1

Last lecture: C basics

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

2

YORK UNIVERSITÉ UNIVERSITY

2

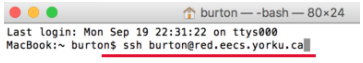
Coding environment summary

https://wiki.eecs.yorku.ca/dept/tdb/covid19:start /tdb/login:sshsupport

- Work on lab directly (work + submit)
  - Remote lab -- an easier way to connect to the lab: (temporary)
    - create folder Desktop browser download directly, or, wget files from terminal →
    - submit 2031A labX fileY.c
  - MAC: ssh pico Windows: Putty, MobaXterm, pico
  - Window/MAC: Virtualbox instructions on course web

- Locally (work + transfer + submit)
  - MAC:
    - work: terminal.
    - transfer: FileZilla MAC, remote lab Guacamole Menu (Ctrl+Alt+Shift), scp command
    - submit: remote lab, ssh

```
scp Point2.java burton@red.eecs.yorku.ca:submit
scp SpiroUtil.java burton@red.eecs.yorku.ca:submit
```



  - Window:
    - work: GCC compiler called MinGW
    - transfer: FileZilla, WinSCP, MobaXterm, remote lab Guacamole Menu (Ctrl+Alt+Shift)
    - submit: remote lab, Putty, MobaXterm
    - Ubuntus on Windows:
      - ❖ Dual boot, got gcc graphic
      - ❖ WLS need to install gcc, no graphic. need command line for two file systems.

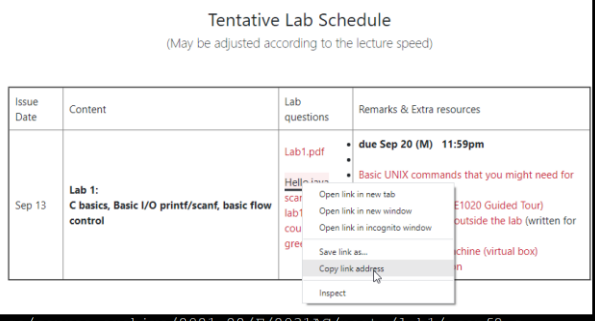
Let me know if you need help

cp hi.c /mnt/C/User/Desktop

3

- Download files from terminal: wget

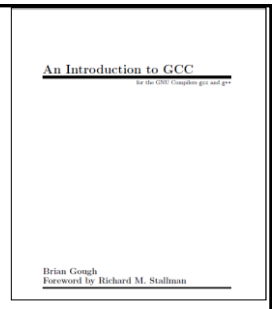
red 310 % wget https://www.eecs.yorku.ca/course_archive/2021-22/F/2031AC/posts/lab1/Hello.java




```
indigo.cse.yorku.ca - PuTTY
red 330 % wget https://www.eecs.yorku.ca/course_archive/2021-22/F/2031AC/posts/lab1/scanf2.c
--2021-09-14 12:50:55--  https://www.eecs.yorku.ca/course_archive/2021-22/F/2031AC/posts/lab1/scanf
.c
Resolving www.eecs.yorku.ca (www.eecs.yorku.ca)... 130.63.94.24
Connecting to www.eecs.yorku.ca (www.eecs.yorku.ca)|130.63.94.24|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 541 [text/x-c]
Saving to: 'scanf2.c'

100%[=====================================================>] 541         --.-K/s   in 0s

2021-09-14 12:50:55 (44.9 MB/s) - 'scanf2.c' saved [541/541]

red 331 % ls
scanf2.c
red 332 %
```

4

gcc c compiler



- `-o` exectuableName

```
% gcc  hello.c  -o hello
% gcc  -o xyz hello.c
```

- default in lab: C89 (ansi) + some C99 `//`

```
int i; for (i=0; i<10;i++)   ok in C89 and lab
for (int i=0; i<10;i++)      c99 only, not ok in lab
```

error: 'for' loop initial declarations are only allowed in C99 mode
note: use option -std=c99 or -std=gnu99 to compile your code

- `-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 now
```

- `-Wall` (warning all)

```
% gcc -Wall hello.c
```

hello.c:11:1: warning: control reaches end of non-void function [-Wreturn-type]

- combine

```
% gcc  -ansi  -Wall hello.c -o hel
```

5

5

functions

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

- ***Must be declared or defined before point of the (first) call !***
  - ***Otherwise compiling error C89, C99***

– different from Java

- Declaration (prototype) – describe arguments and return type, but no implementation

```
int sum (int i, int j);      or   int sum(int, int);
float sum2(int i, float j);  or   float sum2(int, float);
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;          /*  int s = i + j; return s; */
}

float sum2 (int i, float j){
   return i+j;
}

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

6

6

functions

```
#include <stdio.h>

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

main()
{
  float x = 2.1; int y = 2;
  float su  = div(x , y);
  printf( "%f / %d = %f\n", x, y, su);
}
```

Defined before (first) call

functions

```
#include <stdio.h>




main()
{
  float x =2.1; int y=3;
  float su = div(x,y);
  printf( "%f / %d = %f\n", x,y, su);
}

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

Not Defined or Declared before (first) call

Little luckier if return int…

Defined after (first) call

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

functions

```
/* Contains declaration
(prototype) of printf() */
#include <stdio.h>

/* function declaration */
float div (float, int); /* float div (float numerator, int denominator);
                           preferred for readability*/

main()
{
  float x =2.1; int y=3;
  float su = div(x,y);
  printf( "%f / %d = %f\n", x,y, su);
}

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

Declared before (first) call

Defined after (first) call

What about printf()? Declared or defined?

9

9

Basic I/O functions <stdio.h>

- Every program has a Standard Input: keyboard
- Every program has a Standard Output: screen
- Can be redirected in 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

- **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)

10

10

format string

/* conversion specification */

- **printf("This is day %d of Sep\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 to be replaced/filled with an integer (decimal)** "place holders"
  - **%c to be replaced/filled with a character**
  - **%f to be replaced/filled with a floating point number (float, double)**
  - **%s to be replaced/filled with a "string" (array of chars)**
  - **…**

```
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);
```

11

Read two floats

```
#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 wrong e.g., 0.00

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

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

13

getchar, putchar (Ch 1.5)

- **`int getchar(void)`**
  - To read one character at a time from the *standard input*
  - Returns the next input char each time it is called;
  - Returns EOF when it encounters end of file.
    - end of file;
      - Using < : end of input file
      - keyboard: Ctrl-D (Unix) or Ctrl-Z (Windows). *"Keyboard is a file"*
    - EOF: an **`int`** constant defined in <**`stdio.h`**>, value is -1.

- **`int putchar(int c)`**
  - Puts the character c on the *standard output*
  - Returns the character written (usually ignored);
  - Like **`printf("%c", c);`**

getchar, putchar (Ch 1.5)

- **countChar.c**

```
#include <stdio.h> // defines EOF

main(){
 int c;
 int count = 0;

 c = getchar();
 while(c != EOF) /* no end of file*/
 {
   count++; //include spaces and '\n'
   c = getchar(); /* read next */
 }
 printf("# of chars: %d\n",count);
}
```

```
red 309 % a.out
hello
how are you
i am good
^D
red 310 %
# of chars: 28
```

Redirected from file

```
red 312 % cat greeting.txt
hello
how are you
i am good
red 313%

red 314 % a.out < greeting.txt
 # of chars: 28

red 315 % a.out < greeting.txt > out.txt

red 316 % cat out.txt
# of chars: 28
```

redirect input from a file

redirect output to a file

getchar + putchar (Ch 1.5)

- **copy.c**

```
#include <stdio.h>

main(){
 int c;
 c = getchar();
 while(c != EOF)
 {
   putchar(c);

   c = getchar(); /*read next*/
 }
}
```

red 309 % **a.out**
hello ↵
hello
how are you ↵
how are you
^D
red 310 %

Actually get/put chars line by line!

Redirected from file

red 314 % a.out < greeting.txt
hello
how are you
i am good
see you

redirect input from a file

red 315 % a.out < greeting.txt > out.txt

red 316 % cat out.txt
hello
how are you
i am good
see you

redirect output to a file

Not
hheellllloo
Buffer until '\n' or EOF

16

getchar, putchar

- **copy + 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 red % **a.out**
hello ↵
hello
how are you ↵
how are you
i am good ↵
i am good
^D
char:28 line:3

red 337 % **cat greeting.txt**
hello
how are you
i am good

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

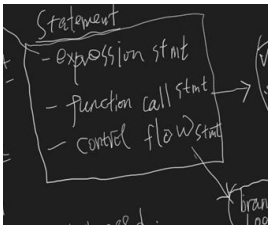
char 'a' 'b' compared directly. Strings not "a"== "b" ✗ Will elaborate today.

17

C basics

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

Statements



- Program to execute
  - Ended with a ;

- Expression statement (ch2)
  - `y = 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

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));

20

Statements

- In ANSI-C (C89): all declarations must appear at the **start** of block, before any 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)

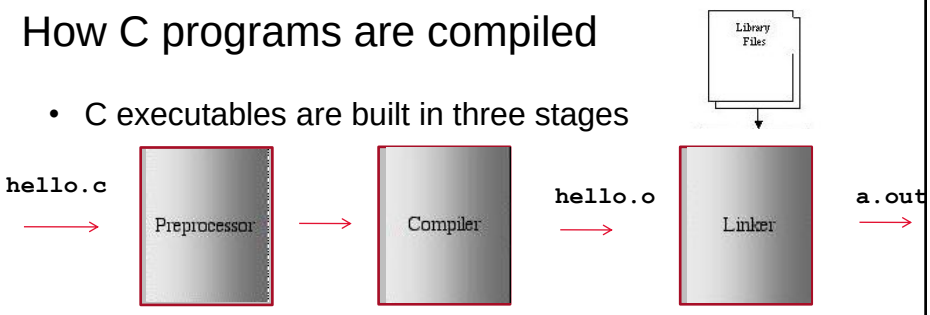
```
gcc hello.c
```

For your information

21

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 object files together,
- linking object files **hello.o** with necessary system libraries (e.g., code of **printf()**)
- creates an executable program e.g., **a.out**
- remove **hello.o**

We might look into this later

23

23

Preprocessing: # include, #define

Textual replace/copy

Declarations/ prototypes

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

}
```

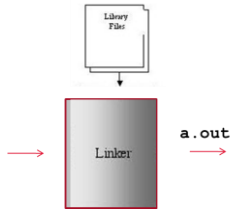
```
int printf (..);
int scanf(..);

int getchar();
int putchar(int);

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

#define EOF -1
```

24

Preprocessing: # include, #define

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

int getchar();
int putchar(int);

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

#define EOF -1
```

```
main()
{
   int i = 4;
   printf("this is %d\n",i);


}
```



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

#define directive

- Syntax `#define name value` No type no ;
  - 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;
}
```

  - Textual replacement
  - Use as constant `N = x + 2;`

```
#define LENGTH 10
#define WIDTH 5
#define NEWLINE '\n'
```

`cpp file.c` or `gcc -E file.c`

Summary

- Course introduction. C basics
  - Variables:
    - names don't start with digit, _ , keyword
  - Functions: declaration vs definition
  - Basic IO functions
    - **scanf & printf,**
    - **getchar putchar**
  - Expression and statements
  - C-preprocessing (brief) **#include #define**

Same in Java

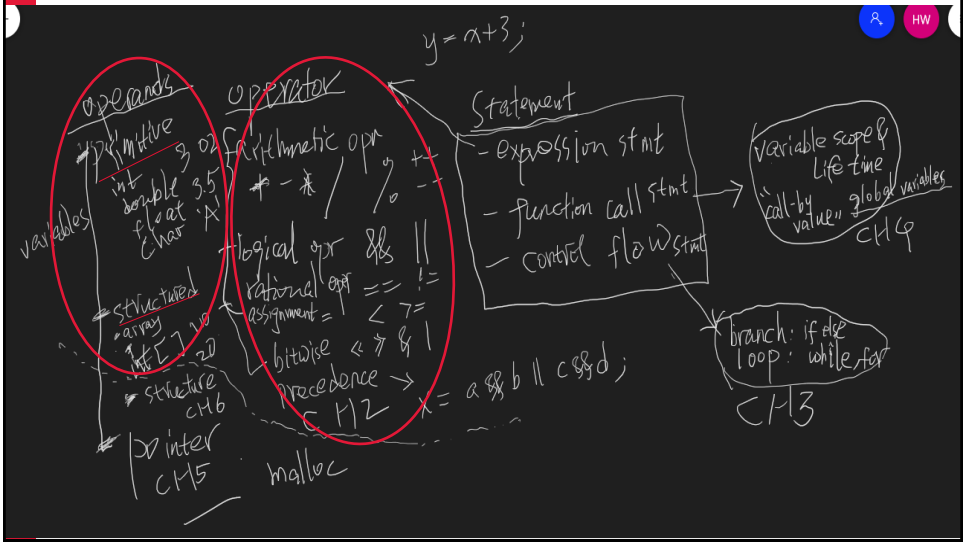
- Today's lecture:
  - C data, type, operators (Ch 2)
  - C flow controls (Ch 3) self-study

27

27

Roadmap -- How the topics are related

RECALL



28

EECS2031-Software Tools

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

YORK UNIVERSITÉ UNIVERSITY

29

Outline

- [Primitive/scalar] Types and sizes
  - Primitive Types
  - Constant values (literals)
- [Structured/aggregated] Array and "strings"

Java primitive types

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

30

YORK UNIVERSITÉ UNIVERSITY

30

RECALL

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

31

31

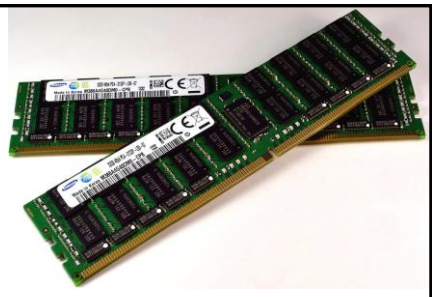
C Primitive Types and sizes

Text book:
4 basic types: char, int, float, double
3 qualifiers: short, long, unsigned

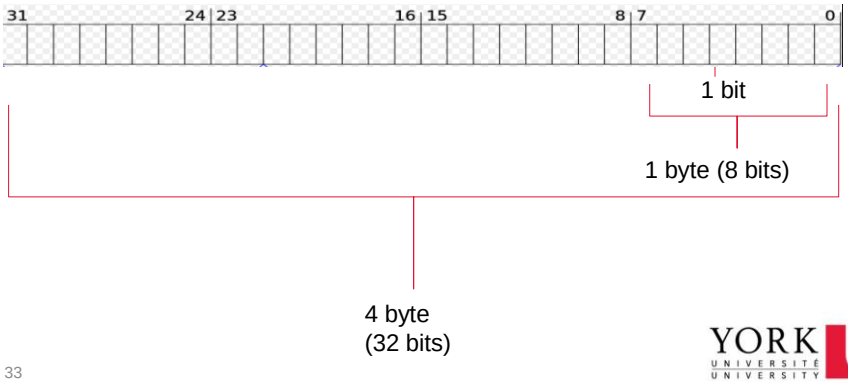
- Variables and values have types
- There are two basic types in ANSI-C: integer, and floating point
  - **Integer type**
    - **char** - character, 1 byte (8 bits)
    - **short (int)** - short integer, usually 2 bytes (16 bits)
    - **int** - integer, usually 2 or 4 bytes (16 or 32 bits)
    - **long (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

32

32



- Bit/byte/K/M/G/T
- `int a;`

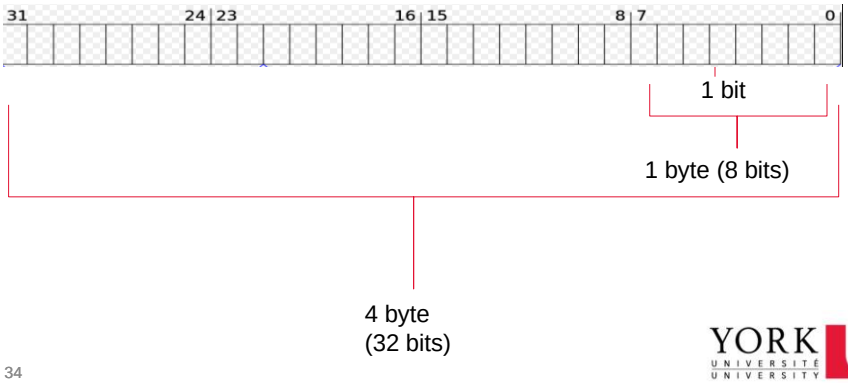


33

33

kB	kilobyte	2^{10} = 1 024 bytes	10^3 = 1 000 bytes
Mb	Megabyte	2^{20} = 1 048 576 bytes	10^6 = 1 000 000 bytes
Gb	Gigabyte	2^{30} bytes = 1,073,741,824 bytes	10^9 = 1000 000 000 bytes
Tb	Terabyte	2^{40} bytes = 1,099,511,627,776 bytes	10^{12} = 1000 000 000 000 bytes

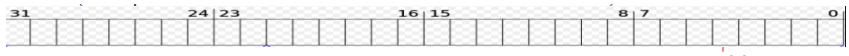
- `int a;`



34

34

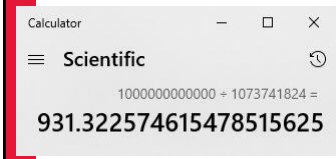
kB	kilobyte	2^{10} = 1 024 bytes	10^3 = 1 000 bytes
Mb	Megabyte	2^{20} = 1 048 576 bytes	10^6 = 1 000 000 bytes
Gb	Gigabyte	2^{30} bytes = 1,073,741,824 bytes	10^9 = 1000 000 000 bytes
Tb	Terabyte	2^{40} bytes = 1,099,511,627,776 bytes	10^{12} = 1000 000 000 000 bytes



1 bit

1 byte (8 bits)

- Be careful;



For your information

35

35

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

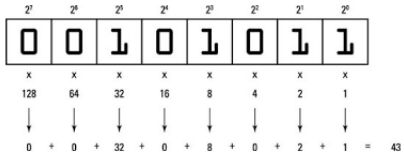
Binary Notation

- 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**

YORK UNIVERSITÉ UNIVERSITY

37

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

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

… 2^6 2^5 2^4 2^3 2^2 2^1 2^0

Binary

$1*2^2 + 0*2^1 + 1*2^0 =$
$4 + 0 + 1 = 5$

Binary representations, arithmetic

- 3 + 4 ?

00…0 0 1 1
00…0 1 0 0
00…0 1 1 1 7

- 3 + 1 ?

0 0…0 0 1 1
0 0…0 0 0 1
0 0 …01 0 0 4

- 3 + 5 ?

0 0…0 0 1 1
0 0…0 1 0 1
0 0…1 0 0 0 8

- 3 + 7 ?

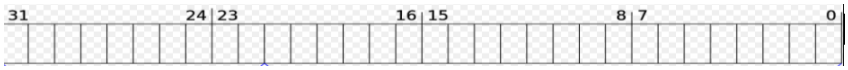
0 0…0 0 1 1
0 0…0 1 1 1
0 0…1 0 1 0 10

Remember that there are only 2 digit symbols in binary, 0 and 1

1 + 1 is 0 with a carry
1+1+1 is 1 with a carry

00…01 0 1 0 1 1 1 87
00…01 0 0 1 0 1 1 75
00…1 01 0 0 0 1 0 162

Qualifiers (modifiers) for integer type



Assume all 32 bits are magnitudes.

- Max value: 1111111….11111
- Min value: 0000000….00000
- How many values: 2^{32} values 2^n
- Range: $0 \sim 2^{32}-1$ $0 \sim 2^n-1$

Negative number?

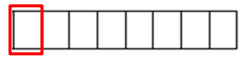
YORK

40

Qualifiers (modifiers) for integer type

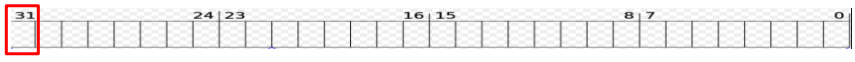
- signed, unsigned qualifiers can be applied to integer types
  - Signed: default. Left most bit signifies sign 0: positive 1: negative
  - Unsigned: positive. Left most bit contributes to magnitude too

- `(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



`unsigned int` $0 \sim 2^{32}-1$ 2^{32} values Max: 1111111….11111

`(signed) int` $-2^{31} \sim 2^{31}-1$ 2^{32} values Max: 0111111….11111

$-2^{n-1} \sim 2^{n-1}-1$ 2^n values

Min?

41

Qualifiers (modifiers) for integer type

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

- 2's complement: "flip + 1"
  - -2's binary representation?
    - 2's binary representation flip + 1
    - 11111101 + 1 = 11111110
  - 11111110's decimal?
    - - (flip +1)
    - - (00000001+1) = -(00000010) = -2

signed value

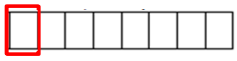
Bits	Unsigned value	2's complement value
00000000	0	0
00000001	1	1
00000010	2	2
01111110	126	126
01111111	127	127
10000000	128	−128
10000001	129	−127
10000010	130	−126
11111110	254	−2
11111111	255	−1
	$0 \sim\sim 2^n-1$	$-2^{n-1} \sim\sim 2^{n-1}-1$
	2^n=256 values	2^n=256 values

44

44

Qualifiers (modifiers) for integer type

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



Why unsigned?

-- potentially save bits

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

- Assume `short int` is 1 byte (8 bits)
- If declared `(signed) short`, max 127, thus need more bits

  `short counter = 150` overflow
- If declared `unsigned short`, max 255, thus 8 bits are enough.

  `unsigned short counter = 150;`

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

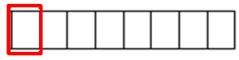
Java: no direct support for unsigned int -- always signed. Thus less types

45

45

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)

For your information

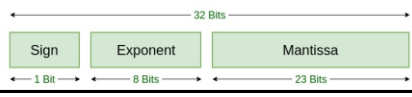
YORK UNIVERSITÉ UNIVERSITY

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 or %lf double ("%Lf") for long double

- Storage of floating point is complicated.
  - **`float x=4.8, float y = 6.4/2+1.6; x == y`** may not always true.
- No unsigned. All signed



Summary

- Integer types:
  - `char ?`
    `signed char` `unsigned char`
  - `(signed) short` `unsigned short`
  - `(signed) int` `unsigned int`
  - `(signed) long` `unsigned long`
- There are three types of (signed) floating points:
  - `float /* single-precision */`
  - `double /* double precision */`
  - `long double /* extended-precision */`
- C99 added:
  - `(signed) long long int`
  - `unsigned long long int bool`

Java defines

Type
int
short
long
byte
float
double
char
boolean

48

Size of Types

- 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 bytes
`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`

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

49

Get exact size of a type in a machine

Size of Types

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

In Java, no direct equivalent

```
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));
```

50

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…..

In Java, no direct equivalent

```
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

51

So How Big Is It?

- Might need to know the min/max of types,
  - avoid overflow. **`int x = 34589643`**? **`short x = 389643`**?
  - signed: $-2^{n-1} \sim 2^{n-1}-1 \rightarrow -2^{\text{sizeof(x)*8-1}} \sim 2^{\text{sizeof(x)*8-1}}-1$
  - unsigned: $0 \sim 2^{n}-1 \rightarrow 0 \sim 2^{\text{sizeof(x)*8}}-1$
- **`<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 K&R book

52

For your information

YORK UNIVERSITÉ UNIVERSITY

52

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

53

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

int x = 34589643? short x = 389643?

```
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 K&R book

54

For your information

54

Outline

- **(Primitive) Types and sizes** — today
  - **Types:** char, short, int, long, unsigned short, unsigned int, float, double …..
  - **Constant values (literals)**
    - **char**
    - **int**
    - **float**

  Next class
- **Array and "strings"**
- Expressions
  - Basic operators
  - Type promotion and conversion
  - Other operators
  - Precedence of operators

55

55