PROGRAMMING

CT103

Week 1b

Lecture Content

- Last lecture (Week 1a):
  - Module overview: Grading, etc.
  - Introduction to algorithms.
  - Pseudocode and flowcharts.

- This lecture (Week 1b):
  - Computer programs.
  - Data types.
  - Example C program.

COMPUTER PROGRAMS

What is a Program?

- **Definition**: A program is a set of *instructions* that are run by the *Central Processing Unit (CPU)* on a computer.

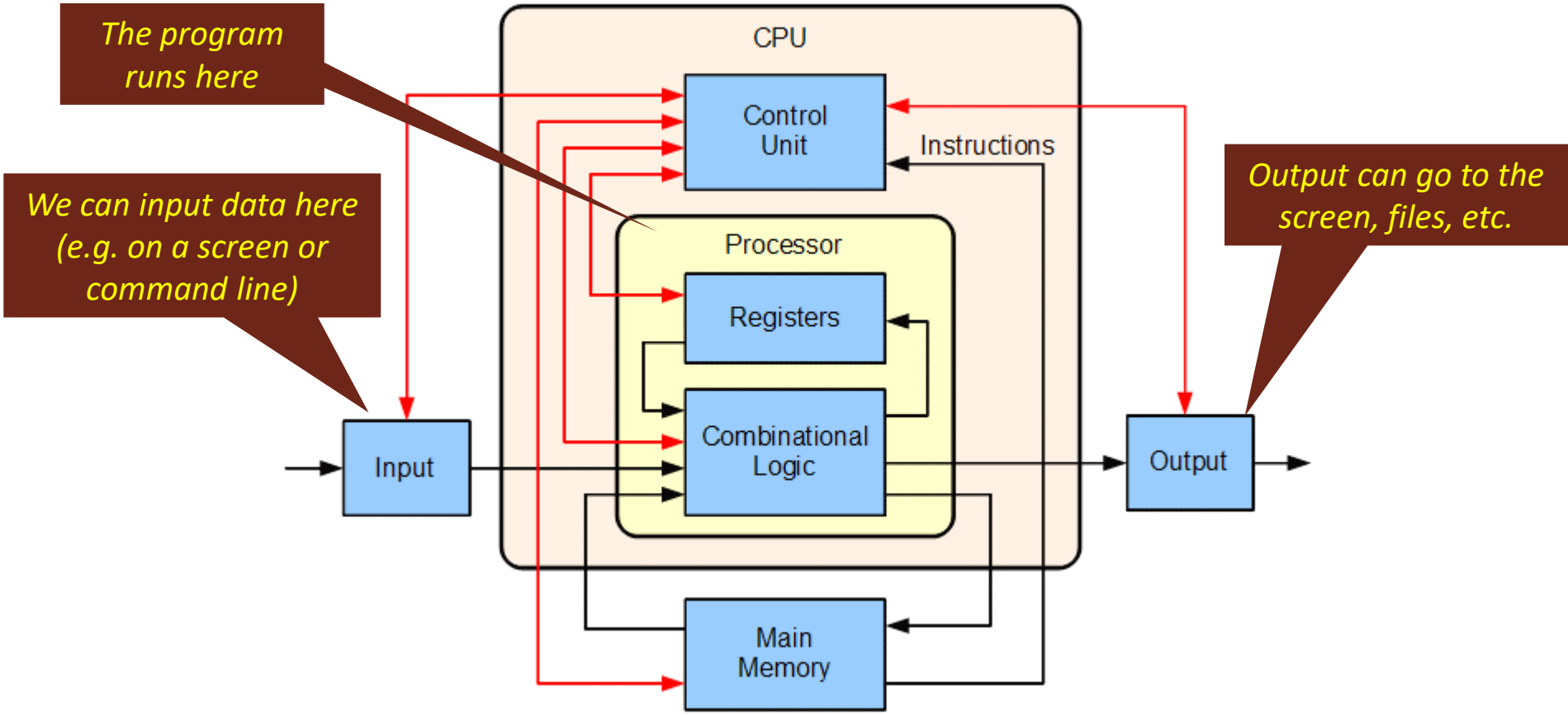
- The instructions are designed to accomplish a specific task and are written in a programming language like C.

What is a Program?

- C is what is called a *high-level language* – this has to be translated into instructions called *machine language* that the CPU can execute.

- Distinction between Program and Algorithm:
  - A program is a set of instructions that the computer executes.
  - An algorithm is a series of steps to complete a task.
  - A program contains the algorithm.
  - Algorithm is the logic, program is the implementation.

Where our Program Runs



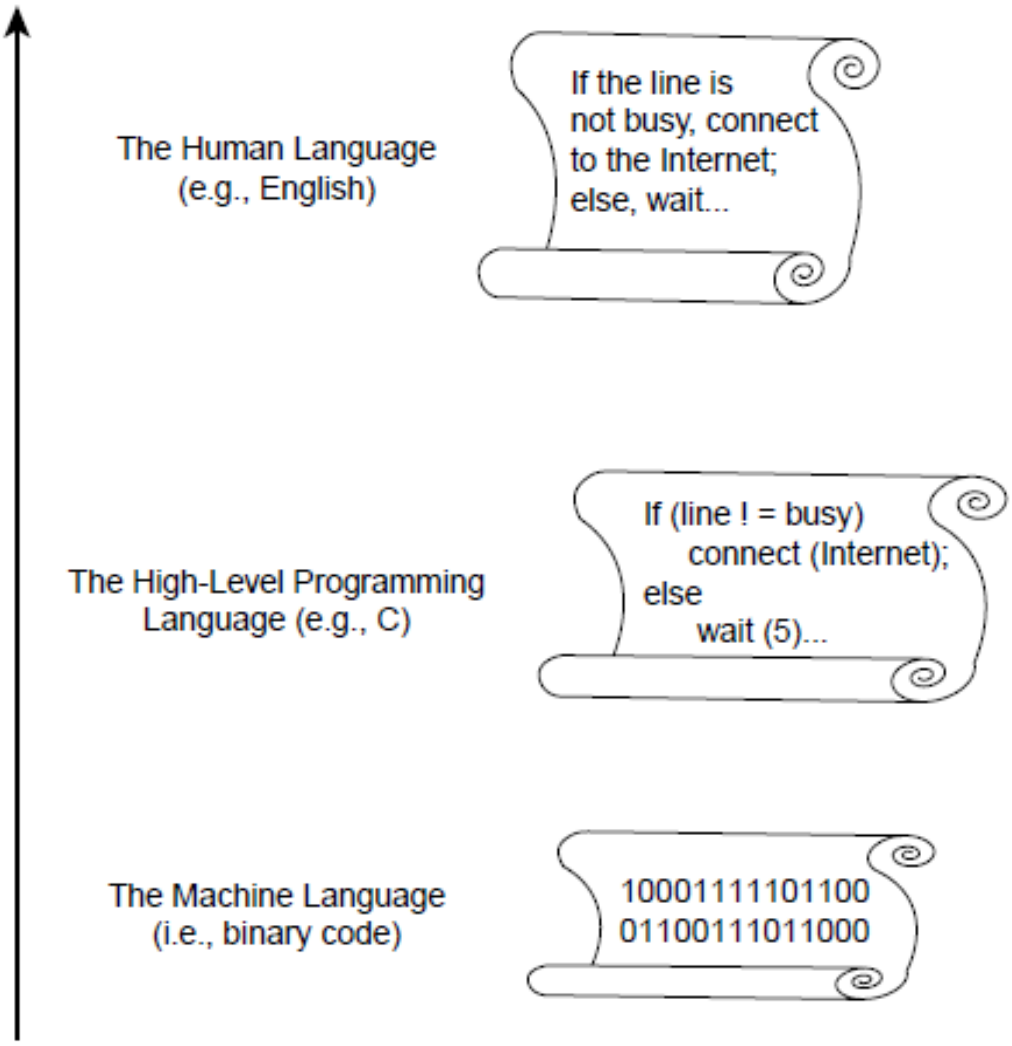
CPU and RAM

Central Processing Unit (CPU)

Random Access Memory (RAM)

CPU Runs What?

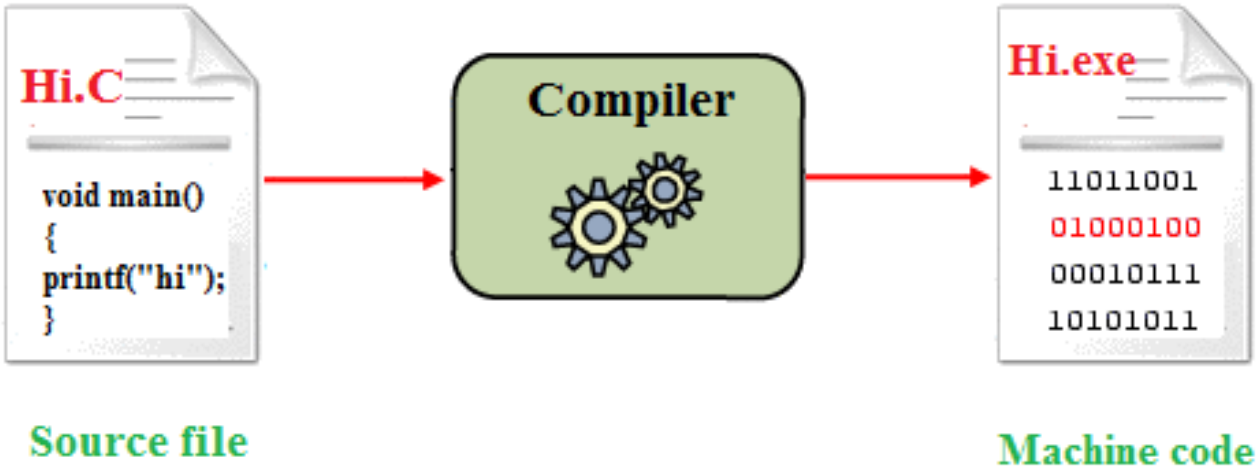
- Each CPU uses a specific set of instructions, called ***machine language***. We write our programs in a **higher level language** which is then translated into a machine-specific set of instructions for execution by the CPU



Sams Teach Yourself C in 24 Hours ©2000 by Sams Publishing

Compiler

- If the CPU understands machine language (1s and 0s), how do we convert our C program to machine language?
  - Answer: The compiler will do this for you!
- The compiler will convert your C program source code (.c file) into binary code for the CPU to understand.



Programming Software

- Applications used to write software:
  - Assist in writing program in high level language (e.g. C)
  - Compile it into machine language
  - Link various bits of machine code together to create an application
  - Run, test and debug the application
- Typically also called IDE (Integrated Development Environment), such as **MS Visual Studio** or NetBeans

Writing Programs

- We use an ***editor*** to write the program (the ***source code***), and then a ***compiler*** to compile it.

- A compiler turns the program into ***machine-language*** instructions that the computer can understand.

DATA & VARIABLES

Variables

- How we temporarily store data that we are using in our programs
- They often represent some *real-world* piece of data, e.g.
  - salary, temperature, interest rate
- In most programming languages, including C, we have to decide on the type of variable most suitable for the data we want to store and manipulate
- Variable examples in C:
  - float salary;
  - float temperate;
  - int age;
  - char exam_grade;

C Number types

- There are actually different kinds of numbers:
  - Integers (no decimal point)
    - E.g. 10 54 0 -121
  - Floating-point or real (with decimal point)
    - E.g. 4343.34 0.0 0.123234 -34.223
  - The choice of integer or floating-point depends on what it represents
    - Age (integer), No. of people in family (integer), Interest Rate (floating-point), price of litre of petrol (floating-point)

Kinds of Data

- So we can see that we need different variable types, or data types, to hold information
- The basic set of C data types is:
  - **int** - this holds an integer
    e.g. 10 21 456 -6899
  - **float** – holds a floating point number
    e.g. 125.467
  - **double** – holds a very big floating point number
    e.g. up to 1.797e+308
  - **char** – holds a character
    e.g. ‘A’ ‘c’ ‘%’
  - Also **strings** – holds multiple characters
    e.g. ‘hello’

Modifiers

- **Short,** i.e. smaller (less memory)
- **Long**, i.e. larger (more memory)
- **Signed**, i.e. positive or negative
- **Unsigned**, i.e. non negative
- The amount of storage used for each data type (+ modifier) is not set in stone
- ANSI has the following rules:

  short int <= int <= long int

  float <= double <= long double

Size (bytes)

- Actual space used to store numbers can vary between machines and operating systems, but in general:

Data Type	Memory (Bytes)	Min Value	Max Value
short int	2	-32768	32768
unsigned short int	2	0	65,535
unsigned int	4	0	4294967295
int	4	-2147483648	2147483648
unsigned char	1	0	255
float	4		
double	8		
long double	12		

Characters

- A *character* is any single character that your computer can represent – usually there are 256 of them

- We usually use the 128 most common (called the *Standard* ASCII character set)

Back to Characters

- The following are all characters:

  A a 4 % ^ . Q + =] #

- A group of multiple characters is called a *string* e.g.

  “I love Programming!”

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

Source: www.LookupTables.com

Functions

- A function is a piece of self-contained code that performs a task
- For example, to print out the text “Hello”, we can use the standard C function *printf()*
  - printf (“Hello”);
- To read an integer input from the keyboard, we could use:
  - scanf_s (“%d”, &age);

- We will learn more about functions later in the course!

PROGRAM RECAP

Designing your Program

- The most basic way of describing what should happen is to just write it down
- The easiest way of doing this is to use *Structured English*
- This means using keywords like IF, THEN, ELSE, DO, to express what should happen
- Another common way is to use a Flowchart

Sequence

- Actions which take place one after the other

Find a teapot

Put in the tea

Pour in boiling water

IF-THEN-ELSE

- Used where you need to decide on what action to take

IF condition A

THEN action B

ELSE action C

ENDIF

“IF” Example

IF you like tea

THEN drink tea

ELSE drink coffee

ENDIF

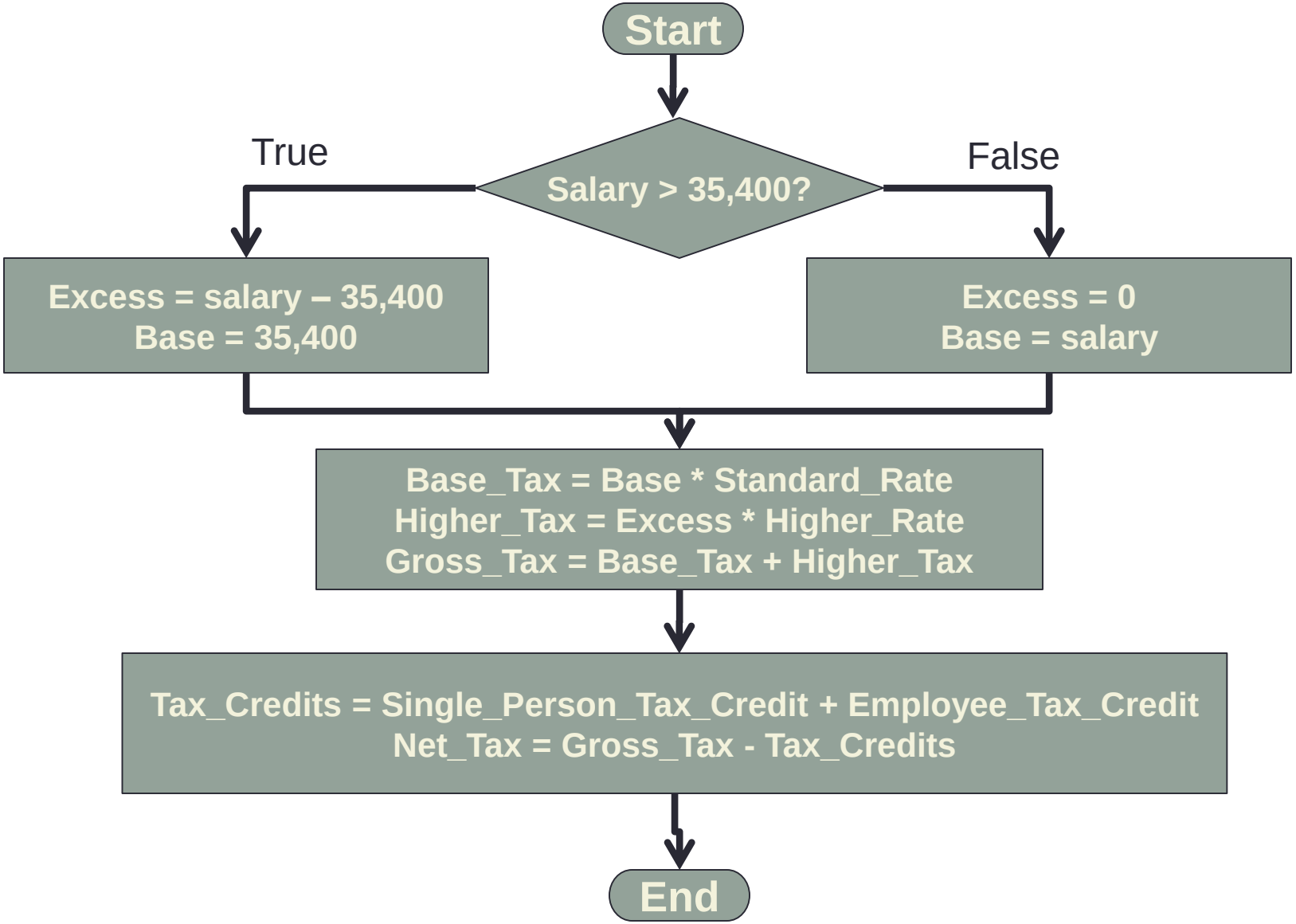
More Realistic IF Example

```
IF customer_order_total  > €400
  THEN
  IF days customer_balance is due > 60 days
          THEN
                  hold the customer_order
                  send reminder letter
          ELSE
                  process the customer order
  END-IF
ELSE
  process the customer order
END-IF
```

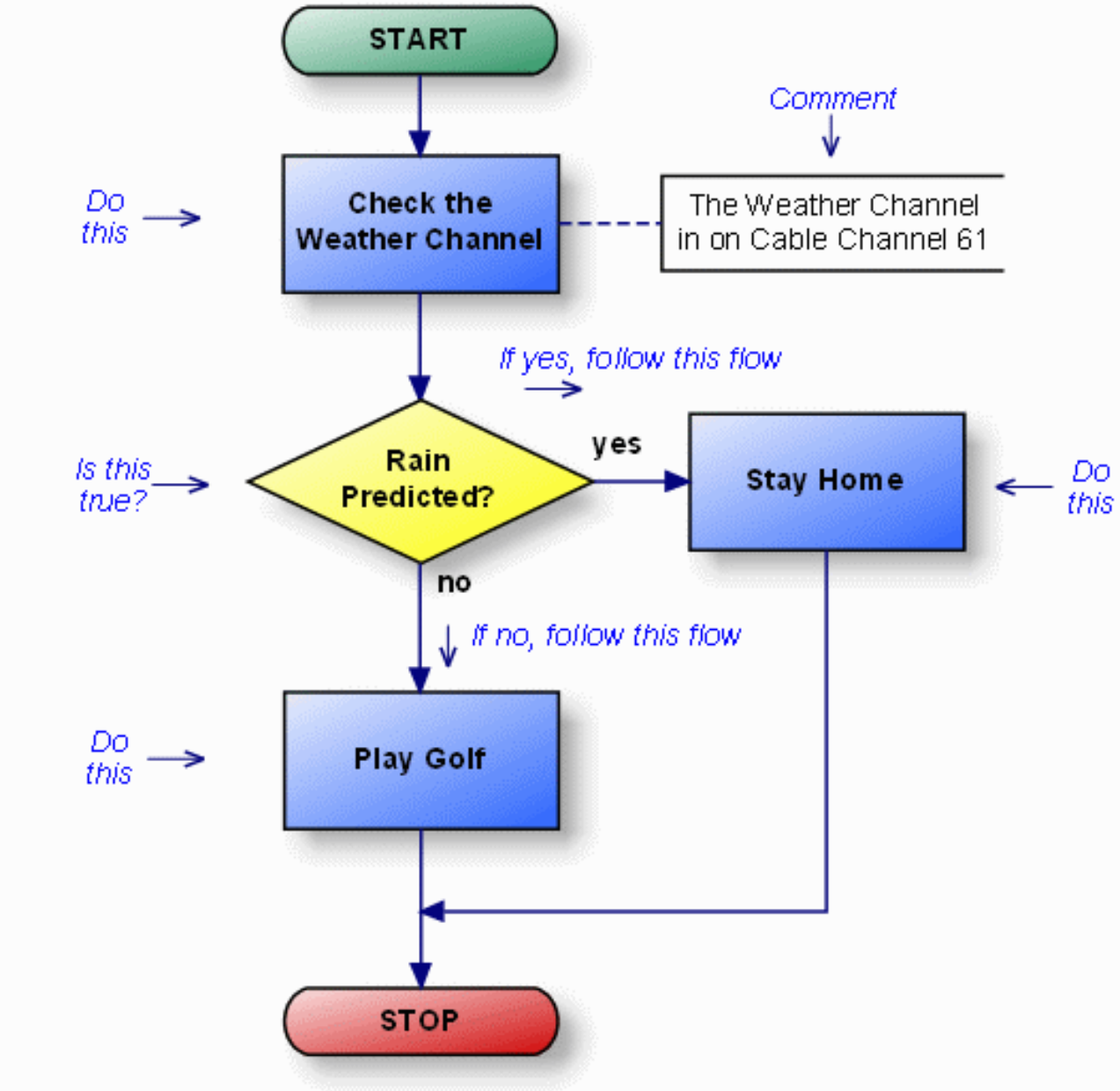
Structured English

```
Read in salary
IF salary > 35,400 THEN
   Excess = salary - 35,400
   BASE = 35,400
ELSE
   Excess = 0
   BASE = salary
ENDIF
Base_Tax = Base * Standard_Rate
Higher_Tax = Excess * Higher_Rate
Gross_Tax = Base_Tax + Higher_Tax
Tax_Credits = Single_Person_Tax_Credit + Employee_Tax_Credit
Net_Tax = Gross_Tax - Tax_Credits
```

Flowchart


Start
Salary > 35,400?
True
False
Excess = salary – 35,400
Base = 35,400
Excess = 0
Base = salary
Base_Tax = Base * Standard_Rate
Higher_Tax = Excess * Higher_Rate
Gross_Tax = Base_Tax + Higher_Tax
Tax_Credits = Single_Person_Tax_Credit + Employee_Tax_Credit
Net_Tax = Gross_Tax - Tax_Credits
End

Workflow


START
Comment
Do this
Check the Weather Channel
The Weather Channel in on Cable Channel 61
If yes, follow this flow
Is this true?
Rain Predicted?
yes
Stay Home
Do this
no
If no, follow this flow
Do this
Play Golf
STOP

PROGRAM EXAMPLES

Worked Through Example

- Problem Description:
  - Write a program that reads in an exam mark and outputs “Passed” if the mark is 60 or more. Otherwise print out “Failed”.

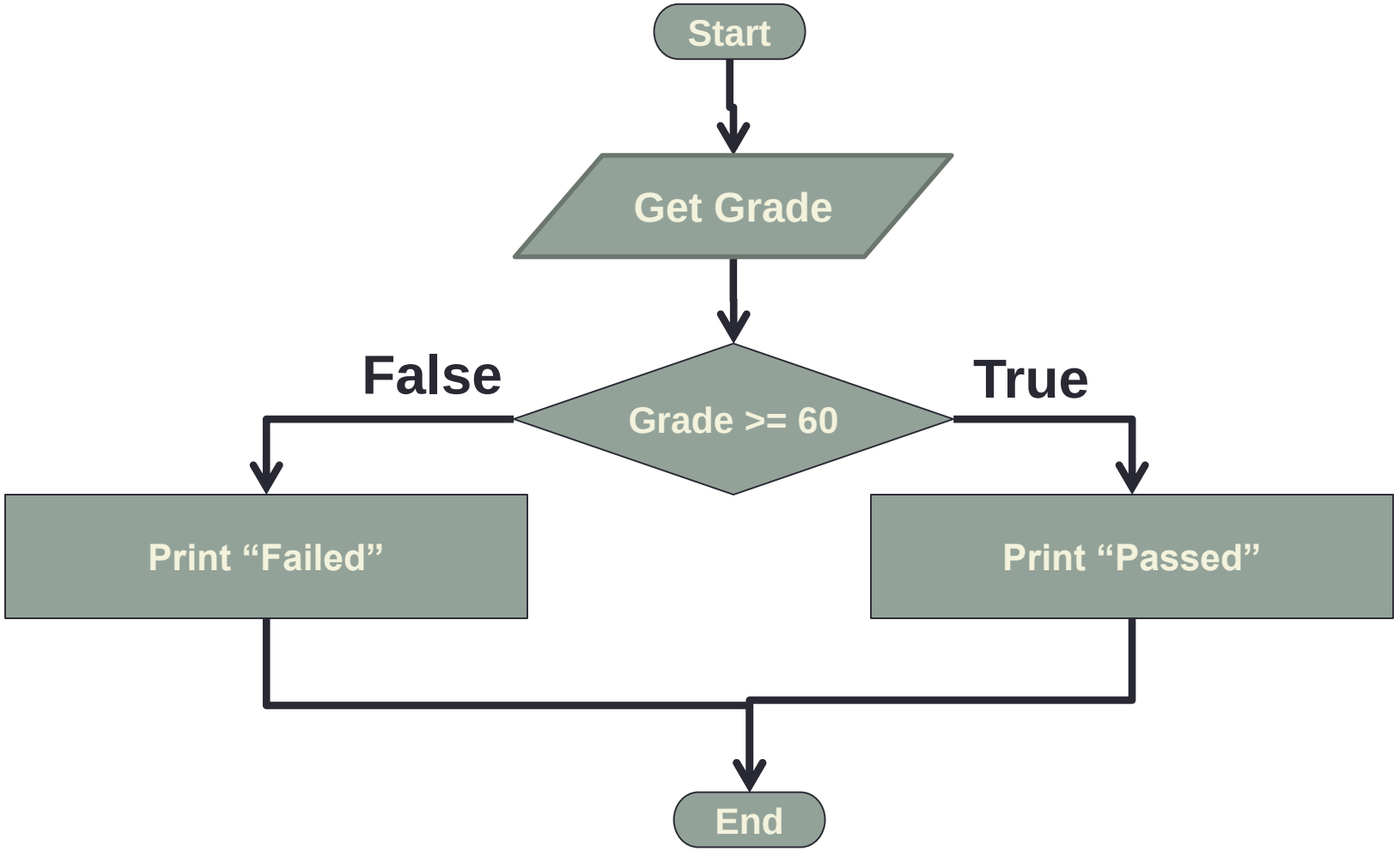
Pseudocode

Get exam grade

If grade is greater than or equal to 60
  Print “Passed”

else
  Print “Failed”

Flowchart


Start
Get Grade
Grade >= 60
False
True
Print “Failed”
Print “Passed”
End

```c
#include <stdio.h>
void main() {
   int grade = 0;

   printf("Enter grade: ");
   scanf_s("%d",&grade);

   if (grade >= 60){
      printf("Passed \n");
   }
   else{
      printf("Failed \n");
   }
}
```

Code in Visual Studio

- Here I will go through some C code