scanf (continued) and Data Types in C

ESC101: Fundamentals of Computing

Nisheeth

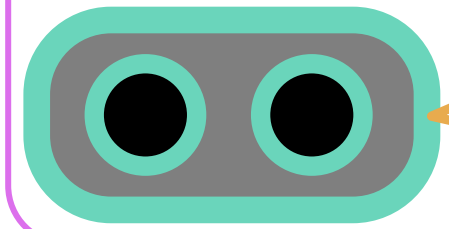
How does scanf work ?

HOW WE MUST SPEAK TO MR. COMPILER

```
scanf("%d%d", &a, &b);
```

Format string

Format string tells me **how** **you** will write things, and then I am told **where** to store what I have read



HOW WE USUALLY SPEAK TO A HUMAN

Please read one integer. Ignore all whitespace (spaces,tabs,newlines) after that till I write another integer. Read that second integer too.

Store value of the first integer in a and value of second integer in b.

Remember Mr. C likes to be told beforehand what all we are going to ask him to do!

Scanf follows this exact same rule while telling Mr. C how to read

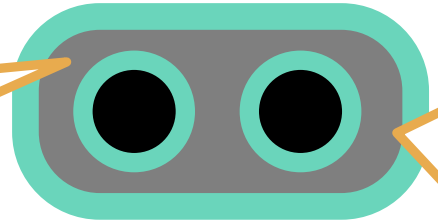
How does scanf work ?

Be a bit careful since Mr C is a bit careless in this matter

He trea haracters the same when integer

scanf w ing

My advice to you is to take input one at a time in the beginning ☺ Try out acrobatics in free time



Hmm … you are going to write the English word Hello followed by space followed by an integer. I will store the value of that integer in a

```
scanf("Hello %d",&a);
```

Use printf to print and scanf to read

Try out what happens with the following

```
scanf("%d %d",&a,&b);      scanf("%dHello%d",&a,&b);
scanf("%d,%d",&a,&b);      scanf("\"%d%d\"",&a,&b);
scanf("%d\n%d",&a,&b);     scanf("%d\t%d",&a,&b);
```

Commenting

Very important programming practice

```
int main(){
    int a; // My first int
    int b; // The other int
    // Assign them values
    a = 5, b = 4;
    a + b;
    return 0;
}
```

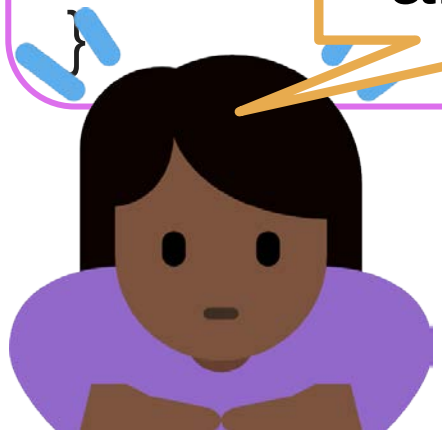
```
int main(){
    int a; /* My first int */
    int b; /* The other int */
    /* Assign them values */
    a = 5, b = 4;
    a + b;
    return 0;
}
```

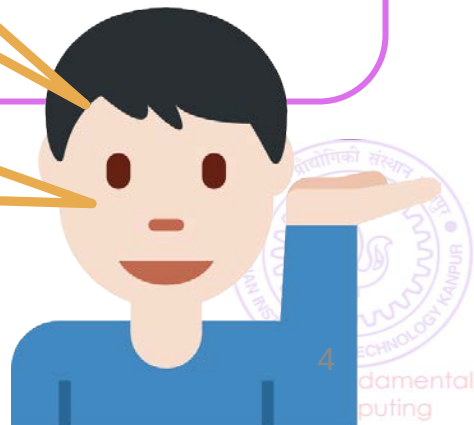
```
int main(){
    int a; // My first int
    int b; // The other int
    /* Assign them values */
    a = 5, b = 4;
    a + b;
    return 0;
}
```

So I can mix and match?

Yes. In fact /* */ is used to comment several lines at once – shortcut!

Just be a bit careful. Some compilers don't understand // comments





More on Comments

Use comments to describe why you defined each variable and what each step of your code is doing

- You will thank yourself for doing this when you are looking at your own code before the end sem exams
- Your team members in your company/research group will also thank you

Multiline comments very handy. No need to write // on every line

```
int main(){
    int a; // My first int
    int b; // The other int
    // Assign them values
    // so that I can add
    // them later on
    a = 5, b = 4;
    a + b;
    return 0;
}
```

A Useful Tip While Problem-Solving

Comments can be also used to identify where is error

Mr C will tell you (compile) where he thinks the error is

pointing out lines also help identify error

Error!

Okay!

Okay!

```
int main(){
    int a, b;
    c = a + b;
    a = 5;
    b = 4;
    return 0;
}
```

```
int main(){
    int a, b;
    // c = a + b;
    a = 5;
    b = 4;
    return 0;
}
```

```
int main(){
    int a, b;
    c = a + b;
    a = 5;
    b = 4;
    return 0;
}
```

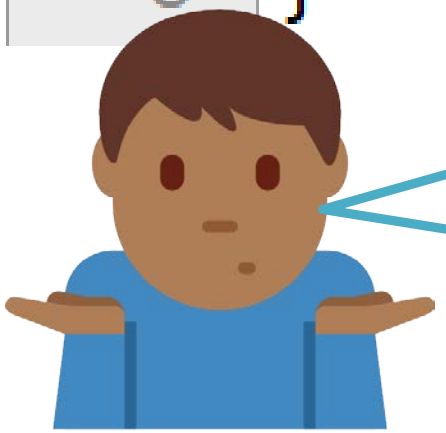
Aha! I forgot to declare c

A Useful Tip While Solving Problems

```
#include<stdio.h>
int main(){
    int x = 3;
    int result;
    result = 2/3*x*x*x + 2*x*x      x;
    printf("The area under the    rve is %d",result);
    return 0;
}
```

Print your solutions to each one of these pieces to see where going wrong

Try breaking up the problem into smaller pieces



I have no idea what is going wrong here!

A Useful Tip While Solving Problems

```
#include<stdio.h>
int main(){
    int x = 3;
    int result;
    result = 2/3*x*x*x + 2*x*x + 9*x;
    printf("The area under the curve is %d",result);
    return 0;
}
```

Equals 0

A Useful Tip While Solving Problems

```c
#include<stdio.h>
int main(){
    int x = 3;
    int result;
    result = 2/3*x*x*x + 2*x*x + 9*x;
    printf("The area under the curve is %d",result);
    return 0;
}
```

Replace this part by (2*x*x*x)/3

Basic Data Types in C

- **Int: %d** specifier
  - Integers like 156, -3, etc
- **float** (short form of "floating point number") and **double: %f** specifier
  - Real numbers like 3.14, 2.0, -1.3, etc
  - double is like float but has larger range
- **char** (short form of "character"): **%c** specifier
  - Single letter (a-z or A-Z), single digit, or single special character
  - A char is always enclosed in inverted single commas
  - Some examples: 'a', 'A', '2', '$', '='
- These basic data types can also be used with a modifier
  - Modifiers change the normal behaviour of a data type (e.g., its range of values) and memory storage space required (more on next slides)

Type Modifiers in C

- **signed** (used with int, float/double, char)
  - signed means the data type can have positive and negative values
  - int, float/double, char are signed by default (no need to write 'signed')
- **unsigned** (used with int, char)
  - unsigned means the data type can have only take positive values
- **short** (used with int)
  - short means it uses only half of the memory size of a normal int
- **long** (used with int)
  - long means it uses twice the memory size of a normal int
  - Can store a larger range of values of that type

Various C Data Types without/with Modifiers

int (signed int)
%d

unsigned int
%u

short int (short)
%d

long int (long)
%ld

short unsigned
%u

long unsigned
%lu

Yes, multiple modifiers also allowed

float
%f

double
%lf

long double
%Lf

char
%c

unsigned char
%u

int

- Can store integers between -2,147,483,648 and 2,147,483,647

```
#include <stdio.h>
int main(){
int a;
scanf("%d", &a);
printf("My first int %d", a);
return 0;
}
```

%d

a

Range: -2^31 to (2^31)-1

signed int uses 32 bits
(4 bytes, 8 bits = 1 byte)
on recent compilers)

Integer arithmetic applies to integers +, -, /, *, %, ()

Have worked with them a lot so far

Printing well-formatted outputs using printf

- When printing an int value, place a number between % and d (say %5d) which will specify number of columns to use for displaying that value

```
int x = 2345, y=123;
printf("%d\n",x); //Usual (and left aligned)

printf("%6d\n",x); //Display using 6 columns (right aligned)

printf("%6d\n",y);

printf("%2d\n",x); //Less columns than digits, same as %d
```

Output

```
2345
  2345
   123
2345
```

Note: So far, we have only seen how to print integers.
We will see how to print other types of variables later today

long int (usually written just long)

- Really long – can store integers between
- -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807

```
#include <stdio.h>
int main(){
long a; //long int also
scanf("%ld", &a);
printf("My first long int %ld", a);
return 0;
}
```

a

%ld

Range: -2^63 to (2^63)-1

long int uses 64 bits on recent compilers

Integer arithmetic applies to long int as well +, -, /, *, %, ()

Try them out on Prutor

How does long work with int int + long, int * long?

Will see in next class…

float

- int, long allow us to store, do math formu
- float allows us to store, do math formulae with reals

float uses 32 bits (4 bytes).
Why this range for double?
Will see reason later

```
#include <stdio.h>
int main(){
float a;
scanf("%f", &a);
printf("My first real %f", a);
return 0;
}
```

a

%f

Very large range ± 3.4e+38

Arithmetic operations apply to float as well +, -, /, *, ()

Try them out on Prutor

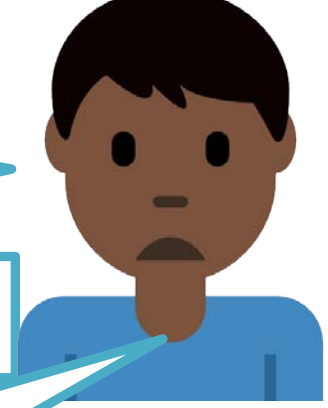
Did you ever do remainders with rea numbers in school?

I remember. Remainders make sense for integers, not for real numbers

What happened to ?

printf revisit: Printing of float

Too many decimal digits being printed. Can I just print one or two?



Correct!

Value of a = 123.46

Value of a = 1.23e+02

Both %f and %lf work for float and double

Yes. Use %0.2f to print 2 decimal places

Oh right. The usual rules of rounding apply here too. 1.5644 will become 1.56 if rounded to 2 places but 1.5654 will become 1.57

%e if want answer in exponent

```
#include <stdio.h>
int main()
    double a = 123.4567;
```

Sure. Just like I did it for integer display. See next slide ☺

```
#include
int ma
    double a = 123.4567;
    printf("V
    return 0
}
```

Be careful, I am rounding while giving answer correct to 2 decimal places
123.4567 → 123.46
1.234567 → 1.23

Great. So can you also help me control the format in which float/double is displayed on screen?

printf revisit: Controlled printing of float/double

- Already saw how to use printf for well-formatted int display
- Can also control how to display a float/double using printf
- Can do it using "%a.bf" specifier where a and b are numbers
  - Here a is the total field width (number of columns) in which the float will be displayed, b is the number of digits printed after decimal

```
float pi = 3.141592;
printf("%f\n",pi); //Usual

printf("%6.2f\n", pi); //2 decimal

printf("%0.4f\n",pi); //4 decimal
                      // Note rounding off!
```

Output

```
3.141592
  3.14
3.1416
```

double

- Double can also handle real numbers but very large ones
- Similar relation to float as long has to int

```
#include <stdio.h>
int main(){
double a;
scanf("%f", &a);
printf("My first real %f", a);
return 0;
}
```

%lf works too!

Very large range ± 1.79e+308

Arithmetic operations apply to double as well +, -, /, *, ()

There is something called long double as well

Use %Lf to work with long doubles

Try these out on Prutor

char

- Basically, a char is a symbol
- Internally stored as an integer between -128 and 127 (if signed char) or between 0 and 255 (if unsigned char)

```
#include <stdio.h>
int main(){
char a = 'p';
printf("My first char %c\n", a);
printf("ASCII value of %c is %d",a,a);
return 0;
}
```

%c

This Will print the ASCII value (integer) of this character

'p'

a

Char constants enclosed in ' '

Integer arithmetic applies to char as well +, -, /, *, %, ()

Case sensitive 'a', 'A' different

Various usages (e.g., in arrays of characters – strings), will see more later

ASCII TABLE

American Standard Code for Information Interchange

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

\t

\n

Image courtesy wikipedia.org

ASCII Table with Extended Characters

ASCII control characters		
00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)
11	VT	(Vertical Tab)
12	FF	(Form feed)
13	CR	(Carriage return)
14	SO	(Shift Out)
15	SI	(Shift In)
16	DLE	(Data link escape)
17	DC1	(Device control 1)
18	DC2	(Device control 2)
19	DC3	(Device control 3)
20	DC4	(Device control 4)
21	NAK	(Negative acknowl.)
22	SYN	(Synchronous idle)
23	ETB	(End of trans. block)
24	CAN	(Cancel)
25	EM	(End of medium)
26	SUB	(Substitute)
27	ESC	(Escape)
28	FS	(File separator)
29	GS	(Group separator)
30	RS	(Record separator)
31	US	(Unit separator)
127	DEL	(Delete)

ASCII printable characters					
32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Extended ASCII characters							
128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

IMAGE COURTESY: https://theasciicode.com.ar/

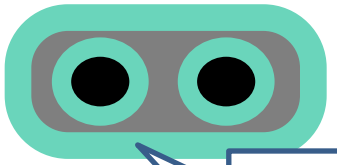
Mixing Types in C Expressions

- We can have C expression with variables/constants of several types
- Certain rules exist that decide the type of the final value computed
- Demotion and Promotion are two common rules

- int a = 2/3; // a will be 0 (no demotion/promotion)
- float a = 2/3; // a will be 0.0 (RHS is int with value 0, promoted to float with value 0.0)
- int a = 2/3.0; // a will be 0 (RHS is float with value 0.66, becomes int with value 0)
- float a = 2/3.0; // a will be 0.66 (RHS is float with value 0.66, no demotion/promotion)
- int a = 9/2; // a will be 4 (RHS is int with value 4, no demotion/promotion)
- float a = 9/2; // a will be 4.0 (RHS is int with value 4, becomes float with value 4.0)

- During demotion/promotion, the RHS value doesn't change, only the data type of the RHS value changes to the data type of LHS variable

Type Casting or Typecasting



Also remember: When assigning values, I always compute the RHS first

- Converting values of one type to other.
  - Example: int to float and float to int (also applies to other types)
- Conversion can be implicit or explicit. Typecasting is the explicit way

Automatic (compiler)

By us

```
int k =5;
float x = k;            // good implicit conversion, x gets 5.0
float y = k/10;         // poor implicit conversion, y gets 0.0
float z = ((float) k)/10; // Explicit conversion by typecasting, z gets 0.5
float z = k/10.0;       // this works too (explicit without typecasting), z gets 0.5
```

Typecasting: An Example Program

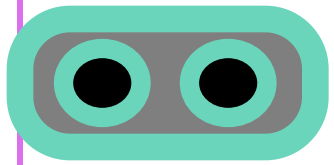
Also, typecasting just one variable on RHS is enough

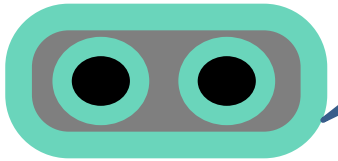
But be careful about which one you are typecasting

```
#include <stdio.h>
int main(){
   int total = 100, marks = 50;
   float percentage;
   percentage = (marks/total)*100;
   printf("%.2f",percentage);
   return 0;
}
```

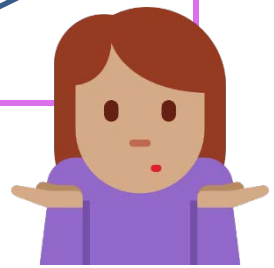
Equals 0

Several other ways also possible, e.g., proper bracketing.



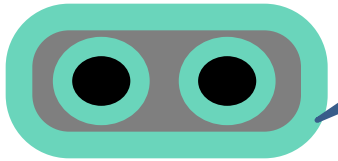


0.00

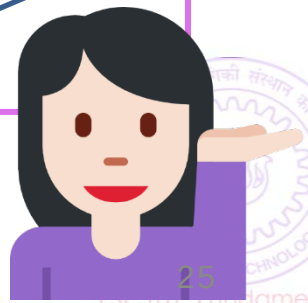


```
#include <stdio.h>
int main(){
   int total = 100, marks=50;
   float percentage;
   percentage = (float)marks/total*100;
   printf("%.2f",percentage);
   return 0;
}
```

Typecasting makes it 50.0/100 which equals 0.5



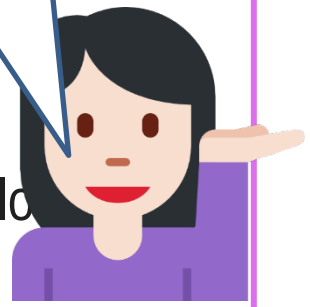
50.00

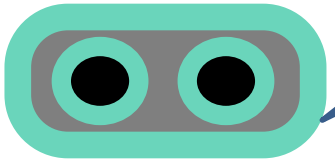


Typecasting is Nice. But Take Care..

```
#include <stdio.h>
int main(){
    float x; int y;
    x = 5.67;
    y = (int) x; // typecast (convert) flo
    printf("%d",y);
    return 0;
}
```

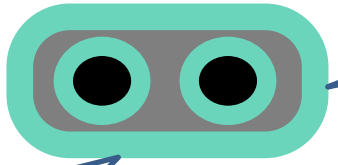
Expected conversion





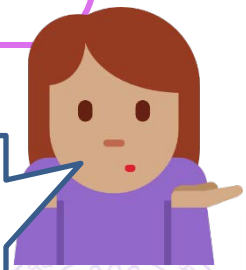
5

```
#include <stdio.h>
int main(){
    float x; int y;
    x = 1.0e50; // 10^50
    y = (int) x; // typecast (convert) float to int
    printf("%d",y);
    return 0;
}
```



-2147483648

No. 1.0e50 is too big to be cast as an int (or even long – try yourself)



Are you kidding? Unexpected!

Reverse typecasting error can happen too: Sometimes converting a smaller data type (say int) to larger data type (say float) can also give unexpected results (more on this later in the semester)