EE261

Introduction to Computer Programming and Software Design

Daqing Hou

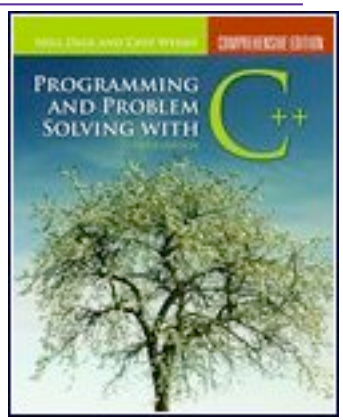
Course management

- Instructor
  Daqing Hou (**q=>ch, ou=>o**)
  Assistant Prof. of Software Engineering
  http://www.clarkson.edu/~dhou
- Teaching Assistant
  Chandan Rupakheti, PhD student
- course web site; **must visit regularly**
  http://www.clarkson.edu/~dhou/courses/EE261-f09
- Office hours
  MW 13:30-15:00 F: 14:00-15:00
  CAMP 127

Course management

- Required textbook and software

(1) Nell Dale, PhD, University of Texas, Austin, Chip Weems, University of Massachusetts, Amherst
Programming and Problem Solving with C++, 5e with the CD-ROM version of A Laboratory Course in C++, 5e. (Textbook+CD ROM Bundle.)
ISBN: 0763779792



Available at the Clarkson bookstore.

(2) Microsoft Visual C++ 2008 Express Edition.
Available free online. Click this link to download.

Course management

- Work load and evaluation
  lab exercises (25%, including 5% for participation)
  homework assignments (25%)
  midterm exam (15%)
  two one-hour quiz's (10%)
  final exam (25%)

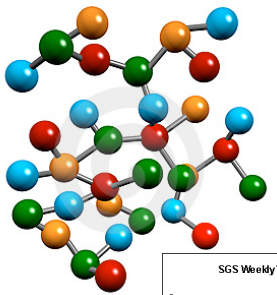
- Budget about six hours per week for this course

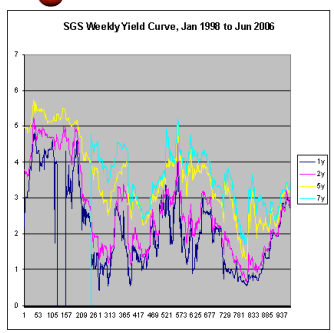
Course management

- Do you have a portable computer?
- Your experience with programming languages C++? Java? C#?
- Anything else that you'd like me to know about?

Why learn to program a computer?

Computers are important components in modern society





- engineering (civil, mechanical, ...)
- sciences (human genome, molecular chem)
- financial and business
- life (control software in washing machines, my HONDA CRV, gaming, social networking, ...)

Some fundamental notions

... Software is just the latest reflection of a history of humans inventing and using processes. In fact, I claim the whole purpose of software is to implement process. The fact that software is involved in the process is a distraction, the real issue is the role and quality of the process being implemented.

Yes, there is a scale difference in that computers do more, do it faster, and so we can perform more complex processes. But this complexity is nothing new either - complex processes have been implemented in large bureaucracies for a period far longer than either engineering or computing science have existed.

H. James Hoover, Professor, Dept. of Computing Science, University of Alberta, Canada

"A Manifesto for Software Engineering at the University of Alberta", Version 1.7a, November 14, 2001.

Our life is full of procedures/processes regardless computers

- how to tie shoelaces (url)
- how to make a cravat for a wedding suit (video)
- how to tie a tie - the full windsor knot (video 1, video 2)
- cooking recipes (e.g, Linzer Cookie Recipe)
- knitting patterns
- musical scores
- directions for getting to airport in strange city
- directions for use (e.g., How Does a Photo Enlarger Works? See Programming Warm-up Exercises on Page 40 of the book.)
- how to peel potatoes (video 1, video 2)
- ... yours?

action, finite period of time, net effect, state

- The first notion is that of an action. An action is a happening, taking place in a finite period of time and establishing a well-defined, intended net effect. In this description, we have included the requirement that the action should be "intended", thereby stressing the purposefulness. If we are interested in action at all, it will be by virtue of its net effect.

- The requirement that the action should take place in a finite period of time is most essential: it implies that we can talk about the moment *T*0, when the action begins, and the later moment *T*1, when the action ends. We assume that the net effect of the action can be described by comparing "the state at moment *T*0" with the "state at moment *T*1".

- An example of an action would be a housewife peeling the potatoes for the evening dinner. The net effect is that the potatoes for the evening dinner are at moment *T*0 still un-peeled, say in the potato basket in the cellar, while at moment *T*1 they will be peeled and, say, in the pan they are to be cooked in.

  next six slides copied from http://www.cs.utexas.edu/~EWD/transcriptions/

sequential process

- When we dissect such a happening as a time sequence of (sub)actions, the cumulative effect of which then equals the net effect of the total happening, then we say that we regard the happening as a sequential process, or process for short.

- The happening of the potato-peeling housewife could, for instance, be described by the time-succession of the following sub-actions of the housewife:

  "fetches the potato basket from the cellar;
  fetches the pan from the cupboard;
  peels and puts the potatoes into pan;
  returns the basket to the cellar" .

- Another example: turning a page by hand. Page 2 of the book.

- More examples: start a car (bottom, page 4); weekly wages (top, page 5)

pattern, algorithm

- We postulate that in each happening we can recognize a pattern of behaviour, or pattern for short; the happening occurs when this pattern is followed. The net effect of the happening is fully determined by the pattern and (possibly) by the initial state (i.e. the state at moment *T*0). Different happenings may follow the same pattern; if these happenings establish different net effects, the net effect must have been dependent on the initial state as well, and the corresponding initial states must have been different.

- An algorithm is the description of a pattern of behaviour, expressed in terms of a well-understood, finite repertoire of named (so-called "primitive") actions of which it is assumed a priori that they can be done (i.e. can be caused to happen).

- An algorithm should finish in finite time (Page 4).

When two happenings can be considered the same?

- We return for a moment to the housewife. On a certain day she has peeled the potatoes for the evening dinner and we have an eye-witness account of this happening. The next day, again, she peels the potatoes for the evening dinner and the second happening gives rise to an eye-witness account equal to the previous one. Can we say, without further ado: "Obviously, the two accounts are equal to each other for on both occasions, she has done exactly the same thing."?

- How correct or incorrect this last statement is depends on what we mean by "doing the same thing". We must be careful not to make the mistake of the journalist who, covering a marriage ceremony, reported that the four bridesmaids wore the same dress. What he meant to say was that the four dresses were made from material of the same design and —apart from possible differences in size— according to the same pattern.

When two happenings can be considered the same?

- The two actions of the housewife are as different from each other as the dresses are: they have, as happenings, at least a different identity: one took place yesterday, one today. As each potato can only be peeled once, the potatoes involved in the two happenings have different identities as well; the first time the basket may have been fuller than the second time; the number of potatoes peeled may differ, etc.

- Yet the two happenings are so similar that the same eye-witness account is accepted as adequate for both occasions and that we are willing to apply the same name to both actions (e.g. "peeling the potatoes for the evening dinner").

Algorithm must be made of well-understood, finite repertoire of named actions.

- In writing down an algorithm, we start by considering the happening to take place as a process, dissected into a sequence of sub-actions to be done in succession. If such a sub-action occurs in the well-understood, finite repertoire of named actions, the algorithm refers to it by its name. If such a sub-action does not occur in the finite repertoire, the algorithm eventually refers to it by means of a sub-algorithm in which the sub-action, in its turn, is regarded as a process, etc. until at the end all has been reduced to actions from the well-understood, finite repertoire.

- In our definition of an algorithm we have stressed that the primitive actions should be executable, that they should be done. "Go to the other side of the square." is perfectly acceptable, "Go to hell.", however, is not an algorithm but a curse, because it cannot be done.

- Besides that we have stressed that the repertoire should be well-understood: between the one who composed the algorithm and the one who intends to follow it there should be no misunderstanding about this repertoire. (In this respect knitting patterns are, as a rule, excellent, recipes are of moderate quality while the instructions one gets when asking the way are usually incredibly bad!) How essential this lack of understanding is may perhaps best be demonstrated by a recipe for jugged hare as it occurs in an old Dutch cookery-book; translated into English the recipe runs as follows: "One taketh a hare and prepareth jugged hare from it." The recipe is not exactly wrong, but it is hardly helpful!

Algorithm must be made of well-understood, finite repertoire of named actions.

"fetches the potato basket from the cellar;
fetches the pan from the cupboard;
peels and puts the potatoes into pan;
returns the basket to the cellar" .

What we have learned so far

- An agent capable of executing a well-understood, finite repertoire of primitive actions.
- Algorithm, a description of a pattern of behavior in terms of the primitive actions that the agent can understand and execute.
- Each primitive action and an algorithm have a net effect. Net effect can be described in terms of how state changes before and after the action occurs. Value of algorithm is the net effect its execution can produce.
- examples of agent: human optionally with tools; computers.
- It is assumed that the agent will follow the algorithm to the letter and does not try to act smart. If the result of executing an algorithm does not produce the desired net effect, often the algorithm needs to be fixed.
- What are a computer's primitive actions? This is an important part of what we shall learn in this course.

homework assignment

- Review the process of finding the Greatest Common Divisor (gcd) of two positive integers. That is, given positive integers x and y, gcd(x, y) is an integer that divides both x and y; furthermore, any common divisor of x and y, say d, is also a divisor of gcd(x, y). For example, gcd(50, 30)=10, gcd(169, 91)=13.

- Read Chapter 1.

A fun video from YouTube

Write in C

See also
The Origins of C++
on Page 32

Algorithm extracts the commonality from multiple happenings - conditional statement.

eye-witness account **1**
yesterday (*dark-colored skirt*):

"fetches the potato basket from the cellar;
fetches the pan from the cupboard;
peels and puts the potatoes into pan;
returns the basket to the cellar".

eye-witness account **2**
today (*light-colored skirt*):

"fetches the potato basket from the cellar;
fetches the pan from the cupboard;
puts on apron because of light-colored skirt;
peels and puts the potatoes into pan;
returns the basket to the cellar".

algorithm peeling potatoes for dinner:

"fetches the potato basket from the cellar;
if (light-colored skirt) do wear apron;
end;
fetches the pan from the cupboard;
peels and puts the potatoes into pan;
returns the basket to the cellar".

conditional statement:

if (condition is true) do
action
end;

Algorithm extracts the commonality from multiple happenings - loops.

eye-witness account **of peeling 6 potatoes:**

"fetches the potato basket from the cellar;
fetches the pan from the cupboard;
peels and puts the potatoes into pan;
peels and puts the potatoes into pan;
peels and puts the potatoes into pan;
peels and puts the potatoes into pan;
peels and puts the potatoes into pan;
peels and puts the potatoes into pan;
returns the basket to the cellar".

- Difficult to describe when peeling large number, say 20, of potatoes.
- **algorithm** peeling **n** potatoes for dinner:

"fetches the potato basket from the cellar;
fetches the pan from the cupboard;
while (#potatoes peeled<n) do
peels and puts the potatoes into pan;
end;
returns the basket to the cellar".

loop:
while** (condition is true) **do
action
end;

infinite loops (see top of page 5), non-termination.

algorithm may never end when the housewife wears light-colored skirt. e.g.,

"fetches the potato basket from the cellar;
fetches the pan from the cupboard;
while (light-colored skirt) do
    peels and puts one potato into pan;
end;
returns the basket to the cellar".

four kinds of control structure

- **sequence**
- **selection** (also called *conditional statement, decision, or branch*)
- **loop** (also called *repetition or iteration*)
- **sub-program** (also called *procedure, function, method, or sub-routine*)

see Figure 1.7

Algorithm defines **a class** of happenings.

gcd (x, y) (Greatest Common Divisor)

```
a) while (x != y) do
   a1)  if (x < y) do
            y = y-x;
        end;
   a2)  if (x > y) do
            x = x-y;
        end;

b) output x;
```

happening 1
t0: x=10, y=10
t1: output 10

happening 2
t0: x=10, y=7
t1: x=3, y=7
t2: x=3, y=4
t3: x=3, y=1;
t4: x=2, y=1;
t5: x=1, y=1;
t6: output 1;

happening 3
t0: x=1334, y=3335 ...
Let's write a C++ program!

Friday Lab, 241 Snell Hall & Homework

study and do the prelab section **before** coming to the lab.

make sure that you are able to log in to the lab computer; understand how to send email to the instructor; be able to launch visual studio for C++ express; do Lesson 1-1 and 1-2.

make sure that your name is checked off when leaving lab to ensure you get the credit for the lab work.

Weekly Homework Assignment

Exam Preparation Exercise 5 (page 39)

Programming Warm-up Exercises 1 and 2 (page 40-41)

Due: Midnight, Monday, August 31

Email your answer to the instructor; put “EE261 Homework 1” as subject line of your email.

2%

Computers, the hardware

- Computer Tour (video)
- Major components of a computer (Sec. 1.3)

  - Main Memory (aka, RAM, or Random-Access Memory)
  - CPU (Central Processing Units = Control Unit + Arithmetic/logic Unit)
  - Input devices (keyboard, mouse, trackpad, touch screen, ...)
  - Output devices (Liquid crystal display - LCD, CRT)
    network card is both Input/Output devices.
  - Auxiliary Storage devices (hard disk, flash memory, floppy disk, CD/DVD)
  - Peripheral devices (printers, scanners, modems, audio sound cards and speakers, microphones and electronic musical instruments, game controllers, digital cameras)
- What makes one computer faster than another (Page 23-24)

The Base-2, Binary Number Systems

- In a computer, data are represented by pulses of electricity.
- Electronic circuits can be either *on* or *off*; a circuit that is *on* represents number *1*; a circuit that is *off* represents number *0*.
- The decimal number system is *base-10*.
  dn dn-1 ... d1 d0, where di can be one of 0 to 9,
  represents value
  *dn*10^n + dn-1*10^(n-1) + ... + d1*10 + d0*
  *e.g., 7456 represents 7*10^3 + 4*10^2 + 5*10^1 + 6*
- The binary number system is *base-2*.
  dn dn-1 ... d1 d0, but *di* can be either 0 or 1.
  e.g., 10 represents 2; 11 represents 3; 100 represents 4; 101 => 5
  110=>6; 111=>7; 1000=>8; 1001=>9; 1010=>10; 1011=>11
- Why computer adopts binary number system? It is easier to implement with circuits than other number systems.

Binary Representation of Data (& instructions)

- Each 0 and 1 in a binary data is called a bit (binary digit).
- A binary of 3 bits can represent 8 different numbers:
  000=>0; 001=>1; 010=>2; 011=>3; 100=>4; 101=>5; 110=>6; 111=>7
- To represent decimal 8, at least 4 bits are needed: 1000
- A binary of 8 bits can represent 256 numbers (0 to 255).
  Every contiguous 8 bits make a byte.
- byte is the basic unit for measuring capacity of RAM and hard disks.
  1 Mega bytes = 2^20 ~ 1 million
  1 Giga bytes = 2^32 ~ 4 billions
- addition: 001+001=010; 010+010=100
  subtraction: 011-011=000; 011-001=010
- All real-world data are represented as binary patterns inside computers.

Source code is represented in ASCII (ask-key)

The **American Standard Code for Information Interchange** is a standard seven-bit code that was proposed by ANSI (American National Standards Institute) in 1963, and finalized

```
#include <iostream>
...
int main(){
...
}
```

```
35
105 110 99 108 117 100 101
32
60
105 111 115 116 114 101 97 109
62
13 10
```

ASCII Table

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

The memory model

address

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

...

byte 4 billion'th

What operations a computer perform?

- **Transfer** data from one place to another.
- **Input** data from input device (a keyboard or a mouse, for example) and output data to an output device (a screen, for example).
- **Store** data to and **retrieve** data from its memory and secondary storage
- **Compare** two data values for equality or inequality.
- Perform **arithmetic operations** very quickly (addition, subtraction, multiplication, division).

- **These operations are also represented as binary and stored in memory!!!**

load x, R6
0001 0010101101010000 110
store x, R6
0010 0010101101010000 110
sub R2, R6, R1
1010 010 110 001
add R1, R3, R5
1001 001 011 101

How a computer execute a program (*fetch-execute cycle*)?

1. The control unit retrieves (*fetches*) the next machine instructions from memory.
   e.g., **sub R2, R6, R1** 1010 010 110 001.
   R6 contains number 5; R1 contains number 3; R2 contains 100
   After this instruction executed, R2 = 2!

2. The control unit translates instruction into control signals.

3. The control signals tell the appropriate unit (arithmetic/logic unit, memory, I/O device) to perform (*execute*) the instruction.

4. The sequence repeats from Step 1 unless an end-of-execution instruction is seen.

Machine language and assembly language

- Machine language is made up of binary-coded instructions that are directly executed by the CPU.
- Assembly language is a low-level programming language in which mnemonic is used to represent the machine language instructions of a particular computer.
- An assembler is a program that translates an assembly program into machine code.

load x, R6
0001 0010101101010000 110
store x, R6
0010 0010101101010000 110
sub R2, R6, R1
1010 010 110 001
add R1, R3, R5
1001 001 011 101

Programming language

- Programming language
  A set of rules, symbols, and special words used to construct a computer program.
- A high-level programming language is easier to use than both assembly language and machine language.
- A compiler translates a source program in a high-level language into an object program in machine code.

load x, R6
0001 001010110101000 0 110
store x, R6
0010 001010110101000 0 110
sub R2, R6, R1
1010 010 110 001
add R1, R3, R5
1001 001 011 101

"**if (x>y) x = x-y**"

x = x-y could be implemented by/ translated into
load x, R6
load y, R1
sub R2, R6, R1
store x, R2

Compilation versus execution (Figs 1.5 and 1.6)

- high-level programming languages allow programs to be compiled on different systems.
- **compilation**: source program => computer executes compiler program => machine language version of source program (object program)
- **execution**: computer loads object program => computer executes machine version of source program, which reads input data and produces results.

Software

- Operating system
  A set of programs that manage a computer's resources.

- An editor
  An interactive program used to create and modify source programs and data.
- A compiler
- A debugger

- office applications: Word Processor, Spreadsheets, Presentation, ...
- web browser

The programming process

- Problem-solving phase
  1. analysis and specification (define what a solution must do)
  2. general solution (algorithm, a logical sequence of steps that solves the problem)
  3. verify (does the solution really solve the problem?)
- Implementation phase
  1. concrete solution (program)
  2. test
- Maintenance
  1. use
  2. revise to address new requirements and correct errors discovered in use

- This is known as the *waterfall model* of software development.
- Another is the *spiral model*.

Software Maintenance

- Useful programs live much longer than its initial development.
- Consequently, skills of archiving, reading and understanding existing code, modifying existing code to add a new feature or fix a bug, are critical.
- Documentation: The written text and comments that make code easier for others to understand and modify.

Ethics and responsibilities in the computing profession (Sec. 1.4)

- Software piracy
- privacy of data
- use of computer resources
  virus
  worm
  zombie
  malware
- software engineering

Problem solving techniques (Sec. 1.5)

- Ask questions
- Look for things that are similar
- Solve by analogy
- Means-ends analysis
- The building blocks approach
- Merging solutions
- Divide and conquer
- Mental blocks: The fear of starting
- Algorithmic problem solving

Lab work and homework for week 2

see course web site.