
  - Functions
  - Basic IO functions
  - Expression
  - Statements
  - Preprocessing: # include, # define

2

YORK UNIVERSITÉ UNIVERSITY

2

gcc c compiler



- `-o` exectuableName
  `%gcc hello.c -o hello`
  `%gcc -o hello hello.c`
- default in lab: C89 + some C99 `//`
  `for (int i=0; i<10;i++)` ✗ `c99 only, not ok in lab`
  `int i=0; for (i=0; i<10;i++)` `ok in C89 and lab`

- `-std` use a standard
  `%gcc -std=c89 hello.c`   `%gcc -ansi hello.c`
  `%gcc -std=c99 hello.c`   `for (int i=0; i<10;i++) ok`
- `-Wall` (warning all)
  `%gcc -Wall hello.c`
- combine
  `%gcc -ansi -Wall hello.c -o hello`

3

3

functions

One thing to get adapted from Java (among many other things)

- ***Must be declared or defined physically before use –** different from Java*
  - ***C89, C99***

- Declaration (prototype) – describe arguments and return type, but no body
  - `int sum (int i, int j);` or `int sum(int, int);`
  - `void display(double i);` or `void display(double);`

- Definition – describe arguments and return value, and gives the code

```
int sum (int i, int j){
   return i+j;
}

void display (double i)
{ printf("this is %f", i);
}
```

- `<stdio.h>` contains declarations (prototypes) for `printf(), scanf()` etc. --- that why we “#include” it

4

functions

/* Contains declaration (prototype) of printf() */

```c
#include <stdio.h>

/* function definition */
float div (float i, float j)
{
   return i / j;
}

main()
{
  float x = 2.1, y = 3.2;
  float su  = div(x , y);
  printf( "%f / %f = %f\n", x,y, su);
}
```

Defined before (first) call

OK

5

5

functions

/* Contains declaration (prototype) of printf() */

```c
#include <stdio.h>



main()
{
  float x =2.1, y=3.2;
  float su = div (x,y);
  printf( "%f / %f = %f\n", x,y, su);
}

/* function definition */
float div (float i, float j){
   return i / j;
}
```

Not Defined or declared before (first) call

Little luckier if return int…

Defined after (first) call

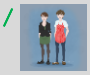
```
error: conflicting types for 'sum'
note: previous implicit declaration of 'sum' was here
```

6

functions

/* Contains declaration (prototype) of printf() */

```
#include <stdio.h>

/* function declaration */
float div (float, float); /* float div (float divd, float divor)
                              preferred for readability*/

main()
{
  float x =2.1, y=3.2;
  float su = div(x,y);
  printf( "%f / %f = %f\n", x,y, su);
}

/* function definition */
float div (float i, float j){
   return i / j;
}
```



Declared before (first) call

Defined after (first) call

7

7

Basic I/O functions

<stdio.h>

- Every program has a Standard Input: keyboard
- Every program has a Standard Output: screen
  - Can use redirection Unix < inputFile > outputFile

- **int printf (char *format, arg1, ….);**
  - Format and prints arguments on standard output (screen or > outputFile)
  - **printf("This is a test %d %f\n", x, y)**

- **int scanf (char *format, arg1, ….);**
  - Formatted input from standard input (keyboard or < inputFile)
  - **scanf("%d %f", &x, &y)**

Others (more later)

- **int getchar();**
  - Reads and returns the next char on standard input (keyboard or < inputFile)

- **int putchar(int c)**
  - Write the character c on standard output (screen or > outputFile)

YORK UNIVERSITÉ UNIVERSITY

8

8

format string

/* conversion specification */

- `printf("This is a test %d \n", x)`
  - Formats and prints arguments on standard output (screen or > outputFile)
  - Returns number of chars printed (often discarded)

- Format string contains: 1) regular chars 2) conversion specifications
  - %d **next argument is an integer (decimal)**
  - %c **next argument is a character**
  - %f **next argument is a floating point number (float, double)**
  - %s **next argument is a "string"**
  - ...

```
System.out.println("Hi " + name + ", double and triple of input " +
                   a + " is " + b + " and " + c + " respectively");
System.out.printf ("Hi " + name + ", double and triple of input " +
                   a + " is " + b + " and " + c + " respectively\n");
```

how about

```
System.out.printf("Hi %s, double and triple of input %d is %d and %d
                                   respectively\n", name, a, b, c);
```

9

Read two ints

```
#include <stdio.h>

main()
{
  float a, b;
  printf("Enter two floats separated by <><>: " );

  scanf( "%f<><>%f",   &a, &b); /* assign value to a b */
}
```

```
  scanf( "%d<><>%d",   &a, &b); ✖    get 0

  scanf( "%f<><>%f",   a, b);  ✖  segmentation fault
```

The compiler might not help much -- a warning -Wall

10

getchar, putchar

- **char, line counting**

```
#include <stdio.h>

main(){
  int c, cC, lC;
  cC = lC = 0;

  c = getchar();   /* read 1 char */
  while(c != EOF)
  {
    putchar(c);

    cC ++;

    if (c == '\n') /*a newline char*/
      lC ++;

    c= getchar(); /* read again */
  }
  printf("char:%d line:%d\n",cC,lC);
}
```

Compare directly

```
indigo 337 % a.out
hello
hello
how are you
how are you
i am good
i am good
^D
char:28  line:3
```

```
indigo 337 % cat greeting.txt
hello
how are you
i am good

indigo 338 % a.out < greeting.txt
hello
how are you
i am good
char:28  line:3
```

11

char 'a' 'b' compared directly. String not "a"== "b" ✗   Will elaborate today.

11

C basics

- Compile and running Comments
- Basic syntax
  - Comments
  - Variables
  - Functions
  - Basic IO functions
  - **Expression**
  - **Statements**
  - **Preprocessing: #include, #define**

12

YORK U UNIVERSITÉ UNIVERSITY

12

Statements

- Program to execute
  - Ended with a ;
- Expression statement (ch2)
  - `i+1;   i++;   x = 4;`
- Function call statement (ch4)
  - `printf("the result is %d");`
- Control flow statement (ch3)
  - `if else, for(), while,  do while, case switch`

Same in Java

13

13

Expression

- Formed by combining **operands** (variable, constants and function calls) using **operators** `(+ - * % > <  == != )`
- Has return values -- always
  - `x+1`
  - `i < 20`   false: 0   true: 1   `printf("%d", i<20);`
  - `sum (i+j)`
  - `x = 5     = is an operator in C (and Java)! Return value 5`
  - `x = k + sum(i,j)`   `printf("%d", x=5);`

"whenever a value is needed, any expression of the same type will do"

- `printf("sum is %d\n", i*y+2)`
- `printf("sum is %d\n", sum(i+j))`

14

Statements

- In ANSI-C (C89): <u>all</u> declarations must appear at the **<u>start</u>** of block, before <u>any</u> variable use statement.

```
{
  int i, j;
  ….
  ….
  i = 0;
  j = i+ 1
}
```

```
{
  int i;
  i = 0;
  …
  …
  int j;
  j = i+ 1
}
```

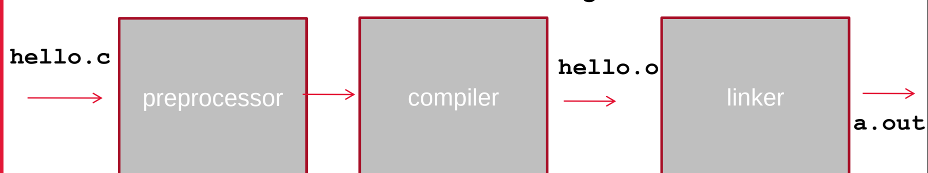
- C99 removed this restriction.
  - Declarations and statements can be mixed (as in Java,C++)
  - Legal in C99
  - OK in lab (default C89+some C99)
    `gcc hello.c`

For your information

15

How C programs are compiled

- C executables are built in three stages



Handles
#include
#define

Converts C code into binary processor instructions -- **hello.o**

- Puts multiple compiled files together
- linking **hello.o** with related system library functions. E.g. **printf()**
- creates an executable program e.g., **a.out**
- remove **hello.o**

16

16

Preprocessing: # include, #define

Textual replace/copy

Declarations/ prototypes

```
#include <stdio.h>
main()
{
    int i = 4;
    printf("this is %d\n",i);

}
```

```
int printf (..)
int scanf(..)

int getchar()
int putchar(int)

char* gets(char *)
int sprintf (..)
```

- Where is the definition (implementation) of the library functions?
  - Linked automatically for you
  - But not always e.g., math library `gcc -lm`

17

YORK UNIVERSITÉ UNIVERSITY

17

#define directive

- Syntax `#define name value`
  - Name called symbolic constant, conventionally written in upper case
  - Value can be any sequence of characters

```
#define    N    100
main() {
   int  i  = 10 + N;
}
```

```
main() {
   int  i  = 10 + 100;
}
```

Discuss later

YORK UNIVERSITÉ UNIVERSITY

18

Summary of last lecture

- Course introduction. C basics
  - Variables:
    - names don't start with digit, _ , keyword
  - Functions: declaration vs definition
  - Basic IO functions
    - **`scanf`** **&** **`printf,`**
    - **`getchar`** **`putchar`**

Same in Java

- Today's lecture:
  - C data, type, operators (Ch 2)
  - C flow controls (Ch 3) self-study

19

19

EECS2031-Software Tools

C-Types, Operators, Expressions (K&R Ch.2)

20

Outline

- Types and sizes
  - Types
  - Constant values (literals)
- Array and “strings”

- Expressions
  - Basic operators
  - Type promotion and conversion
  - Other operators
  - Precedence of operators

21

Java defines eight primitive types:

Type	Explanation
`int`	A 32-bit (4-byte) integer value
`short`	A 16-bit (2-byte) integer value
`long`	A 64-bit (8-byte) integer value
`byte`	An 8-bit (1-byte) integer value
`float`	A 32-bit (4-byte) floating-point value
`double`	A 64-bit (8-byte) floating-point value
`char`	A 16-bit character using the Unicode encoding scheme
`boolean`	A `true` or `false` value

22

Text book:
4 basic types: char, int, float, double

3 qualifiers: short, long, unsigned

C Types and sizes

- Variables and values have types

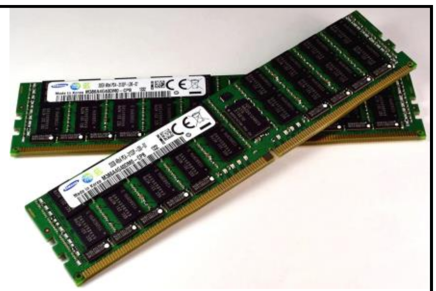
- There are two basic types in ANSI-C: integer, and floating point

  - **Integer type**
    - **char** - **character**, single byte (8 bits)
    - **short (int)** - **short integer**, 1 or 2 bytes (8 or 16 bits)
    - **int** - **integer**, usually 2 or 4 bytes (16 or 32 bits)
    - **long (int)** - **long** integer, usually 4 or 8 bytes (32 or 64 bits)

  - **Floating point**
    - **float** - single-precision, usually 4 bytes (32 bits)
    - **double** - double-precision, usually 8 bytes (64 bits)
    - **long double** - extended-precision

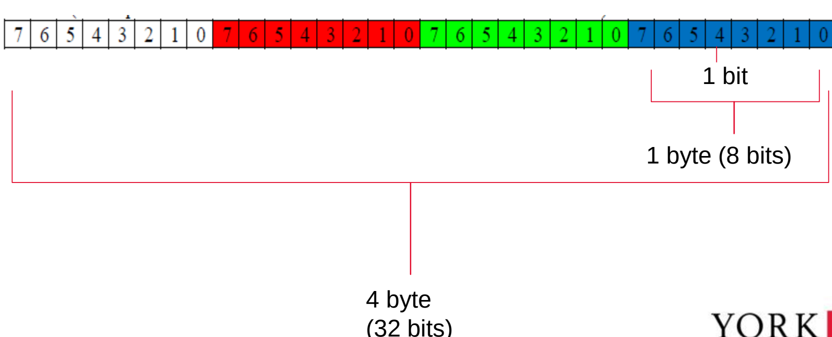
23

23



- Bit/byte/K/M/G/T

- int x;



24

24

kB	kilobyte	2^{10} = 1 024 bytes	approx. 1 000 bytes
Mb	Megabyte	2^{20} = 1 048 576 bytes	approx. 1 000 000 bytes
Gb	Gigabyte	2^{30} bytes = 1,073,741,824 bytes	approx. 1000 000 000 bytes
Tb	Terabyte	2^{40} bytes = 1,099,511,627,776 bytes	approx. 1000 000 000 000 bytes

- int x;

7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

1 bit

1 byte (8 bits)

4 byte
(32 bits)

25

25

kB	kilobyte	2^{10} = 1 024 bytes	approx. 1 000 bytes
Mb	Megabyte	2^{20} = 1 048 576 bytes	approx. 1 000 000 bytes
Gb	Gigabyte	2^{30} bytes = 1,073,741,824 bytes	approx. 1000 000 000 bytes
Tb	Terabyte	2^{40} bytes = 1,099,511,627,776 bytes	approx. 1000 000 000 000 bytes

7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

1 bit

1 byte (8 bits)

- Be careful;

26

For your information

26

Decimal Notation

- base 10 or radix 10 ... uses 10 symbols

  0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- Position represents powers of 10
- 5473_{10} or 5473

  $(5 * 10^3) + (4 * 10^2) + (7 * 10^1) + (3 * 10^0)$

Binary Notation

- base 2 ... uses only 2 symbols

  0, 1
- Position represents powers of 2
- 11010_2

  $(1 * 2^4) + (1 * 2^3) + (0 * 2^2) + (1 * 2^1) + (0 * 2^0)$ **= 26**

27

Binary representations

- 3 2 8

  10^2 10^1 10^0

  $3*10^2 + 2*10^1 + 8*10^0 =$

  $300 + 20 + 8 = 328$

  Decimal 328

7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

… 2^5 2^4 2^3 2^2 2^1 2^0

0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

Binary

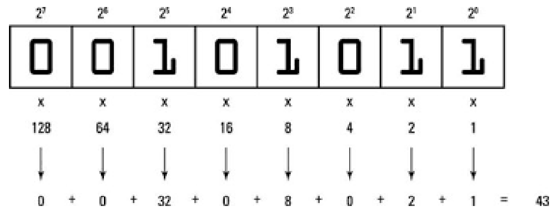
2^2 2^1 2^0

$1*2^2 + 0*2^1 + 1*2^0 =$

$4 + 0 + 1 = 5$

28

28



Binary Value				Decimal Representation	Decimal Value
				8 4 2 1	
0	0	0	0	0 + 0 + 0 + 0	0
0	0	0	1	0 + 0 + 0 + 1	1
0	0	1	0	0 + 0 + 2 + 0	2
0	0	1	1	0 + 0 + 2 + 1	3
0	1	0	0	0 + 4 + 0 + 0	4
0	1	0	1	0 + 4 + 0 + 1	5
0	1	1	0	0 + 4 + 2 + 0	6
0	1	1	1	0 + 4 + 2 + 1	7
1	0	0	0	8 + 0 + 0 + 0	8
1	0	0	1	8 + 0 + 0 + 1	9
1	0	1	0	8 + 0 + 2 + 0	10

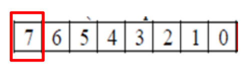
29

8 4 2 1

29

Qualifiers (modifiers) for integer type

- signed, unsigned qualifiers can be applied to integer types
  - Signed: default. Left most bit signifies sign 0: positive 1: negative
  - Unsigned: positive. Left most bit contributes to magnitude




```
▪ (signed) char
▪ (signed) int
▪ (signed) short int
▪ (signed) long  int

▪ unsigned char
▪ unsigned int
▪ unsigned short int
▪ unsigned long  int
```

Java: no direct support for unsigned int. Always signed

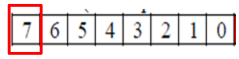
- Range?
  - **signed** $-2^{n-1} \sim 2^{n-1}-1$ 2^n values
  - **unsigned** $0 \sim 2^n-1$ 2^n values

30

30

Qualifiers (modifiers) for integer type

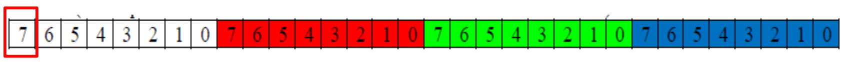
- signed, unsigned qualifiers can be applied to integer types
  - Signed: default. Left most bit signifies sign 0: positive 1: negative
  - Unsigned: positive. Left most bit contributes to magnitude



- `(signed) char`
- `(signed) int`
- `(signed) short int`
- `(signed) long int`

- `unsigned char`
- `unsigned int`
- `unsigned short int`
- `unsigned long int`

Java: no direct support for unsigned int. Always signed



`(signed) int` -2^{31}~2^{31}-1 -2145483648~ 2147483647 2^{32} values

`unsigned int` 0~2^{32}-1 0~ 4294967295 2^{32} values

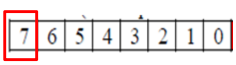
31

Max: signed 0111111….11111 Unsigned: 1111111….11111

31

Qualifiers (modifiers) for integer type

- **signed/unsigned** can be applied to char
  - **signed** char -2^7~2^7-1 /* -128 ~~ 127 */
  - **unsigned** char 0 ~2^8-1 /* 0 ~~ 255 */



signed value

Bits	Unsigned value	2's complement value
00000000	0	0
00000001	1	1
00000010	2	2
01111110	126	126
01111111	127	127
10000000	128	−128
10000001	129	−127
10000010	130	−126
11111110	254	−2
11111111	255	−1
	0~~2^n-1	-2^{n-1} ~~ 2^{n-1}-1
	2^n=256 values	2^n=256 values

Unsigned potentially save bits

E.g., Count # student in our class (about 150)

- If declared `signed short`, max 127, 8 bits not enough
- If declared `unsigned`, then 8 bits are enough.

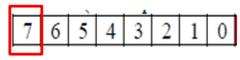
```
unsigned short counter;
```

32

32

Qualifiers (modifiers) for integer types -- finally

- If a qualifier, including **`long`**, **`short`**, is applied then **`int`** can be omitted




```
  ▪  signed   char
  ▪ (signed)  int
  ▪ (signed)  short  (int)   ⟺  short
  ▪ (signed)  long   (int)   ⟺  long

  ▪ unsigned char
  ▪ unsigned (int)           ⟺  unsigned
  ▪ unsigned short  (int)    ⟺  unsigned short
  ▪ unsigned long   (int)    ⟺  unsigned long
```

scanf ("%hd") for short int, ("%ld") for long int, ("%lld") for long long (C99)
printf ("%hd) for short int, ("%ld") for long int, ("%lld") for long long (C99)

33 For your information

YORK UNIVERSITÉ UNIVERSITY

33

Qualifiers for floating points

- "**`long`**" can be used with double:
  - **`long double`**
- Thus, there are three types of floating points:
  - **`float`** `/*single-precision floating point*/`
  - **`double`** `/*double precision floating point*/`
  - **`long double`** `/*extended-precision floating point*/`
- More bits, more precise.
  - 3.1415926535….

- scanf ("%f") for float, ("%lf") for double, ("%Lf") for long double
- printf ("%f") for float, double, ("%Lf") for long double

34

34

Summary

Java defines

Type
int
short
long
byte
float
double
char
boolean

- Integer types:
  - `char`
    `signed char` `unsigned char`
  - `(signed) short` `unsigned short`
  - `(signed) int` `unsigned int`
  - `(signed) long` `unsigned long`

- There are three types of floating points:
  - `float /* single-precision */`
  - `double /* double precision */`
  - `long double /* extended-precision */`

- C99 added:
  - `(signed) long long int`
  - `unsigned long long int`

YORK UNIVERSITÉ UNIVERSITY

35

35

Size of Types

Java defines eight primitive type

Type	
int	A 32-bit (4-byte)
short	A 16-bit (2-byte)
long	A 64-bit (8-byte)
byte	An 8-bit (1-byte)
float	A 32-bit (4-byte)
double	A 64-bit (8-byte)
char	A 16-bit characte
boolean	A true or false

- Exact sizes of types depend on machine

char = 8 bits [for sure] 1 byte
short ≥ 16 bits [usually 16 bits] 2 bytes
int ≥ 16 bits [usually 32 bits] 4 bytes
long ≥ 32 bits [usually 32 or 64 bits] 4 or 8 l
float ≥ 32 bits [usually 32 bits] 4 bytes
double ≥ 64 bits [usually 64 bits] 8 bytes

- Relations of sizes:
  - **short ≤ int ≤ long**
  - **float ≤ double ≤ long double**

YORK UNIVERSITÉ UNIVERSITY

36

36

Size of Types

- To get exact size of a type in a machine, use **sizeof** operator
  - **sizeof (int)** or **int a; sizeof a**; or **sizeof (a)**

  -- Memory allocation in **byte**

```
int main(int argc, char *argv[])
{
        printf("size of char %d\n", sizeof(char));
        printf("size of unsigned char %d\n", sizeof(unsigned char));
        printf("size of signed char %d\n\n", sizeof(signed char));

        printf("size of short int %d\n", sizeof(short int));
        printf("size of unsigned short int %d\n\n", sizeof(unsigned short int));

        printf("size of int %d\n", sizeof(int));
        printf("size of unsigned int %d\n\n", sizeof(unsigned int));

        printf("size of long int %d\n", sizeof(long int));
        printf("size of unsigned long int %d\n\n", sizeof(unsigned long int));

        printf("size of float %d\n", sizeof(float));
        printf("size of double %d\n", sizeof(double));
        printf("size of long double %d\n\n", sizeof(long double));

        printf("size of long long int %d\n", sizeof(long long)); /* new in c99 */
        printf("size of unsigned long long int %d\n", sizeof(unsigned long long));
```

37

Size of Types

- To get exact size of a type in a machine, use **sizeof** operator
  - **sizeof (int)** or **int a; sizeof a ; or sizeof (a)**
  - Let us see our lab…..

```
indigo 270 % gcc size-2017.c
indigo 271 % a.out
sizes in byte

size of char: 1
size of unsigned char: 1
size of signed char: 1

size of short int: 2
size of unsigned short int: 2

size of int: 4
size of unsigned int: 4

size of long int: 8
size of unsigned long int: 8

size of float: 4
size of double: 8
size of long double: 16

size of long long int: 8        // c99
size of unsigned long long int: 8    // c99
```

Different on different machines (except char)

```
short ≤ int ≤ long
float ≤ double ≤ long double
```

YORK UNIVERSITÉ UNIVERSITY

38

So How Big Is It?

- Might need to know the min/max of types,
  - avoid over flow. **`int x = 34589643`**?
  - signed: $-2^{n-1} \sim 2^{n-1}-1$ → $-2^{\text{sizeof(x)*8-1}} \sim 2^{\text{sizeof(x)*8-1}}-1$
  - unsigned: $0 \sim 2^{n}-1$ → $0 \sim 2^{\text{sizeof(x)*8}}-1$
- **`<limits.h>`** provides constants:
  - **`char`** **`CHAR_MIN, CHAR_MAX`** … **`0~256 -127~127`**
  - **`int`** **`INT_MIN, INT_MAX….`**
  - **`long`** **`LONG_MIN, LONG_MAX`**
  - **`short`** **`SHRT_MIN, SHRT_MAX`**
- **`<float.h>`** provides min/max for floating points.
- See appendix B11 of the textbook

39

For your information

YORK U UNIVERSITÉ UNIVERSITY

39

So How Big Is It?

- **`<limits.h>`**provides constants:
  - **`char`** **`CHAR_MIN, CHAR_MAX`** … **`0~256 -127~127`**
  - **`int`** **`INT_MIN, INT_MAX….`**
  - **`long`** **`LONG_MIN, LONG_MAX`**
  - **`short`** **`SHRT_MIN, SHRT_MAX`**

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

int main() {

   printf("The minimum/maximum value of SIGNED CHAR:  %d ~ %d\n", SCHAR_MIN, SCHAR_MAX);
   printf("The minimum/maximum value of UNSIGNED CHAR:  %d ~ %d\n\n", 0, UCHAR_MAX);

   printf("The minimum/maximum value of SIGNED SHORT INT:  %d ~ %d\n", SHRT_MIN, SHRT_MAX);
   printf("The minimum/maximum value of UNSIGNED SHORT INT:  %d ~ %d\n\n", 0, USHRT_MAX);


   printf("The minimum/maximum value of INT:  %d ~ %d\n", INT_MIN, INT_MAX);
   printf("The minimum/maximum value of UNSIGNED INT:  %d ~ %u\n\n",0, UINT_MAX);

   printf("The minimum/maximum value of LONG:  %ld ~ %ld \n", LONG_MIN, LONG_MAX);
   printf("The minimum/maximum value of UNSIGNED LONG:  %d ~ %lu\n", 0,ULONG_MAX);

   return(0);
}
```

For your information

40

So How Big Is It?

- `<limits.h>`provides constants:
  - `char` `CHAR_MIN, CHAR_MAX` … `0~256 -127~127`
  - `int` `INT_MIN, INT_MAX`….
  - `long` `LONG_MIN, LONG_MAX`
  - `short` `SHRT_MIN, SHRT_MAX`

```
indigo 273 % a.out
The minimum/maximum value of SIGNED CHAR:  -128 ~ 127
The minimum/maximum value of UNSIGNED CHAR:  0 ~ 255

The minimum/maximum value of SIGNED SHORT INT:  -32768 ~ 32767
The minimum/maximum value of UNSIGNED SHORT INT:  0 ~ 65535      // 0 ~ 2^16-1 -1

The minimum/maximum value of INT:  -2147483648 ~ 2147483647     // -2^32-1 -1 ~ 2^32-1 -1
The minimum/maximum value of UNSIGNED INT:  0 ~ 4294967295      // 0 ~ 2^32 -1

The minimum/maximum value of LONG:  -9223372036854775808 ~ 9223372036854775807
The minimum/maximum value of UNSIGNED LONG:  0 ~ 18446744073709551615
```

- `<float.h>` provides min/max for floating points.
- See appendix B11 of the textbook

41

For your information

41

Outline

- Types and sizes
  - Types
  - **Constant values (literals)**
    - **char**
    - int
    - float
- Array and “strings”
- Expressions
  - Basic operators
  - Type promotion and conversion
  - Other operators
  - Precedence of operators

42

42

Character Constants

- A **`char`** in C is one byte (8-bit) in size (Java 16-bit)
- A constant char is specified with single quotes:
  - Regular characters: `'A', 'C', 'z', '0', '#', '$',…`
    - `char x = 'A';`
  - Special characters: invisible or control chars
    - New line, tab ….
    - Use escape sequence to represent

Same in Java

43

43

Special Characters

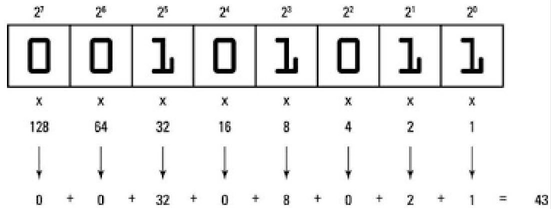
Escape sequence	Meaning
\n	New line
\t	Tab
\0	The null character
\\	The \ character
\"	Double quote
\'	Single quote

```
char c = '\t';
char c2 = '\n'
```

Same in Java

44

Internal representation of characters

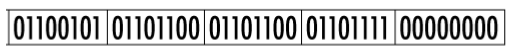



```
int i =  43;

char a = 'A';
```

How to represent 'A' using 0s and 1s

45

45



Internal Representation of characters

- characters as 1/0 bits. So they are stored as (small) <u>integer</u> values, interpreted according to the character set encoding (usually ASCII, 7 bits for 128 characters),
  - `'a'` has encoding `97`, `'0'` has `48`, `'9'` has `57`
- Escape sequences are integers too
  - e.g. `'\n'` has `10` (newline character)
    `'\t'` has `9` (horizontal tab)
- Special escape: `'\0'`
  has encoding 0 - the null character

46

46

01100101 01101100 01101100 01101111 00000000

Internal Representation of Characters

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN	(synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB	(end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN	(cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM	(end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB	(substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS	(group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

47

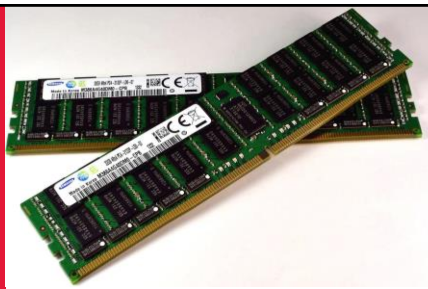
01100101 01101100 01101100 01101111 00000000

Internal Representation of Characters

- '0' - '9' are encoded consecutively (48~57)
- 'A' - 'Z' are encoded consecutively (65~90)
- 'a' - 'z' are encoded consecutively (97~122)
- Upper letters before lower. Index/encoding difference of 'a' and 'A' is 32, so does 'b' and 'B', 'c' and 'C', …
- 8 bits is enough
- Java uses a bigger character set table Unicode, 0~127 are same

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
					33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
					34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
					35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
					36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
					37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
					38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
					39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
					40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
					41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
					42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
					43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
					44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
					45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
					46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
					47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
					48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
					49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
					50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
					51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
					52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
					53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
					54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
					55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
					56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
					57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
					58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
					59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
					60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
					61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
					62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
					63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

48



65	41	101	A	A	97	61	141	a	a
66	42	102	B	B	98	62	142	b	b
67	43	103	C	C	99	63	143	c	c
68	44	104	D	D	100	64	144	d	d
69	45	105	E	E	101	65	145	e	e
70	46	106	F	F	102	66	146	f	f
71	47	107	G	G	103	67	147	g	g
72	48	110	H	H	104	68	150	h	h
73	49	111	I	I	105	69	151	i	i
74	4A	112	J	J	106	6A	152	j	j
75	4B	113	K	K	107	6B	153	k	k
76	4C	114	L	L	108	6C	154	l	l
77	4D	115	M	M	109	6D	155	m	m
78	4E	116	N	N	110	6E	156	n	n
79	4F	117	O	O	111	6F	157	o	o
80	50	120	P	P	112	70	160	p	p
81	51	121	Q	Q	113	71	161	q	q
82	52	122	R	R	114	72	162	r	r
83	53	123	S	S	115	73	163	s	s
84	54	124	T	T	116	74	164	t	t
85	55	125	U	U	117	75	165	u	u
86	56	126	V	V	118	76	166	v	v
87	57	127	W	W	119	77	167	w	w
88	58	130	X	X	120	78	170	x	x
89	59	131	Y	Y	121	79	171	y	y
90	5A	132	Z	Z	122	7A	172	z	z

'H'	'e'
72	101
01001000	01100101

'H'	'e'	'l'	'l'	'o'
72	101	108	108	111
01001000	01100101	01101100	01101100	01101111

YORK UNIVERSITÉ UNIVERSITY

49

49

Characters

46	2E	056	.	.
47	2F	057	/	/
48	30	060	0	0
49	31	061	1	1
50	32	062	2	2
51	33	063	3	3
52	34	064	4	4
53	35	065	5	5
54	36	066	6	6
55	37	067	7	7
56	38	070	8	8
57	39	071	9	9
58	3A	072	:	:
59	3B	073	;	;
60	3C	074	<	<
61	3D	075	=	=
62	3E	076	>	>
63	3F	077	?	?

- **chars** are treated in C as <u>small integers</u>, **char** variables and constants are identical to **int** in arithmetic expressions:
  - **char c** is converted to its encoding (index in the character set table)

```
char aChar = '5';    // encoding 53
aChar + 12         // expression with value 53+12 = 65
```

same in Java

- Same for other expressions. In relational expression, characters can be compared directly, comparing indexes/encodings

```
aChar == EOF       // 53 == -1?  ➔ expr with value 0 (false)

aChar == 'H'      // index == 72?  ➔ expr with value 0 (false)

aChar == '/n'   // index = 10?  ➔ exp with value 0 (false)

'5' < 'H'   // 53 < 72?  Earlier in table? ➔ expr with 1 (true)
```

50

50

char is (represented as) small integers (≤256)

47	2F	057	/	/
48	30	060	0	0
49	31	061	1	1
50	32	062	2	2
51	33	063	3	3
52	34	064	4	4
53	35	065	5	5
54	36	066	6	6
55	37	067	7	7
56	38	070	8	8
57	39	071	9	9
58	3A	072	:	:
59	3B	073	;	;

```
class CharTest
{
    public static void main(String[] args)
  {
        System.out.println("Hello World!")

        char aCh = '3'; // encoding 51
        System.out.println(aCh)   // 3
        System.out.println(aCh+0); // 51

        System.out.println(aCh+4); // 55
        System.out.println(aCh - '0'); // 51-48=3 !
        System.out.println(aCh - '0'+4); // 7

        System.out.println(aCh > 40);   // true
    }
}
```

51

51

Characters

46	2E	056	.	.
47	2F	057	/	/
48	30	060	0	0
49	31	061	1	1
50	32	062	2	2
51	33	063	3	3
52	34	064	4	4
53	35	065	5	5
54	36	066	6	6
55	37	067	7	7
56	38	070	8	8
57	39	071	9	9
58	3A	072	:	:
59	3B	073	;	;
60	3C	074	<	<
61	3D	075	=	=
62	3E	076	>	>

- Since **chars** are just small integers, **char** variables and constants are identical to **int** in arithmetic expressions:
  - **char c** is converted to its encoding (index in the character set table)

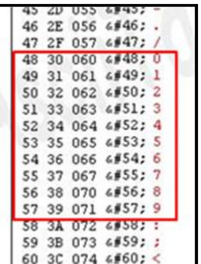
```
char aCh = '6';  // same as   char aCh = 54;
printf("value is %c\n", aCh ); // char 6
printf("value is %d\n", aCh ); // numerical 54
                               // print encoding

printf("value is %c\n", aCh + 2 ); // char 8
printf("value is %d\n", aCh + 2 ); //numerical 56

printf("value is %d\n", aCh-'0' ); // numerical 6
```

52

same in Java

52

Characters



45	2D	055	-	-
46	2E	056	.	.
47	2F	057	/	/
48	30	060	0	0
49	31	061	1	1
50	32	062	2	2
51	33	063	3	3
52	34	064	4	4
53	35	065	5	5
54	36	066	6	6
55	37	067	7	7
56	38	070	8	8
57	39	071	9	9
58	3A	072	:	:
59	3B	073	;	;
60	3C	074	<	<

- Since **chars** are just small integers, **char** variables and constants are identical to **int**s in arithmetic expressions: take advantage of this

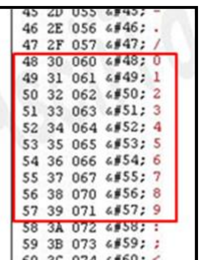
```
if(c >= '0' && c <= '9') /*index 48~57,is a digit */
                 (located after '0' and before '9')

if(c >='a' && c <= 'z') /* low case letter*/

if(c >='A' && c <= 'Z') /*upper case letter*/


if(c >='0' && c <= '9'){  // c<= 48 c>=57 isdigit(c)
  printf("c is a digit\n");
  printf("numerical value is %d\n",        );
}
```

same in Java

53

53

Characters



45	2D	055	-	-
46	2E	056	.	.
47	2F	057	/	/
48	30	060	0	0
49	31	061	1	1
50	32	062	2	2
51	33	063	3	3
52	34	064	4	4
53	35	065	5	5
54	36	066	6	6
55	37	067	7	7
56	38	070	8	8
57	39	071	9	9
58	3A	072	:	:
59	3B	073	;	;
60	3C	074	<	<

- Since **chars** are just small integers, **char** variables and constants are identical to **int**s in arithmetic expressions: take advantage of this

```
if(c >= '0' && c <= '9') /*index 48~57,is a digit */
                 (located after '0' and before '9')

if(c >='a' && c <= 'z') /* low case letter*/

if(c >='A' && c <= 'Z') /*upper case letter*/


if(c >='0' && c <= '9'){  // c<= 48 c>=57 isdigit(c)
  printf("c is a digit\n");
  printf("numerical value is %d\n", c-'0');
}
```

same in Java

54

54

Example

- Upper case letters before lower case letters.
- Encoding difference of 'a' and 'A' is 32, so does 'b' and 'B', 'c' and 'C', 'd' and 'D'…

64	40	100	@	@	96	60	140	`	`
65	41	101	A	A	97	61	141	a	a
66	42	102	B	B	98	62	142	b	b
67	43	103	C	C	99	63	143	c	c
68	44	104	D	D	100	64	144	d	d
69	45	105	E	E	101	65	145	e	e
70	46	106	F	F	102	66	146	f	f
71	47	107	G	G	103	67	147	g	g
72	48	110	H	H	104	68	150	h	h
73	49	111	I	I	105	69	151	i	i
74	4A	112	J	J	106	6A	152	j	j
75	4B	113	K	K	107	6B	153	k	k
76	4C	114	L	L	108	6C	154	l	l
77	4D	115	M	M	109	6D	155	m	m
78	4E	116	N	N	110	6E	156	n	n
79	4F	117	O	O	111	6F	157	o	o
80	50	120	P	P	112	70	160	p	p
81	51	121	Q	Q	113	71	161	q	q
82	52	122	R	R	114	72	162	r	r
83	53	123	S	S	115	73	163	s	s
84	54	124	T	T	116	74	164	t	t
85	55	125	U	U	117	75	165	u	u
86	56	126	V	V	118	76	166	v	v
87	57	127	W	W	119	77	167	w	w
88	58	130	X	X	120	78	170	x	x
89	59	131	Y	Y	121	79	171	y	y
90	5A	132	Z	Z	122	7A	172	z	z
91	5B	133	[	[	123	7B	173	{	{
92	5C	134	\	\	124	7C	174			|

```
#include<stdio.h>

/*copying input to output with
converting upper-case to lower-case letters */
main(){
    int c; int lowC;
    c= getchar();
    while (c != EOF)
    {
       if (c >= 'A' && c <= 'Z') /* 65~90 upper case letter*/
          lowC = c + 'a'- 'A';   /* c + 'b' - 'B'  */
                                 /* c + 'c' - 'C'  */
       putchar(lowC);            /* c = tolower(c)  */

       c = getchar(); // read again
    }
    return 0;
}
```

`c + 32` works but not good for portability. Avoid that!

55

55

```
main(){
 char le = 'J';  // 74
 while (le <= 'Q') {
   printf ("%d %c %cack %c\n", le,le,le, le+1);
   le++;
 }
}
```

64	40	100	@	@
65	41	101	A	A
66	42	102	B	B
67	43	103	C	C
68	44	104	D	D
69	45	105	E	E
70	46	106	F	F
71	47	107	G	G
72	48	110	H	H
73	49	111	I	I
74	4A	112	J	J
75	4B	113	K	K
76	4C	114	L	L
77	4D	115	M	M
78	4E	116	N	N
79	4F	117	O	O
80	50	120	P	P
81	51	121	Q	Q
82	52	122	R	R
83	53	123	S	S
84	54	124	T	T
85	55	125	U	U
86	56	126	V	V
87	57	127	W	W
88	58	130	X	X
89	59	131	Y	Y
90	5A	132	Z	Z
91	5B	133	[	[
92	5C	134	\	\
93	5D	135	]	]
94	5E	136	^	^
95	5F	137	_	_

```
74   J    Jack   K
75   K    Kack   L
76   L    Lack   M
77   M    Mack   N
78   N    Nack   O
79   O    Oack   P
80   P    Pack   Q
81   Q    Qack   R
```

same in Java

56

56

Outline

- Types and sizes
  - Types
  - **Constant values (literals)**
    - **char** treated as small int
    - **int** different bases
    - float
- Array and “strings”
- Expressions
  - Basic operators
  - Type promotion and conversion
  - Other operators
  - Precedence of operators

57

57

0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

Stored always binary 2^2 2^1 2^0

Integer Constants

- Integer constants can be expressed in <u>three</u> different ways:

1. **Decimal** [base 10]
   - `int x = 31`

   same in Java

2. **Octal** [base 8]
   - Start with zero **0**
   - `int x = 037` (31 in decimal)

   same in Java

3. **Hexadecimal** [base 16]
   - Start with **0x** or **0X**
   - `int x = 0x1F` (31 in decimal)

   same in Java

Ways for people to write numbers. No effect on how the numbers are stored –- always binary.

Java also has the 4th way: binary
`int x = 0b00011111`

58

58

Octal Notation

- base 8 ... uses 8 symbols

  0, 1, 2, 3, 4, 5, 6, 7
- Position represents power of 8
- 1523_8

  $(1 * 8^3) + (5 * 8^2) + (2 * 8^1) + (3 * 8^0) = 851$

Hexadecimal Notation

- base 16 or 'hex' ... uses 16 symbols

  0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
- Position represents powers of 16
- $B65F_{16}$ or 0xB65F

  $(11 * 16^3) + (6 * 16^2) + (5 * 16^1) + (15 * 16^0) = 46687$

59

Others To decimal

- 3 2 8

  10^2 10^1 10^0

  ➡ $3*10^2 + 2*10^1 + 8*10^0 =$

  $300 + 20 + 8 = 328$

  Decimal 328

- 1 0 1

  2^2 2^1 2^0

  ➡ $1*2^2 + 0*2^1 + 1*2^0 =$

  $4 + 0 + 1 = 5$

  00 00 00 00 00101

  Binary

- 3 4 5

  8^2 8^1 8^0

  ➡ $3*8^2 + 4*8^1 + 5*8^0 =$

  $192 + 32 + 5 = 229$

  Octal 0345

- 3 4 F

  16^2 16^1 16^0

  ➡ $3*16^2 + 4*16^1 + F*16^0 =$

  $3*256 + 4*16 + 15 *1 =$

  $768 + 64 + 15 = 847$

  Hex 0x34F

  0X34f

60 You should know these conversions.

60

Binary to others -- why Hex and Oct

I know I want an int with representation 01001100, how to code it in C?

Java, can do binary `int a = 0b01001100`

- 0 1 0 0 1 1 0 0 ➡ $1*2^6 + 1*2^3 + 1*2^2 =$ 64 + 8 + 4 — Decimal `int a = 76`

  $2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

- $\underbrace{0\,1}_{1}\ \underbrace{0\,0\,1}_{1}\ \underbrace{1\,0\,0}_{4}$ ➡ `int a = 0114` Octal

- $\underbrace{0\,1\,0\,0}_{4}\ \underbrace{1\,1\,0\,0}_{C}$ ➡ `int a = 0X4C` `= 0x4c` Hex

61 You should know these conversions.

YORK UNIVERSITÉ UNIVERSITY

61

Decimal number	Binary representation	Octal representation	Hexadecimal representation
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

int a=16	int a=0b10000	int a=020	int a=0X10
int a=76	int a=0b1001100	int a=0114	int a=0x4C

Java only

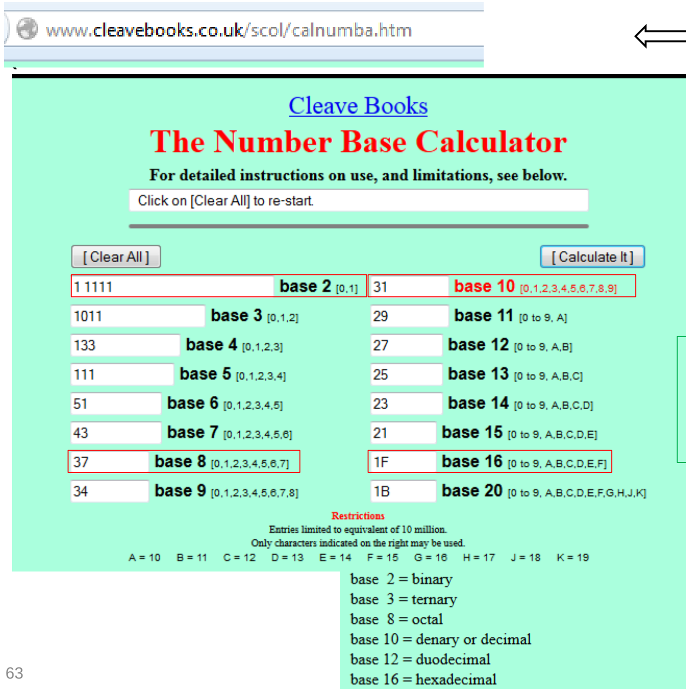
Dec	Hx	Oct	Char
0	0	000	
1	1	001	
2	2	002	
3	3	003	
4	4	004	
5	5	005	
6	6	006	
7	7	007	
8	8	010	
9	9	011	
10	A	012	
11	B	013	
12	C	014	
13	D	015	
14	E	016	
15	F	017	
16	10	020	
17	11	021	
18	12	022	
19	13	023	
20	14	024	
21	15	025	
22	16	026	
23	17	027	
24	18	030	
25	19	031	
26	1A	032	
27	1B	033	ESC (escape)
28	1C	034	FS (file se
29	1D	035	GS (group s
30	1E	036	RS (record
31	1F	037	US (unit se

64	40	100	@	@
65	41	101	A	A
66	42	102	B	B
67	43	103	C	C
68	44	104	D	D
69	45	105	E	E
70	46	106	F	F
71	47	107	G	G
72	48	110	H	H
73	49	111	I	I
74	4A	112	J	J
75	4B	113	K	K
76	4C	114	L	L
77	4D	115	M	M
78	4E	116	N	N
79	4F	117	O	O
80	50	120	P	P
81	51	121	Q	Q
82	52	122	R	R
83	53	123	S	S
84	54	124	T	T
85	55	125	U	U
86	56	126	V	V
87	57	127	W	W
88	58	130	X	X
89	59	131	Y	Y
90	5A	132	Z	Z
91	5B	133	[	[
92	5C	134	\	\
93	5D	135	]	]
94	5E	136	^	^
95	5F	137	_	_

62

YORK UNIVERSITÉ UNIVERSITY

62



Try it!

Also a writeup "Number system.pdf" on the course website

YORK UNIVERSITÉ UNIVERSITY

63

63

Integer Constants (finally)

- We can specify type qualifier at the end:
  - 'u' or 'U' ⇨ unsigned (int)
  - 'l' or 'L' ⇨ long (int)
  - nothing ⇨ just int

same in Java

- E.g.

5	as an	**"(signed) (decimal) int" 5**
5U	as an	**"unsigned (decimal) int" 5**
5L	as a	**"(signed) long (int)" 5**
5UL or 5ul	as an	**"unsigned long (int)" 5**
037	as an	**"(signed) int (oct)" decimal: 31**
0x32dUL	as an	**"unsigned long (int) in hex"**
059	as an	? 0ctal decimal 59
0x39G2	as an	? hex

YORK UNIVERSITÉ UNIVERSITY

64

64

Outline

- Types and sizes
  - Types
  - Constant values (literals)
    - char
    - int
    - **float**
- Array and "strings"
- Expressions
  - Basic operators
  - Type promotion and conversion
  - Other operators
  - Precedence of operators

65

65

Floating Point Constants

- All floating point constants contain a decimal point('.') and/or an exponent ('e' of "E")
  - E.g. 1.532 3e5 234.112e-10
  - 5.3e12==5.3 × 10^{12}
- Floating point constants are of type 'double'
  - Nothing – means **"double"** e.g., `double x = 1.532`
  - 'f' or 'F' - means **"float"** e.g. `float x = 1.532F`
    `float x = 1.532` ok
  - 'l' or 'L' - means **"long double"** e.g. `long double x=1.5L`

same in Java

66

66

Outline

- Types and sizes
  - Types
  - Constant values (literals)
    - char
    - int
    - float
- **Array and “strings” (Ch1.6,1.9)**
- Expressions
  - Basic operators
  - Type promotion and conversion
  - Other operators
  - Precedence of operators

67

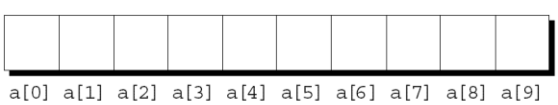
67

Arrays

- Indexed list of objects of the **same** type
  - **`int a[10];`** -- declare an array of 10 int’s
  - **`float x[20];`** -- declare an array of 20 float’s

  type name size

- Index numbering starts from 0 (!)
  - **`a[0]`** … **`a[9]`**
  - **`x[0]`** … **`x[19]`** ← array elements



68

same in Java

68

Declaring Arrays

- Declare and initialize (how to do in Java?)

```
int k[5];        /* each element get some garble value*/
int k[5]; k = {1,5,4,2,25};  /* invalid  as in Java */
int k[5] = {1,5,3,2,25}; /* valid 1 5 3 2 25 as Java */

int k[5] = {1};      /* valid. 1 0 0 0 0 (rest is 0) */
int k[4] = {1,4};  /* valid 1 4 0 0 (rest is 0) */
```

Interview questions

```
int k[3] ={1,4,2,1}       /* invalid */
int k[] ={1,4,2,1};  /*valid  1 4 2 1 valid in Java too */
int k[];      /*invalid "size missing" */
```

69

69

Accessing Arrays

- In C, you can only assign to array members
  - This means you cannot assign to an array:

```
int  i, k[4], j[4];
for (i=0; i<4; i++)
   j[i]= 0;     /* another way? int j[4]={0}  */

k = j;  /* invalid *//* perfectly valid in Java */
```

j → | 0 | 0 | 0 | 0 |
k

```
for (i=0; i<4; i++)
   k[i] = j[i];
```

```
i=0;
while(i<4)
{
    k[i] = j[i];
    i++;
}
```

70

70

An example involving array and chars

What does this program do?

```
/*counting digits*/

#include <stdio.h>
#define N 10
int main () {
    int c, i;
    int digit[N];

    for (i=0; i< N; i++)
      digit[i]=0;

    while ((c = getchar()) != EOF)
      if ( c>= '0' && c <= '9' )
        digit[c - '0'] ++;     // digit[c] ++ ?

    for (i=0; i< N; i++)
       printf ("%d ", digit[i]);

    return 0;
}
```

45	2D	055	-	-
46	2E	056	.	.
47	2F	057	/	/
48	30	060	0	0
49	31	061	1	1
50	32	062	2	2
51	33	063	3	3
52	34	064	4	4
53	35	065	5	5
54	36	066	6	6
55	37	067	7	7
56	38	070	8	8
57	39	071	9	9
58	3A	072	:	:
59	3B	073	;	;
60	3C	074	<	<

Strings ⟷ Character Arrays !

Dec	Hx	Oct	Char	
0	0	000	NUL	(null)
1	1	001	SOH	(start of heading)
2	2	002	STX	(start of text)

- There is no separate “string” type in C
- Strings are just arrays of char that end with `'\0'`
  - `char s[]= "Hello";`

⇩

'H'	'e'	'l'	'l'	'o'	'\0'

\0 added for you

‘H’	‘e’	‘l’	‘l’	‘o’	‘\0’
72	101	108	108	111	0
01001000	01100101	01101100	01101100	01101111	00000000

is equivalent to

```
char s[]= {'H', 'e', 'l', 'l', 'o', '\0'}
```

72

Strings ⟺ Character Arrays !

- There is no separate "string" type in C

Dec	Hx	Oct	Char
0	0	000	NUL (null)
1	1	001	SOH (start of heading)
2	2	002	STX (start of text)

- Strings are just arrays of char that end with `'\0'`
  - `char s[]= "Hello";`

⇩

'H'	'e'	'l'	'l'	'o'	'\0'
01001000	01100101	01101100	01101100	01101111	00000000

\0 added for you

- What's the size of s in memory? 6×1 bytes (chars)! **sizeof s? 6**
  - `char s[5]= "Hello";` ✗
  - `char s[8] = "Hello";` 8×1 bytes

'H'	'e'	'l'	'l'	'o'	'\0'	'\0'	'\0'

sizeof s? 8

- What is the length of s?
  `strlen(s) = 5` later

YORK UNIVERSITÉ UNIVERSITY

73

An example involving char arrays

```
#include<stdio.h>

main() {
   char s1[]= "Hello";
   char s2[8];
   printf("s1: %s\n",s1); // s1: hello

   int i=0;
   while (s1[i] != '\0'){
      s2[i] = s1[i];
      i++;
   }
   s2[i]='\0'; /*finally add \0 manually*/

   printf("s2: %s\n",s2); // s2: Hello
   return 0;              // printf stops at first \0
}
```

H	e	l	l	o	\0

sizeof s1: 6 strlen(s1): 5

H	e	l	l	o	\0		

sizeof s2: 8 strlen(s2): 5

74

An example involving char arrays

```
#include<stdio.h>
void stringcopy(char dest [], char src [])
{
    int i=0;
    while (src[i] != '\0'){
        dest[i] = src[i];
        i++;
    }
    dest [i]='\0'; /*finally add \0 manually*/
}
main() {
    char s1[]= "Hello!";
    char s2[8];
    stringcopy(s2, s1);
    printf("s2 is %s\n",s2);

    return 0;
}
```

Passing array in C is a big topic, investigate later

H	e	l	l	o	\0

sizeof s1: 6 strlen(s1): 5

H	e	l	l	o	\0		

sizeof s2: 8 strlen(s2): 5

75

75

An example involving char arrays

```
#include<stdio.h>
void stringcopy2(char dest [], char src [])
{
    int i=0;
    while (1){                              /* Another version */
        dest[i] = src[i];
        if (src[i] == '\0')         // if (dest[i] == '\0')
            break;

        i++;
    }
}

main() {
    char s1[]= "Hello!";
    char s2[8];
    stringcopy2(s2, s1);
    printf("s2 is %s\n",s2);

    return 0;
}
```

76

76

An example involving reading char arrays

```
#include<stdio.h>
int length (char []);

main() {
   char my_strg[100];
   int a;

   printf("Enter a word and an int by blank>");
   scanf("%s %d", my_strg, &a);
   printf("%d", length(s));
}

int length(char arr[]){
    int i = 0;
    while (arr[i] != '\0')
      i++;
    return i;
}
```

No & needed!
Another big topic.
Investigate later

Outline

- Types and sizes
  - Types
  - Constant values (literals)
    - char
    - int
    - float
- Array and “strings” (Ch1.6,1.9)
- **Expressions**
  - **Basic operators (arithmetic, relational and logical)**
  - Type promotion and conversion
  - Other operators (bitwise, bit shifting , compound assignment, conditional)
  - Precedence of operators

Expressions

- Expressions are made up of *operands* (things we operate upon) and *operators* (things that do the operations: `+ - * % > <`)
  - `x+y/2, i>=0, x==y, i++,…`
- Operands can be constants, variables, array elements, function calls and other expressions
- Every expression has a return value.
  - `x+2` has return value 3 if `x` was 1
  - `i < 20` has return value true or false -- 1 or 0
- In C/Java, `=` is a operator, so assignment is also an expression
  - **`variable = expression`**
  - `x = 2+3` has return value 5 `printf("%d", x=2+3); // 5`
  - Assignment expression can be an operand in other expressions
    - ○ `y = x = 2;`
    - ○ `while ((c=getchar())!= EOF )`

"whenever a value is needed, any expression of the same type will do"

79

```
printf("sum is %d\n", i*y+2);
```

79

Expressions

- Some of the common operators:
  - `+, -, *, /, %, ++,--` (basic arithmetic)
  - `<, >, <=, >=` (relational operators)
  - `==, !=` (equality operators)
  - `&&, ||, !` (logical operators)
  - `= += -=` (assignment & compound assignment)
- Others: bitwise `& | ~`, bit shifting `<< >>`, conditional `? :`

  `sizeof`

80

YORK U UNIVERSITÉ UNIVERSITY

80

Arithmetic (unary) Increment/Decrement Operators

- **++** increment
- **--** decrement

same in Java

- May come before (prefix) or after the operand (postfix)

`++x`	increment x, result of expression is new value (pre-increment)
`x++`	increment x, result of expression is old value (post-increment)
`--x`	decrement x, result of expression is new value (pre-decrement)
`x--`	decrement x, result of expression is old value (post-decrement)

```
while (x < 10){
  ......
  x++;  // increment later,
        before next statement
  ......
}
```

```
while (x < 10){
  ......
  ++x;  // increment immediately
  ......
}
```

Same effects

81

81

Arithmetic (unary) Increment/Decrement Operators

- **++** increment
- **--** decrement

same in Java

- May come before (prefix) or after the operand (postfix)

`++x`	increment x, result of expression is new value (pre-increment)
`x++`	increment x, result of expression is old value (post-increment)
`--x`	decrement x, result of expression is new value (pre-decrement)
`x--`	decrement x, result of expression is old value (post-decrement)

```
x = 2;
y = x++;  // increment after
             assignment
printf("%d %d",x, y);
```

x:3 y:2

```
x = 2;
y = ++x;  // increment before
             assignment
printf("%d %d",x, y);
```

x: 3 y:3

82

82

Arithmetic (unary) Increment/Decrement Operators

- ++ increment
- -- decrement

same in Java

- May come before (prefix) or after the operand (postfix)

`++x`	increment x, result of expression is new value (pre-increment)
`x++`	increment x, result of expression is old value (post-increment)
`--x`	decrement x, result of expression is new value (pre-decrement)
`x--`	decrement x, result of expression is old value (post-decrement)

```
x = 2;
y = x--;  // decrement after assignment
printf("%d %d",x, y);
```

```
x = 2;
y = --x;  // decrement before assignment
printf("%d %d",x, y);
```

83

x:1 y:2

x: 1 y:1

83

Arithmetic (unary) Increment/Decrement Operators

- The prefix/postfix effect can be subtle

```
int x = 3, y, z;
y= x++;  // post-increment. y=x; x=x+1;
z= ++x; //  pre-increment.  x=x+1; z=x;
printf("x:%d y:%d z:%d",x, y,z);
```

same in Java

- What are the output?

x:5 y:3 z:5

84

84

Arithmetic (unary) Increment/Decrement Operators

- The prefix/postfix effect can be subtle

same in Java

```
int x = 3, y, z;
y= x++;  // post-increment. y=x; x=x+1;
z= ++x; //  pre-increment.  x=x+1; z=x;
printf("x:%d y:%d z:%d",x, y++, --z);
```

```
z = z-1;
printf("x:%d y:%d z:%d",x, y, z);
y = y+1;
```

- What are the output?

  x:5 y:3 z:4

85

85

A common use

```
/*initialize to 0 */

#include <stdio.h>
#define N 10

int main () {

   int i=0;
   int digit[N];

   while (i< N)
   {
     digit[i]=0;
     i++;
   }
```

```
// succinct code

while ( i< N)
{
  digit[i++]=0;
          //post-increment
}
```

86

86

A common use

```
/*copy 4 elements from pos 10 of arr1 to arr2 */

#include <stdio.h>
#define N 10
int main () {
   int i,j;
   …..

   i=0; j=10;                        // succinct code
   while (i<4 && j<14…)               while(i<4 && j<14…)
   {                                  {
     arr2[i] = arr1[j];      <->        arr2[i++] = arr1[j++];
     i++;                                 //post-increments
     j++;
   }                                  }
```

87

87

Summary and future work

- Today (ch2):
  - Types and sizes
    - Basic types, their size and constant values (literals)
      - ✓ char: x > ‘a’ && x < ‘z’; x > ‘0’ && x < ’9’
      - ✓ int: 122, 0122, 0x12F convert between decimal, bin, oct, hex
    - Arrays (one dimension) and strings (Ch1.6,1.9)
      - ✓ “hello” has size 6, | H | e | l | l | o | \0 |
  - Expressions
    - Basic operators (arithmetic, relational and logical)
      - ✓ y=x++; y=++x;
      - ✓ if (x = 2)
    - Type conversion and promotion
    - Other operators (bitwise, bit shifting , compound assignment, conditional)
      - ✓ Bit: |, &, ~. ^, << >>
      - ✓ Compound: x + = 10; x >>= 10; x += y + 3
    - Precedence of operators
- Functions and Program Structure (Chapter 4)

YORK UNIVERSITÉ UNIVERSITY

88

88