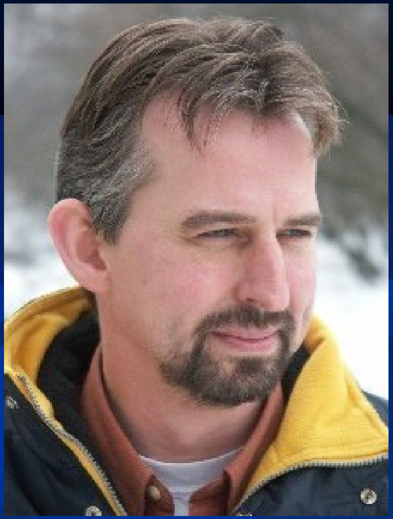
Imperative Programming

- Lecturer: Dr. Arnold Meijster
- e-mail: a.meijster@rug.nl
- Phone: +31 50 363 9246
- room: 5161.343 (Bernoulliborg)



Organisational matters

- The course “*Imperative Programming*” consists of a series of lectures, tutorials, and practicals (computer lab sessions).
- Small programs and aspects of the programming language C are presented in the lectures.
- Students make exercises with pen(cil) and paper and laptops/pcs during the tutorials
- Students make exercises on their own laptops during the computer lab sessions.
  - Labs are made in individually!

Groups + Schedule

The groups + tutorial/lab schedules can be found via the following URL:

http://rooster.rug.nl

It is your own responsibility that you join the right tutorial/lab session at the right time!
If you do not know in which group you are, find out quickly!

Groups are fixed! You are not allowed to switch to another group!

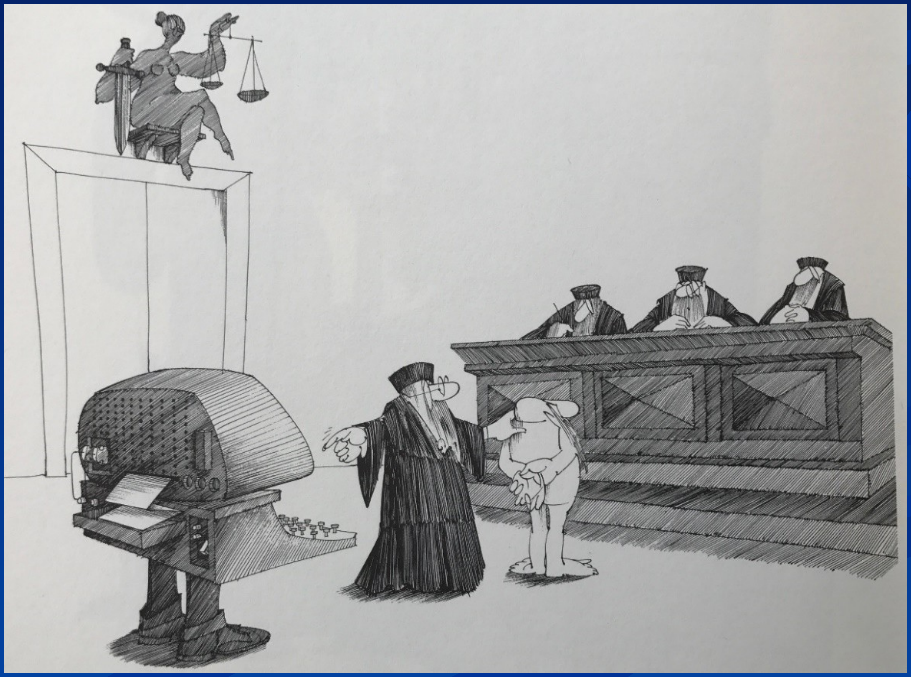
Practicals

- The labs exercises are posted on Nestor (weekly).
  - They are not in the reader!
- Practicals are made individually!
- There is assistance by the teaching assistants (TAs) during the scheduled lab sessions!
- Deadlines : hard!

Practicals/Labs

- There are 4 programming assignments per lab session (one session per week).
  - Exception: week 1 has 5 assignments, since the first assignment is ‘trivial’

- You submit your solutions (source code) to the online electronic verification system Themis. Themis is available 24/7.
  - *http://themis.housing.rug.nl*
  - Once a program has been accepted by Themis, the assignment is completed/done.
  -



Grading of Practicals

- You get 2 points for an accepted solution, yielding 8 points in total.

- The remaining 2 points are given by manual inspection by the TAs. They consider criteria like programming style, layout, efficiency, etc.

- In case Themis does not accept a solution, you can submit an improved version. However, points are subtracted for too many trials:
  - ≤5 submissions: no points are subtracted (for the given problem)
  - 6-15 submissions: 0.1 point is subtracted for each submission (for the given problem)
  - > 15 submissions: 1 full point is subtracted (for the given problem)

- So, test your programs thoroughly before submitting them!

  Note: In the first lab session there are 5 exercises. In this lab, the programs will be inspected manually, but no points are given/subtracted (only feedback is given). Points are subtracted for too many submissions.

Warning: NO plagiarism!

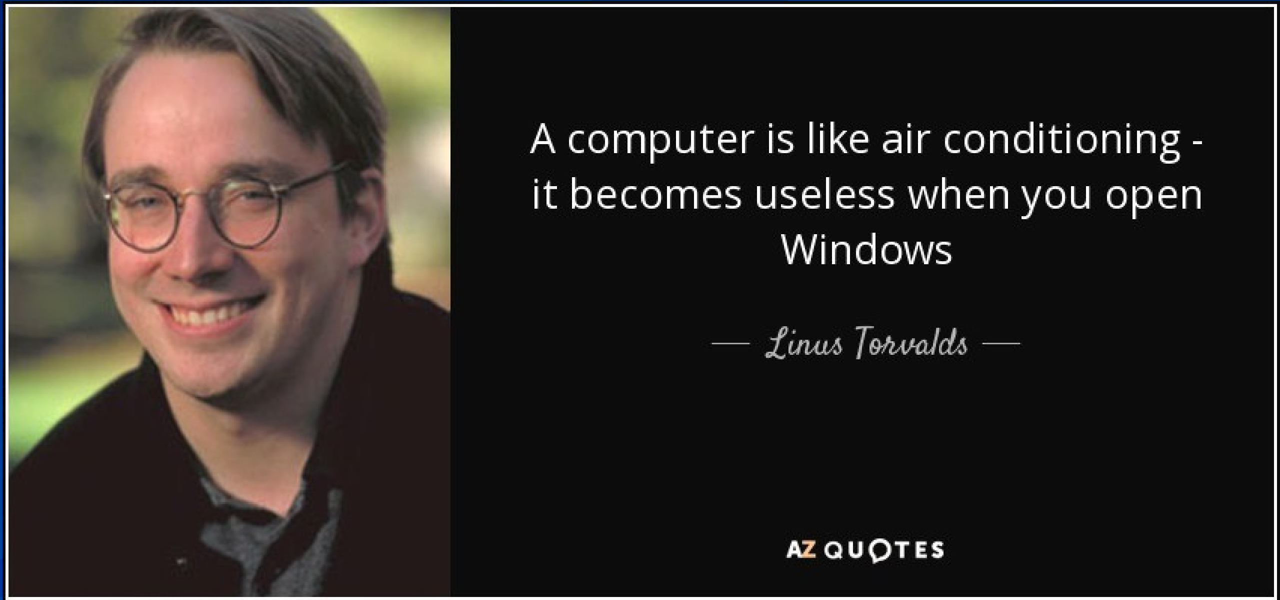
Copyright 2008 by Randy Glasbergen.
www.glasbergen.com
GLASBERGEN
"I need you to do a presentation on the topic of 'plagiarism'. If you don't have time to prepare anything, just steal something off the Internet."

We compare submissions! If we suspect plagiarism (copying) then we forward the submissions to the board of examiners. We also check for copying/ using code from internet fora.

Practicals

- We use Linux!

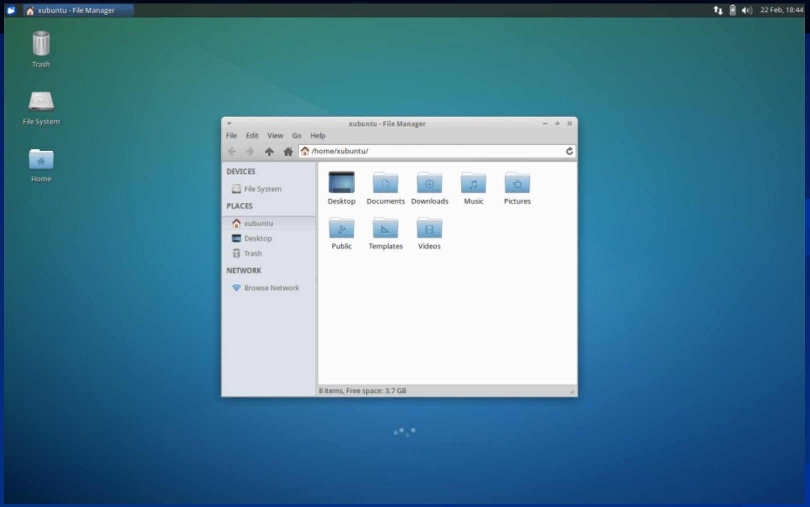
- All admitted RUG-students have an account.
  - S-number + password



A computer is like air conditioning - it becomes useless when you open Windows

— Linus Torvalds —

Linux on your PC/Laptop ?

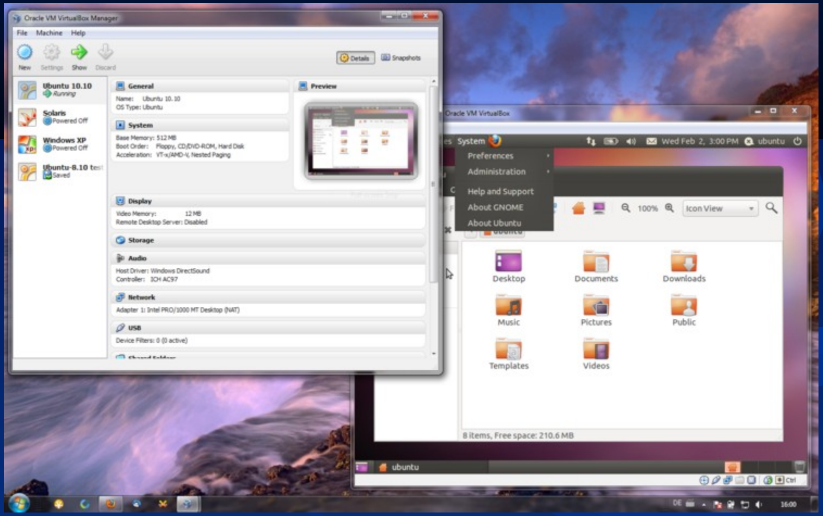


https://xubuntu.org/

Linux on your PC/Laptop ?

Replacing your windows is too scary? Use virtualbox!

https://www.virtualbox.org/



General
Name: Ubuntu 10.10
OS Type: Ubuntu

System
Base Memory: 512 MB
Boot Order: Floppy, CD/DVD-ROM, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging

Display
Video Memory: 12 MB
Remote Desktop Server: Disabled

Audio
Host Driver: Windows DirectSound
Controller: ICH AC97

Network
Adapter 1: Intel PRO/1000 MT Desktop (NAT)

USB
Device Filters: 0 (0 active)

On Nestor, you can find a link to a virtual disc image containing a pre-installed xubuntu system.

Warning: this file is BIG (several GBs)!

Helpdesk

- Outside the scheduled hours for the lab sessions the following e-mail address is available as a digital helpdesk.

iprug20@gmail.com

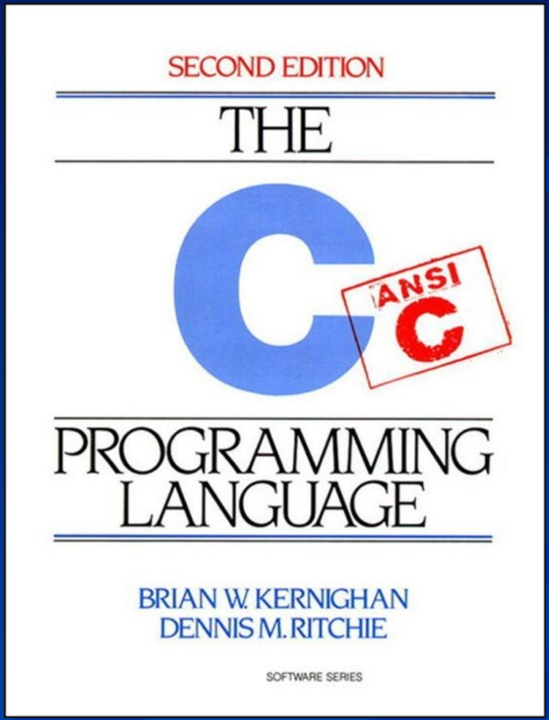
Examination

- All practical sessions must be completed.
  - If (at most) one session is missing, then you will receive the grade 1 for that session, but you are still admitted to the exam!
  - If more than one session is missing, then you will not be allowed to make the exam!
  - There are no resits for lab assignments!
- The grade P for the practical sessions is the mean of the grades for the lab sessions.
- Halfway the course there is a practical mid-exam (Grade M).
- There is a final exam (grade E).
- Final Grade: $F = (P + M + 2E)/4$ provided that $E \geq 5$ (otherwise $F = E$)

Literature

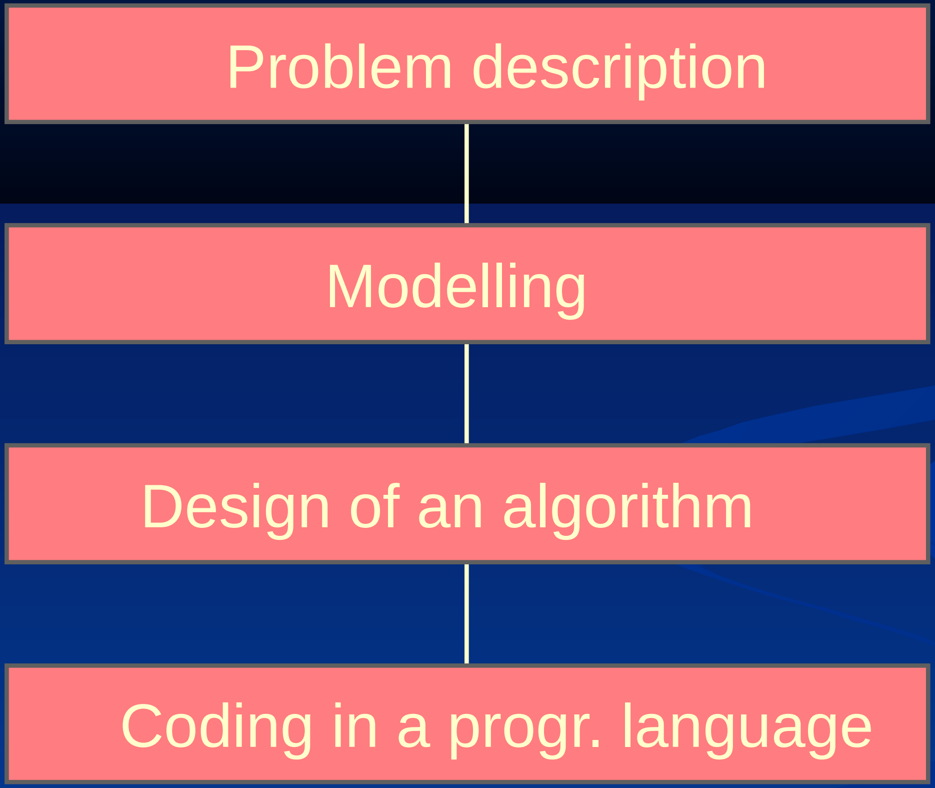
- Reader: Imperative programming
    - available as a pdf via Nestor

Book: C Programming Language (ANSI C)



Programming

Programming is more than (just) coding:



During this course, we will pay attention to all these aspects of programming.

Palindrome (problem description)

Task: write a program that decides whether a word is a *palindrome* or not.

A **palindrome** is a word that reads the same in either forward or reverse direction.

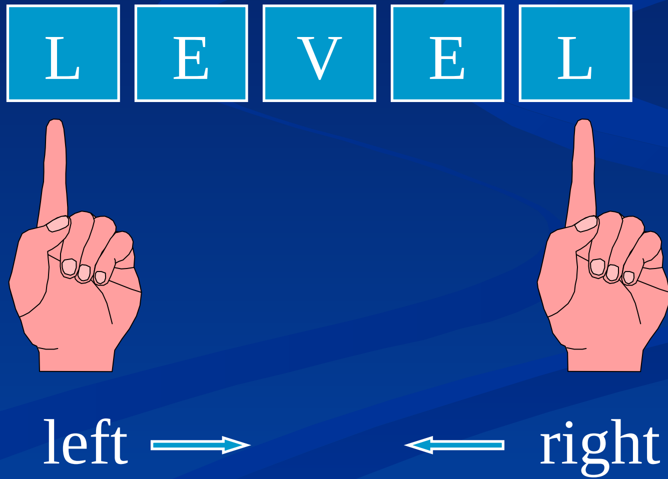
- civic
- level
- madam
- radar
- rotator
- Meetsysteem (Dutch)

Palindrome (modelling)

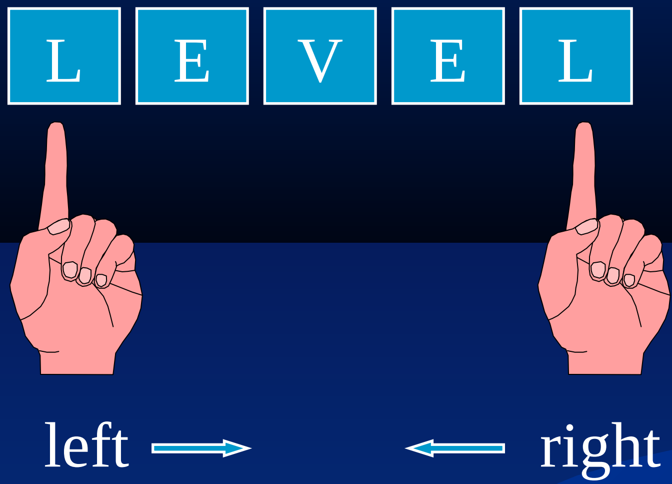
We regard a word as a sequence of characters.

We can use two fingers as pointers in the word:

- one from head to tail
- one from tail to head.



Palindrome (algorithm design)



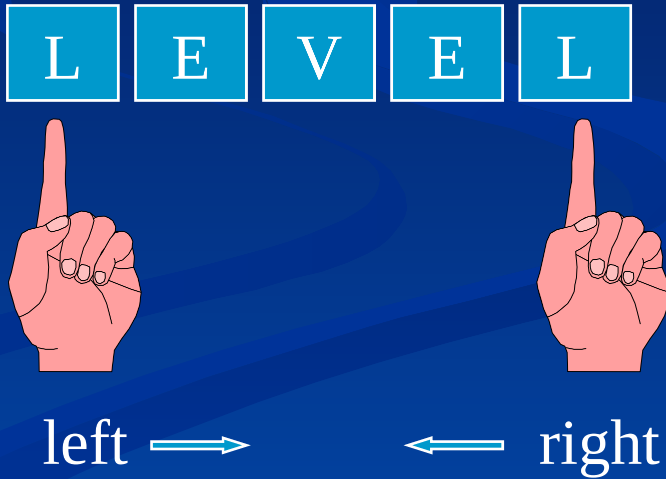
If both fingers point to the same character, then we go on.

Otherwise, we stop: the word is not a palindrome.

We can also stop if the pointers point to the same position (or even passed each other): the word is a palindrome.

Palindrome (algorithm design)

```
LET left point to the first character
LET right point to the last character
REPEAT
   IF character at left equals character at right
   THEN
      shift left 1 position to the right
      shift right 1 position to the left
   ELSE
      OUTPUT "NO"
      STOP
UNTIL left >= right
OUTPUT "YES"
STOP
```



Palindrome (coding)

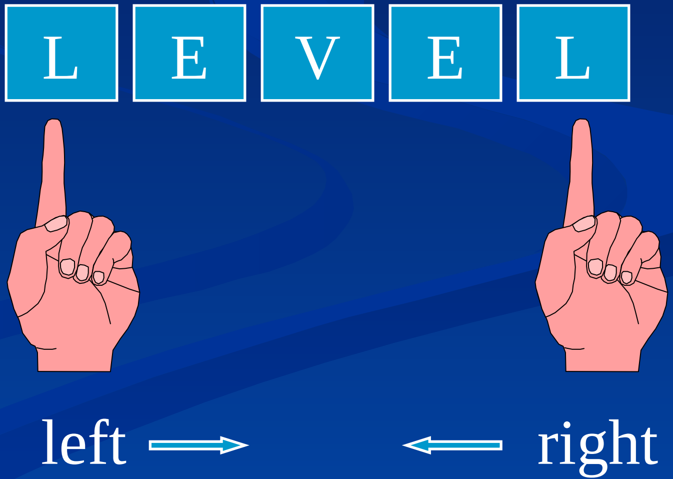
- Finally, we code our algorithm in some programming language. In this course, we will use the programming language C.

- This may seem to be the real hard part, but it really is not. Once you have designed an algorithm, the hard part is done!

- Designing good algorithms can be hard. Converting them into C code is only a matter of routine.

Palindrome (coding)

```
int left = 0;
int right = length - 1;
while ((left < right) && (word[left] == word[right])) {
    left++;
    right--;
}
if (left >= right) {
    printf("YES");
} else {
    printf("NO");
}
```



Statements

- A statement is some elementary command:
  - print “Hello world” on the screen
  - Increment the value of some variable
  - Repeat 10 times
  - Etc.
- A program is basically a sequence of statements.
- Statements are separated by a semicolon ;
- Convention: one statement per line

Comments

- Comments in a program are human-readable text
  - Enclosed by **/*** and ***/**
    - May span several lines of code
  - Or, single line comments starting with `//`
- Comments are ignored by the compiler
  - Intended to make the logic of a program easier to understand
- Don't be lazy! Use comments!
- Example:

```
/* This is a multi line
 * comment.
 */

// This is a single line comment.
```

Structure convention

```c
/* file    : skeleton.c */
/* author  : Joe Sixpack (joe@student.rug.nl) */
/* date    : Mon Aug 31 2020 */
/* version: 1.0 */

/* Description:
 * Here should be a description of
 * what this program is supposed to do.
 */

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
  /* here your own code */
  return 0;
}
```

First example: hello world

```c
/* file    : hello.c */
/* author : Joe Sixpack (joe@student.rug.nl) */
/* date    : Mon Aug 31 2020 */
/* version: 1.0 */

/* Description:
 * This is the standard hello-program.
 * It prints ‘Hello world’ on the screen.
 */

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
  printf ("Hello world!\n");
  return 0;
}
```

file: hello.c

Not like this !!!!!!!!!!!!!

```
/* file    : hello.c */
  /* author : Joe
 Sixpack (joe@student.rug.nl) */
/* date    : Mon Aug 31 20
20 *//* version: 1.0 */

/* Description:
 * Th
is is the standard hello-program.
 * It prints ‘Hello world’ on the screen.
 */

#include <stdio.h>
#include <stdlib.h>
int
main(int argc,
char *                              argv[]) { printf (
   "Hello world!\n");  return   0;}
```

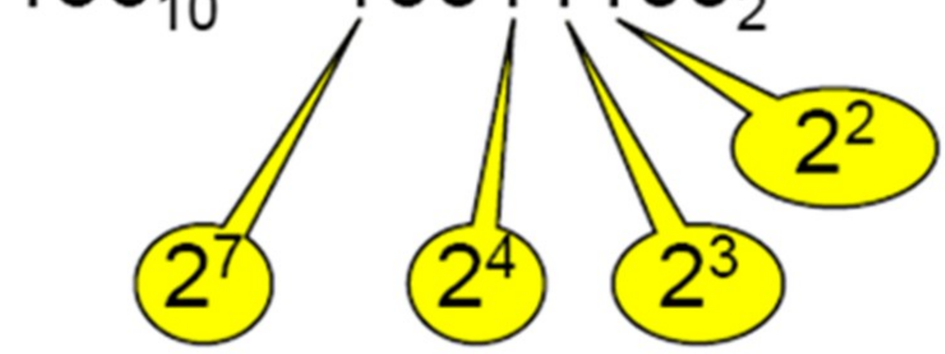
Compilation (Linux)

- Editor: **`geany`** (or **`kate`**, or **`emacs`**, or **`vi`**, or ….)
- Compiler: **`gcc`**
- Compilation: **`gcc –std=c99 –Wall –pedantic hello.c`**
  - Converts C source code in a binary executable **`a.out`**
- Execution: **`./a.out`**

Binary representation of integers

$$156 = 1 \times 10^2 + 5 \times 10^1 + 6 \times 10^0$$

$$156_{10} = 10011100_2$$



$$156_{10} = 1x2^7 + 1x2^4 + 1x2^3 + 1x2^2$$
$$= 128 + 16 + 8 + 4$$

Numerical types (on a PC)

type	kind	bits	maximum
short	integer	16	32767
int	integer	32	2147483647
long	integer	64	>10E18
float	floating point	32	>10E38
double	floating point	64	>10E308

For each type the minimum value is equal to – (maximum+1).

For example, the smallest `int` value is `-2147483648`.

Integers: operators

operation	operator	example
addition	+	17+5=22
subtraction	-	17-5=12
multiplication	*	17*5=85
integer division	/	17/5=3
remainder (modulo)	%	17%5=2

Note that x is equal to `d*(x/d) + x%d` (provided that d is non-zero).

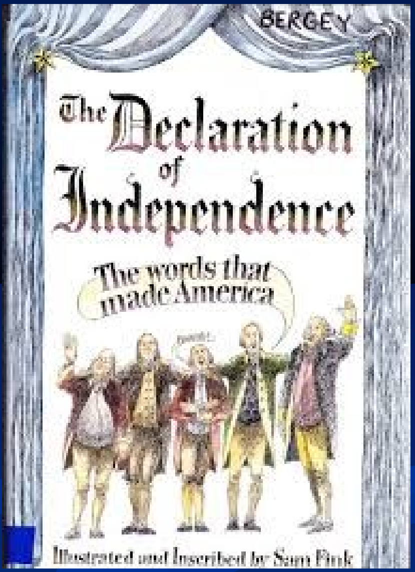
Variables

- In imperative programming *variables* play a central role
  - Data is stored in variables
  - A program modifies the values of variables

- Examples:
  - A student number can be stored in a variable of the type **int**.
  - The number pi (3.14159265…) can be stored in a variable of the type **float** (or **double**).

Declaration of variables

- You need to *declare* a variable before usage.
  - It tells the compiler to reserve memory space.
  - Examples:
    - `int x, small, numberOfPersons;`
    - `float pi;`
    - `double cheeseburger;`
    - `int independence;`


BERGEY
The Declaration of Independence
The words that made America
Illustrated and Inscribed by Sam Fink

- Variable names (*identifiers*) consist of letters, digits and underscores.
  - Convention: we will use the *camelCase* convention for identifiers: each identifier start with a lower-case letter, followed by letters and digits. Each word starts with an upper-case letter (e.g. `numberOfPersons`). We do not use other characters, like underscores!
- Note: C is case sensitive! (`pi` differs from `Pi` or `PI`).

Assignment

The simplest way to give a variable a value is the *assignment statement*.

Assignments have the following form:

<variable> = <expression>;

```
int x, y;

x = 17;     /* x==17         */
y = x + 9;  /* x==17, y==26 */
x = y;      /* x==26, y==26 */
```

Assignment

```
<variable> = <expression>;
```

Conditions:

- The variable has been declared,
- **<variable>** and **<expression>** have compatible types,
- the expression can be evaluated,
- the value of the expression ‘fits’ in the variable.

Declaration with initialisation

Instead of writing

```
int x, y;
x = 17;
y = 12;
```

it is much shorter (and easier to read) to write

```
int x = 17, y=12;      /* x==17, y==12 */
```

Assignments

Exercise: find a series of assignments that satisfies the following *specification*:

```
/* x==A, y==B */
???
/* x==A+B, y==A*B */
```

Assignments

```
/* x == A, y == B */
/* x+y == A+B, y == B */
x = x + y;
/* x == A+B, y == B */
/* x == A+B, (x-y)*y == A*B */
y = (x-y)*y;
/* x == A+B, y == A*B */
```

Assignments

Exercise: find a series of assignments that satisfies the following *specification*:

```
/* x == A, y == B */
???
/* x == B, y == A */
```

Assignments (swap)

The following attempt is clearly wrong!

```
/* x == A, y == B */
x = y;
/* x == B, y == B */
y = x;
/* x == B, y == B */
```

Assignments (swap)

```
/* x == A, y == B */
z = x;
/* x == A, y == B, z == A */
x = y;
/* x == B, y == B, z == A */
y = z;
/* x == B, y == A, z == A */
/* x == B, y == A */
```

Assignments (tricky swap)

```
/* x == A, y == B */
/* x+y == A+B, y == B */
x = x + y;
/* x == A+B, y == B */
/* x == A+B, x-y == A */
y = x - y;
/* x == A+B, y == A */
/* x-y == B, y == A */
x = x - y;
/* x == B, y == A */
```

Special assignments

In C, there are some special abbreviations available for frequently occurring assignments.

assignment:	abbreviation:
`x = x + 1;`	`x++;`
`x = x – 1;`	`x--;`
`x = x op exp;`	`x op= exp;`
e.g. `x = x*(y+1);`	e.g. `x *= y+1;`

Type: char

The type **char** is used to represent characters.

E.g.: **'0'**, **'1'**, .., **'9'**, **'a'**, .., **'z'**, **'A'**, .., **'Z'**, **'#'**, **'!'**

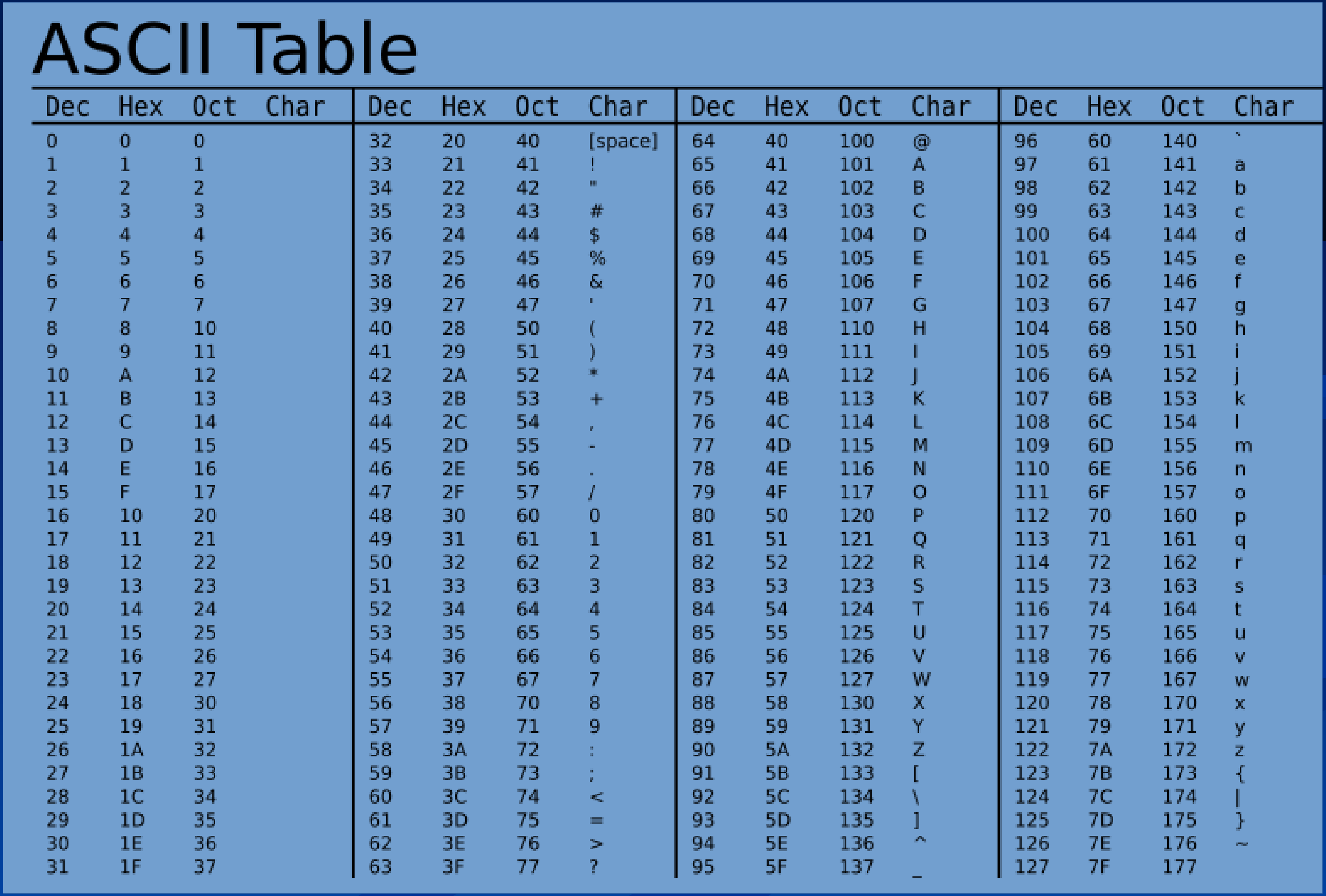
```
char a, letter;
a = 'K';
letter = 'a';
```

Order: defined in the so-called ASCII table.

ASCII = American Standard Code for Information Interchange

'0'<'1'<..< '9', **'a'<'b'<..< 'z'**, **'A'<'B'<..< 'Z'**, **'A'<'a'**

ASCII table



ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

char is actually a small int

The character `’B’` has location (index) 66 in the ASCII table.

Therefore, the following two variables have the same value.

```
char c = ‘B’;
char d = 66;
```

Standard Input/Output (I/O)

- `#include <stdio.h>`
- ***Standard output*** is the screen
- ***Standard input*** is the keyboard

Single Character I/O

- **`int getchar()`**
  reads a single character from standard input
  Note: yields an **`int`** (not a **`char`**)!

- **`putchar(int)`**
  prints a single character on standard output

Example (echo character)

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

int main(int argc, char *argv[]) {
   int c;
   c = getchar();
   putchar(c);
   putchar('\n');
   return 0;
}
```

Formatted Output: printf

- General form: `printf("format", …)`
- Examples:

```
printf("Hello world\n");

printf("%d", 123);

printf("pi=%f and e=%f.\n", 3.1415926, 2.718);
```

Formatted I/O: int specifiers

`%c`	character
`%d`	signed decimal
`%ld`	long decimal

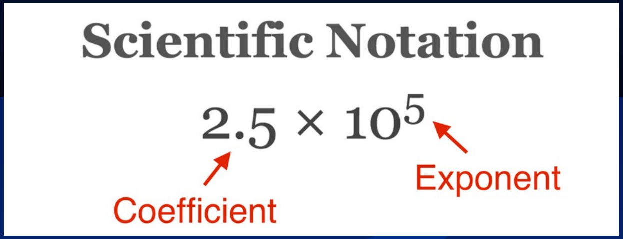
```
printf("%c %d\n", 'e', 42);
```

Output: `e 42`

Formatted I/O: float specifiers

Standard precision is 11 positions (including decimal dot):

```
%f    ddd.ddd
%e    d.dddddde{sign}dd
%E    d.ddddddE{sign}dd
```



```
printf("%f\n", 1234.56789);
printf("%5.3f\n", 1234.56789);
printf("%12.4e\n", 1234.56789);
```

```
1234.567890
1234.568
  1.2346e+03
```

(at least 5 positions, 3 digits after .)

(at least 12 positions, 4 digits after .)

Arithmetic with characters

- The type **`char`** is actually a small integer with which you can do arithmetic.
- `printf ("%c", 'A'+1);`
- This prints the character at location 65+1=66 in the ASCII table. In other words, the character **`'B'`**.

So, this is perfectly valid (but clumsy code) to print 42:

```
char a = '4';   // ‘0’==48, ‘2’==50, ‘4’==52
char b = '2';
printf ("%d\n", 10*(a – '0') + b – '0');
```

Formatted Input: scanf

General form:

```
n = scanf("format", &var, …)
```

BEWARE &

Examples:

```
int i, n;
double d;

scanf("%d", &i);   /* beware!: & */
scanf("%lf", &d);
printf("i==%d and d==%lf\n", i, d);
n = scanf("%d %lf", &i, &d);
printf ("You typed in %d numbers\n", n);
```

Formatted Input: scanf

`scanf()` returns the number of matching input elements

For example: `n = scanf("%d %d", &i, &j);`

Returns (value of n):

- 2 two numbers read from the keyboard
- 1 only one number was read from the keyboard
- 0 no match (i.e. no numbers on the input)
- **EOF** End of file (CTRL-D)

A simple calculator

Exercise: write a program that reads two integers from the keyboard and then prints the sum, difference and product of these numbers.

A simple calculator

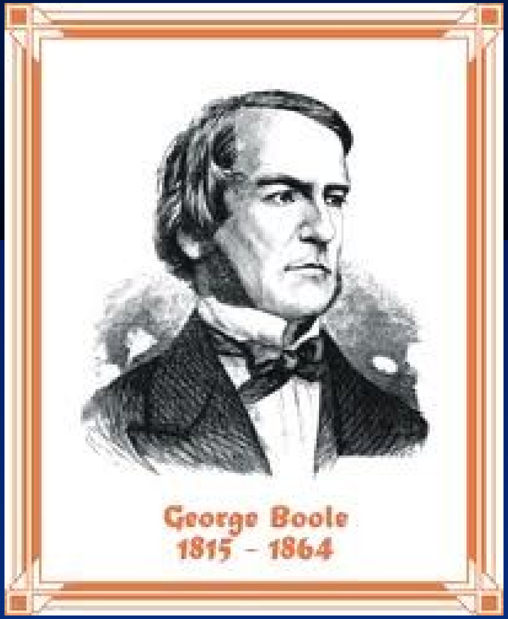
file: calculator.c

```c
#include <stdio.h>
#include <stdlib.h>


int main(int argc, char *argv[]) {
   int x, y;
   printf ("Type 2 integers: ");
   scanf("%d %d", &x, &y);

   printf("%d + %d = %d\n", x, y, x + y);
   printf("%d - %d = %d\n", x, y, x - y);
   printf("%d * %d = %d\n", x, y, x*y);
   return 0;
}
```

Boolean algebra



Boolean algebra acts on the thruth values TRUE (1) and FALSE (0) only.

C-convention: 0 represents false. Any other non-zero value represents true.

Boolean operators

!	not (negation)
&&	and (conditional conjunction)
||	or (conditional disjunction)

In C we have short circuit Boolean evaluation:

Conditional AND:	P&&Q:	IF P is true THEN Q ELSE false
Conditional OR:	P||Q:	IF P is true THEN true ELSE Q

Boolean operators

0=false, 1=true

AND (&&)

Inputs		Output
0	0	0
0	1	0
1	0	0
1	1	1

OR (||)

Inputs		Output
0	0	0
0	1	1
1	0	1
1	1	1

NOT (!)

Input	Output
0	1
1	0

Comparison operators

`==`	equal
`!=`	not equal
`<`	less than
`<=`	less than or equal
`>`	greater than
`>=`	greater than or equal

Examples: Boolean expressions

```
int x = 123;
int y = 200;
int a = x < y;   /* a will be 1 (TRUE) since 123 < 200 */


int doorIsLocked, lightsOn, readyToGo;
readyToGo = doorIsLocked && !lightsOn;


int canBuy, hasCash, hasCreditCard;
double money, cost;
canBuy=(hasCash && money>=cost) || hasCreditCard;
```

IF-statement

General form:

```
if (condition) {
   thenStatements;
};
```

IF-ELSE-statement

General form:

```
if (condition) {
   thenStatements;
} else {
   elseStatements;
};
```

IF-statement

Exercise: read an integer from the keyboard and print whether the number is even or odd.

```
int number;

printf("Type an integer: ");
scanf("%d", &number);

if (2*(number/2) == number) {
  printf("The number %d is even.\n", number);
} else {
  printf("The number %d is odd.\n", number);
};
```

IF-statement (better)

Exercise: read an integer from the keyboard and print whether the number is even or odd.

```
int number;

printf("Type an integer: ");
scanf("%d", &number);

if (number%2 == 0) {
  printf("The number %d is even.\n", number);
} else {
  printf("The number %d is odd.\n", number);
};
```

IF-statement (alternative)

Exercise: read an integer from the keyboard and print whether the number is even or odd.

```
int number;

printf("Type an integer: ");
scanf("%d", &number);

if (number%2) {
  printf("The number %d is odd.\n", number);
} else {
  printf("The number %d is even.\n", number);
};
```

Prime number < 100?

- Exercise: read an integer x from the keyboard (where 1 < x < 100), and print whether the number is prime or not.

A prime number (or a prime) is an integer greater than 1 that is not a product of two smaller positive integers.

Prime number < 100?

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

int main(int argc, char *argv[]) {
  int x;
  printf("Type an integer x (1 < x < 100): ");
  scanf("%d", &x);
  if ((x <= 1) || (x > 99)) {
    printf ("Hey, I told you: 1 < x < 100!\n");
    return -1;  /* stop program with error code -1 */
  }

  if ((x==2) || (x==3) || (x==5) || (x==7)) {
    printf ("%d is a prime number.\n", x);
    return 0;
  }
  if ((x%2 == 0) || (x%3 == 0) || (x%5 == 0) || (x%7 == 0)) {
      printf ("%d is not a prime number.\n", x);
  } else {
      printf ("%d is a prime number.\n", x);
  }
  return 0;
}
```

Prime number < 100?

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
  int x;
  printf("Type an integer x (1 < x < 100): ");
  scanf("%d", &x);
  if ((x <= 1) || (x > 99)) {
    printf ("Hey, I told you: 1 < x < 100!\n");
    return -1;  /* stop program with error code -1 */
  }

  if ((x==2) || (x==3) || (x==5) || (x==7)) {
    printf ("%d is a prime number.\n", x);
    return 0;
  }
  if ((x%2) && (x%3) && (x%5) && (x%7)) {
      printf ("%d is a prime number.\n", x);
  } else {
      printf ("%d is not a prime number.\n", x);
  }
  return 0;
}
```

BEWARE!

- The following two if-statements are NOT the same!

-
  ```
  if (1 < i  &&  i < 10) {
        printf("1 < i < 10\n");
  }
  ```

-
  ```
  if (1 < i < 10) { /* Note: 1 < i < 10 is always true! */
    printf("1 < i < 10\n");
   }
  ```

- It is better to write:

  ```
  if ((1 < i) && (i < 10)) {
     printf("1 < i < 10\n");
  }
  ```

Nested IF-statements

```
if (grade == 10) {
  printf("outstanding\n");
} else {
  if (grade == 9) {
    printf("excellent\n");
  } else {
    if (grade == 8) {
      printf("very good\n");
    } else {
      if (grade == 7) {
        printf("good\n");
      } else {
        if (grade == 6) {
          printf("satisfactory\n");
        } else {
          if (grade == 5) {
            printf("almost satisfactory\n");
          } else {
            printf("(very) unsatisfactory\n");
          }
        }
      }
    }
  }
}
```

Switch statement

```
switch (grade) {
case 10: printf("outstanding\n");
  break;
case 9: printf("excellent\n");
  break;
case 8: printf("very good\n");
  break;
case 7: printf("good\n");
  break;
case 6: printf("satisfactory\n");
  break;
case 5: printf("almost satisfactory\n");
  break;
default: printf("(very) unsatisfactory\n");
}
```

Switch statement (fall-through)

```
switch (grade) {
case 10: printf("outstanding\n");
  break;
case 9: printf("excellent\n");
  break;
case 8: printf("very ");
case 7: printf("good\n");
  break;
case 5: printf("almost ");
case 6: printf("satisfactory\n");
  break;
default: printf("(very) unsatisfactory\n");
}
```

Conditional/ternary expression

- General from:

```
condition ? expr1 : expr2
```

- Example:

```
absValue = (n < 0 ? –n : n);
```

- Much better (shorter) than:

```
if (n < 0) {
  absValue = -n;
} else {
  absValue = n;
}
```

End week 1