C Syntax, Primitives and Datatypes

IC 100

November 02,2022

Remember this program?

- **Program to add two integers (17 and 23).**

```
# include <stdio.h>
int main ()   {
   int a = 17;
   int b = 23;
   int c;
   c = a + b;
   printf(“Result is %d”, c);
   return 0;
}
```

The program prints the message:
Result is 40

This Class

- The alphabet of C
  - Like +
- The grammar of C
  - Like using “ ” to delimit strings
- The keywords of C
  - Like int
- C data types
- Type conversions
  - Implicit
  - ExplicitConsequences
- Consequences

Language Building Blocks

A B C D E F

G H I K L M

N O P Q R S

T V X Y Z

“Bring me tea”

The C Character Set

- The C language alphabet
  - Uppercase letters ‘A’ to ‘Z’
  - Lowercase letters ‘a’ to ‘z’
  - Digits ‘0’ to ‘9’
  - Certain special characters:

<table>
<tr><td>!</td><td>#</td><td>%</td><td>^</td><td>&</td><td>*</td><td>(</td><td>)</td></tr>
<tr><td>-</td><td>_</td><td>+</td><td>=</td><td>~</td><td>[</td><td>]</td><td>\</td></tr>
<tr><td>|</td><td>;</td><td>:</td><td>‘</td><td>“</td><td>{</td><td>}</td><td>,</td></tr>
<tr><td>.</td><td><</td><td>></td><td>/</td><td>?</td><td>blank</td><td></td><td></td></tr>
</table>

A C program should not contain anything else

Structure of a C program

- A collection of functions
- Program always starts main
- Statements are executed one by one

Context-Sensitive Rules

- ; ends statements
- What happens if I write printf(“Hi;”) ?
- How about if I want to print out “Hi” with the quotation marks?

Context-sensitive rules

- Printf prints things to console
- // printf will do nothing
- /* printf */ will do nothing
- These are special character combinations indicating programmer comments
- Best way to learn the rules
  - Play the game
  - Don’t be afraid

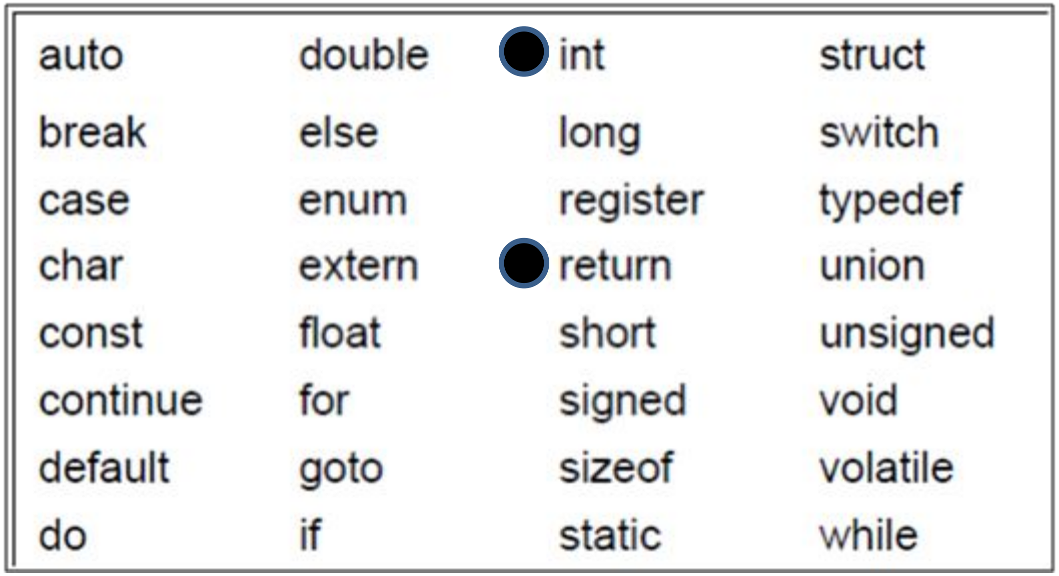
Words

- Made of alphabets
- Used to convey meaning
- English words have fixed meanings
- C *keywords* have fixed meanings
- All other C words (identifiers) have variable meanings
  - They take the meaning you want to give them

C Keywords

Used by the C language, cannot be used as variable names

● Seen already



auto	double	● int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	● return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

These 32 key words mean the same across every C compiler.

Some compilers reserve a few extra keywords, but those are less important

Read-only variables

- Variables whose values can be initialized during declaration, but cannot be changed after that
- Declared by putting the const keyword in front o the declaration
- Storage allocated just like any variable
- Used for variables whose values need not be changed
  - Prevents accidental change of the value

Correct

```
void main() {
   const int LIMIT = 10;
   int n;
   scanf(“%d”, &n);
   if (n > LIMIT)
       printf(“Out of limit”);
}
```

Incorrect: Limit changed

```
void main() {
   const int Limit = 10;
   int n;
   scanf(“%d”, &n);
   Limit = Limit + n;
   printf(“New  limit is %d”, Limit
}
```

Constants

- Integer constants
  - Consists of a sequence of digits, with possibly a plus or a minus sign before it
  - Embedded spaces, commas and non-digit characters are not permitted between digits
- Floating point constants
- Two different notations:
  - Decimal notation: 25.0, 0.0034, .84, -2.234
  - Exponential (scientific) notation

    3.45e23, 0.123e-12, 123e2

Contd.

- Character constants
  - Contains a single character enclosed within a pair of single quote marks.
  - Examples :: ‘2’, ‘+’, ‘Z’
- Some special backslash characters

‘\n’	new line
‘\t’	horizontal tab
‘\’’	single quote
‘\”’	double quote
‘\\’	backslash
‘\0’	null

Variables

- Very important concept for programming
- An entity that has a value and is known to the program by a name
- Can store any temporary result while executing a program
- Can have only one value assigned to it at any giver time during the execution of the program
- The value of a variable can be changed during the execution of the program

Contd.

- Variables stored in memory
- Remember that memory is a list of storage locations, each having a unique address
- A variable is like a bin
  - The contents of the bin is the value of the variable
  - The variable name is used to refer to the value of the variable
  - A variable is mapped to a location of the memory, called its address

Variable Names

- Sequence of letters and digits
- First character must be a letter or ‘_’
- No special characters other than ‘_’
- No blank in between
- Names are case-sensitive (max and Max are two different names)
- Examples of valid names:
  - i   rank1   MAX   max   Min   class_rank
- Examples of invalid names:
  - a’s   fact rec   2sqroot   class,rank

More Valid and Invalid Identifiers

- Valid identifiers

  X

  abc

  simple_interest

  a123

  LIST

  stud_name

  Empl_1

  Empl_2

  avg_empl_salary

- Invalid identifiers

  10abc

  my-name

  “hello”

  simple interest

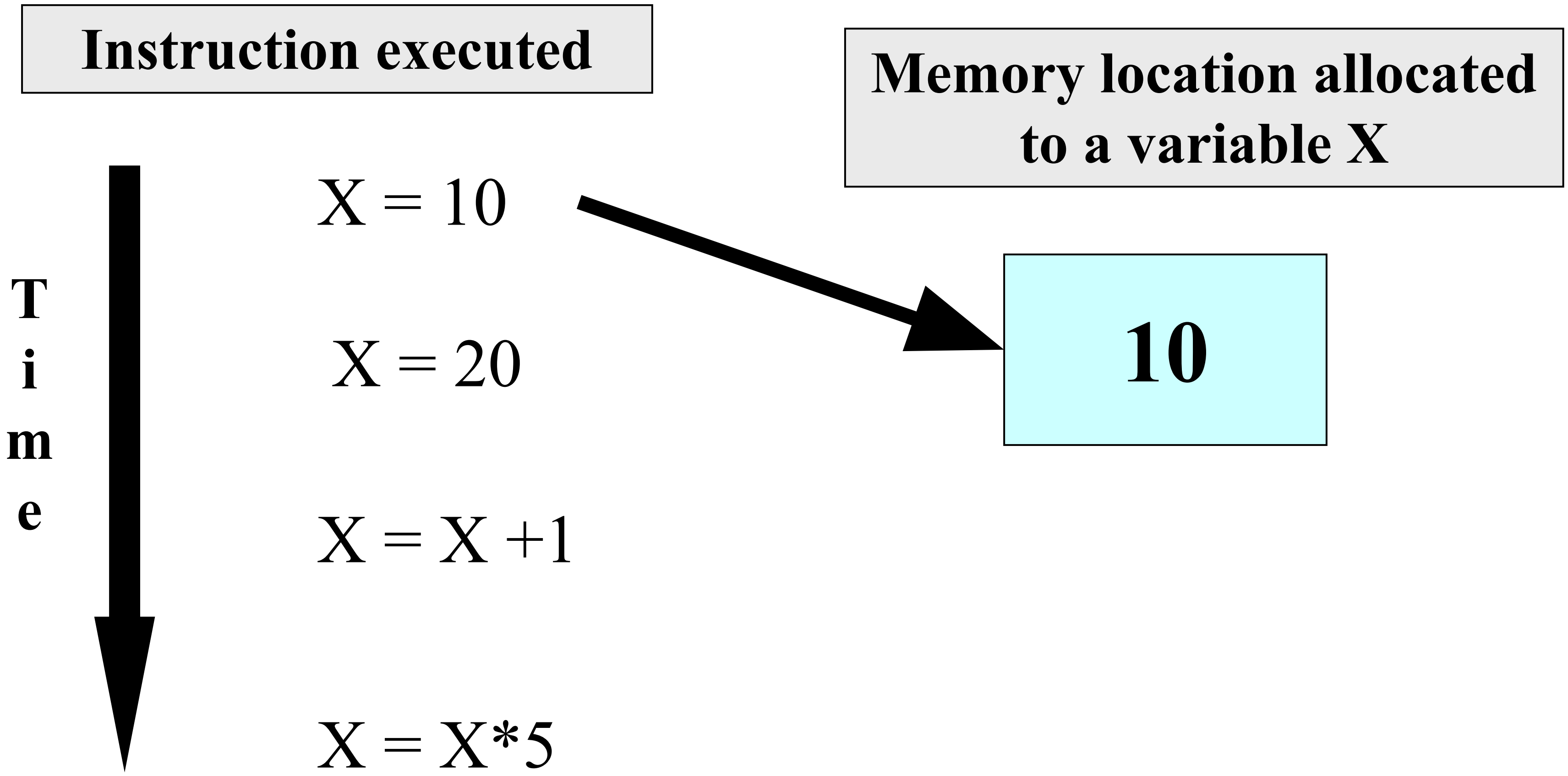
  (area)

  %rate

Example

```
#include <stdio.h>
void main( )
{
    int x;
    int y;
    x=1;
    y=3;
    printf("x = %d, y= %d\n", x, y);
}
```

Variables in Memory



Instruction executed

Memory location allocated to a variable X

Time

X = 10

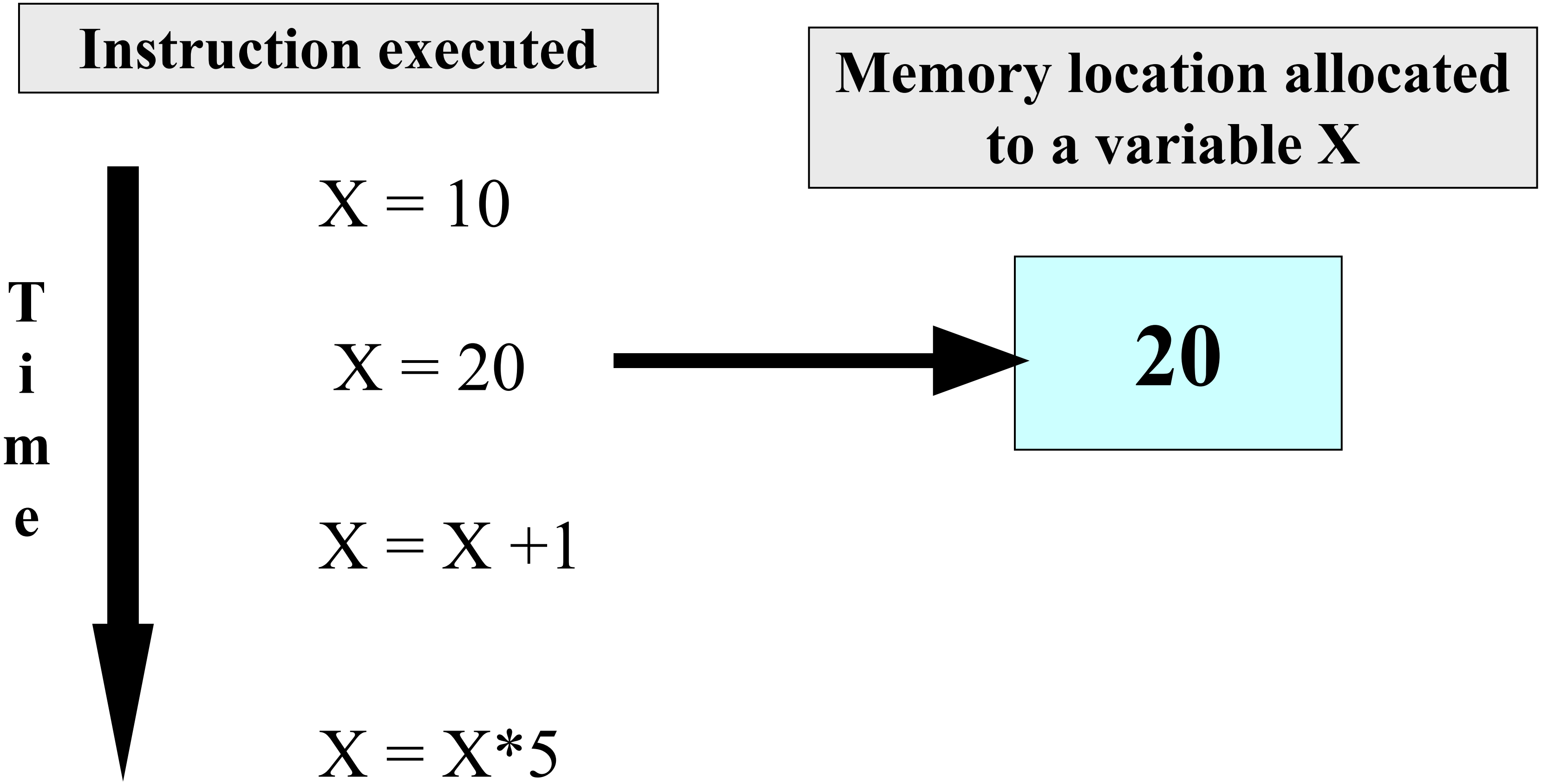
X = 20

X = X +1

X = X*5

10

Variables in Memory



Instruction executed

Memory location allocated to a variable X

T i m e

X = 10

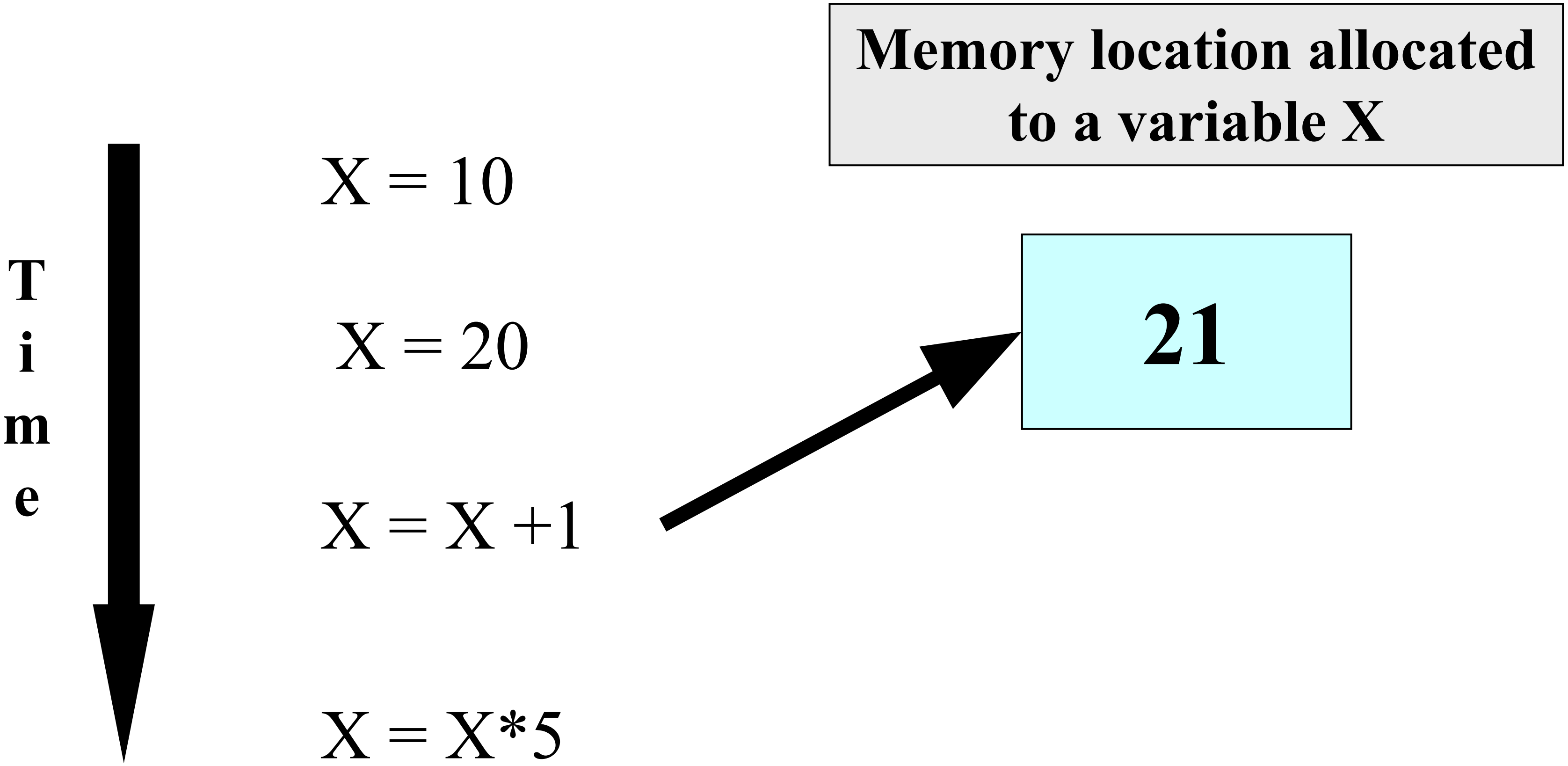
X = 20

X = X +1

X = X*5

20

Variables in Memory



Memory location allocated to a variable X

T
i
m
e

X = 10

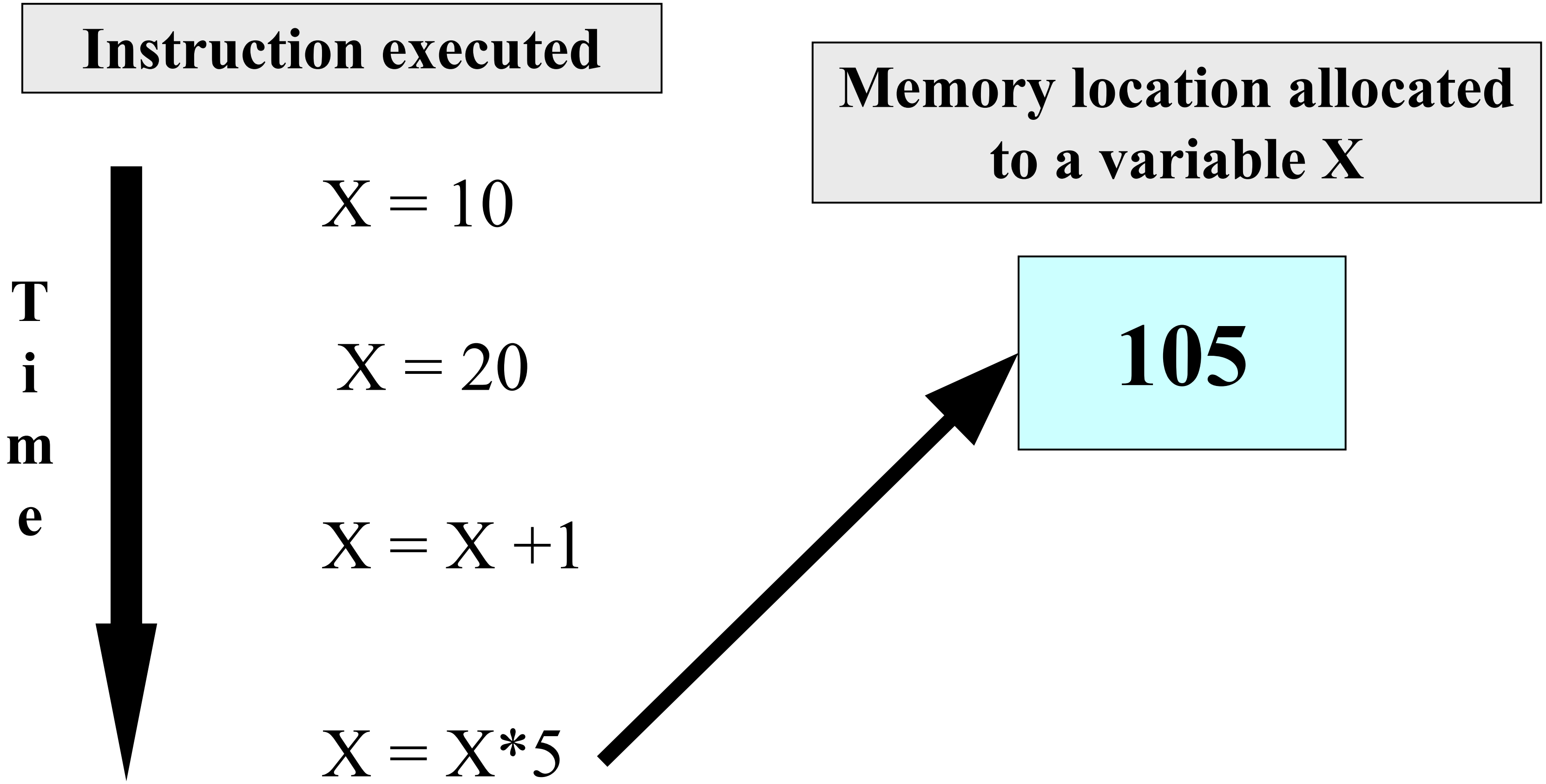
X = 20

X = X +1

X = X*5

21

Variables in Memory



Instruction executed

Memory location allocated to a variable X

T i m e

X = 10

X = 20

X = X +1

X = X*5

105

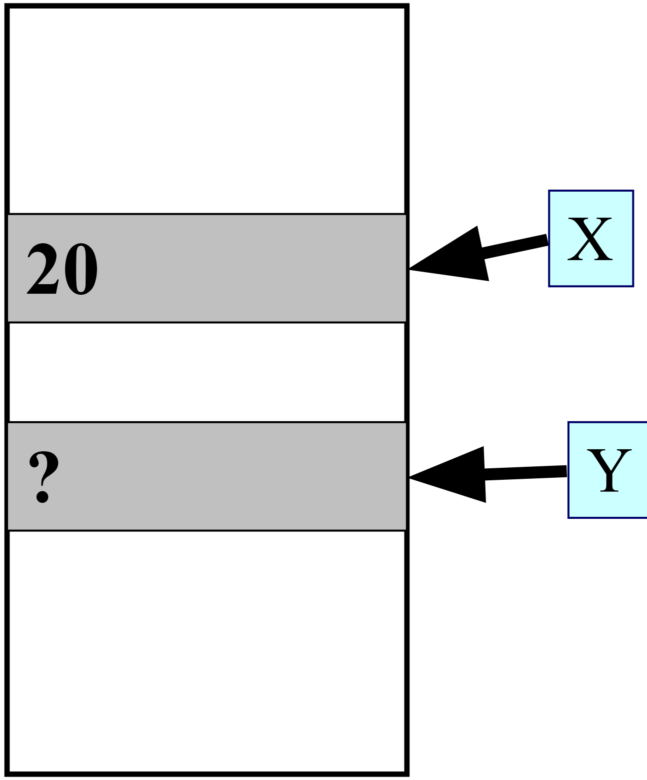
Variables (contd.)

```
X = 20

Y=15

X = Y+3

 Y=X/6
```



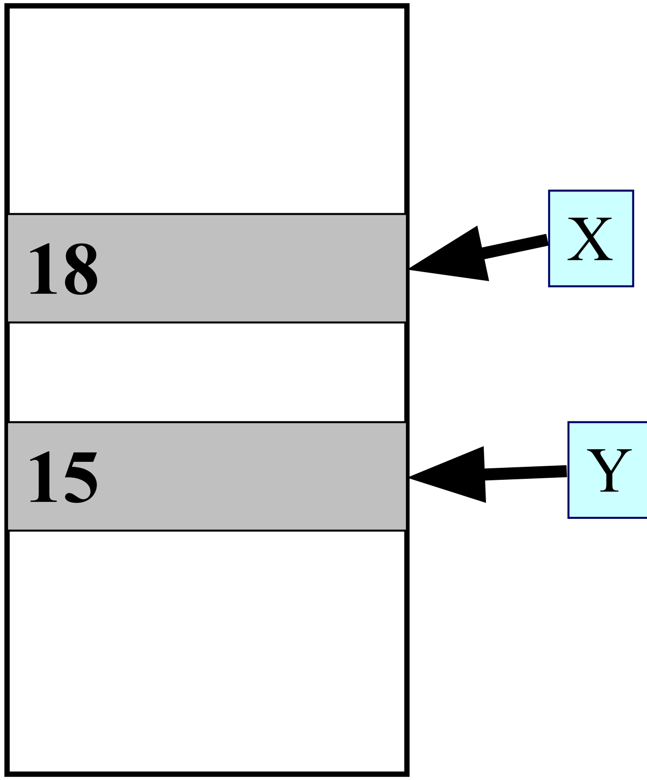
Variables (contd.)

```
X = 20

Y=15

X = Y+3

Y=X/6
```



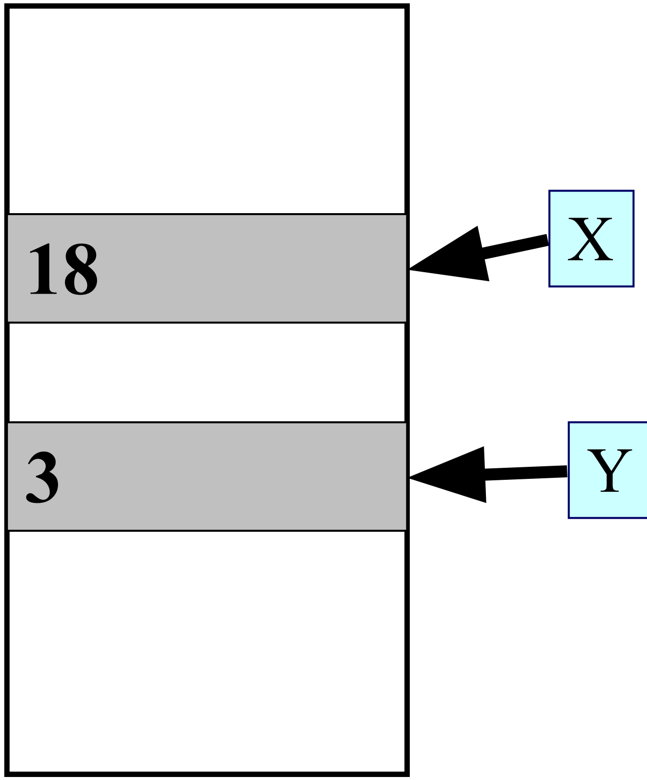
Variables (contd.)

X = 20

Y=15

X = Y+3

Y=X/6



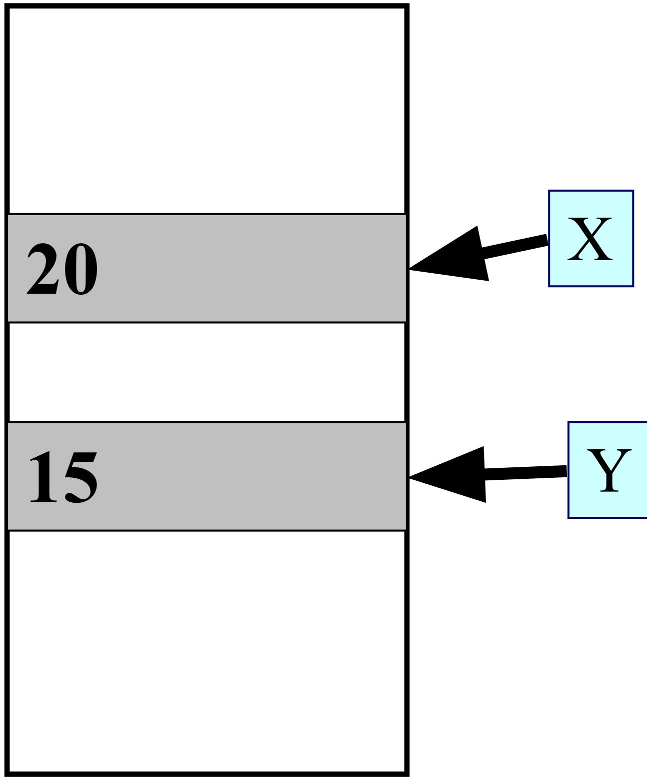
Variables (contd.)

X = 20

Y=15

X = Y+3

Y=X/6



int

- Computers store data in binary code
  - A 0 or 1 is a bit
  - 8 bits make a byte
  - 2/4/8 bytes make a word (depending on architecture)
- The keyword *int* asks the computer to assign one *word* of memory to store an integer value
  - int a = 34; 0000 0000 | 0010 0010
- How many integers can you store using N bits?
- Can only use *int* to store integers in a limited range
  - If you exceed the range, you will get a compilation error

return

- You will understand this better if/when you learn about OS
- For this course, return is what you use to end execution of the current set of instructions
  - It returns the value 0 to indicate successful execution of instructions
  - Unsuccessful execution returns non-zero value
  - In C, using void main and no return statement terminates the program abnormally
    - Best avoided in this course

Do You Now Understand This Program?

- **Program to add two integers (17 and 23).**

```
#include <stdio.h>
int main ()  {
   int a = 17;
   int b = 23;
   int c;
   c = a + b;
   printf(“Result is %d”, c);
   return 0;
}
```

The program prints the message:

Result is 40

C Identifier Syntax

- Can use
  - A – Z
  - a – z
  - 0 – 9
  - The underscore character
- Cannot begin with a number
- A_3, abcDS2, this_variable are fine
- 321, 5_r, dfd@dhr, this variable, no-entry are not

C Identifier Conventions

- Prefer to use **short but meaningful** names
- Use capital letters to identify program constants
- Use small letters to identify program variables

Keyword Usage

```
#include <stdio.h>
int main(void){
    int else = 3;
    printf("%d", else);
    return 0;
}
```

This won’t work

C character constants

```
#include <stdio.h>
int main(void){
    int a = 'B';
    printf("%d\n", a);
    return 0;
}
```

What do you think the output will be?

Character Constant Operations

```
#include <stdio.h>
int main(void){
    int a = ‘C‘ – ‘3’;
    printf("%d\n", a);
    return 0;
}
```

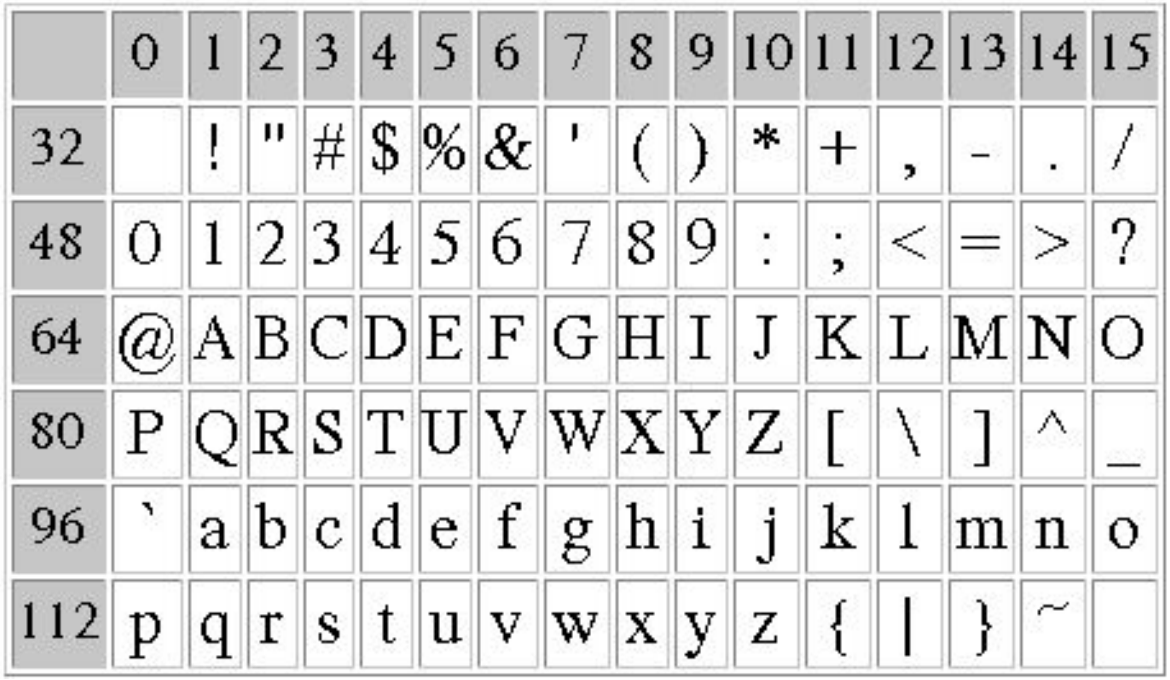
```
#include <stdio.h>
int main(void){
    int a = ‘c‘ – ‘3’;
    printf("%d\n", a);
    return 0;
}
```

Another Example: Playing with ASCII

- A program that converts Capital to small characters

```
# include <stdio.h>
int main(){
   char a = ‘D’;
   char b =_______;
   printf(“__ is now __\n”,a, b);
   return 0;
}
```

ASCII Table



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
32		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
48	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
96	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

Playing with ASCII

- A program that converts Capital to small characters

```
# include <stdio.h>
int main(){
    char a = ‘D’;
    char b = a-‘A’+’a’;
    printf(“__ is now __\n”,a, b);
    return 0;
}
```

Playing with ASCII

- A program that converts Capital to small characters

```
# include <stdio.h>
int main(){
   char a = ‘D’;
   char b = a-‘A’+’a’;
   printf(“%c is now %c\n”,a, b);
   return 0;
}
```

Another Simple Program

- A program that uses multiple types

```
# include <stdio.h>
int main(){
   char letter = ‘3’;
   int number = __________
   printf(“letter __ as a number is
__\n”, letter, number);
   return 0;
}
```

Another Simple Program

- A program that uses multiple types

```
# include <stdio.h>
int main(){
   char letter = ‘3’;
   int number = letter - ‘0’;
   printf(“letter __ as a number is
__\n”, letter, number);
   return 0;
}
```

Another Simple Program

- A program that uses multiple types

```
# include <stdio.h>
int main(){
   char letter = ‘3’;
   int number = letter - ‘0’;
   printf(“letter %c as a number is
%d\n”, letter, number);
   return 0;
}
```

Data Types in C

char

- Single character, e.g. a or C or 6 or $

- **int**
  - Bounded integers, e.g. 732 or -5
- **float**
  - Real numbers, e.g. 3.14 or 2.0
- **double**
  - Real numbers with more precision

Data Types in C

- Must declare a variable (specify its type and name) before using it anywhere in your program
- All variable declarations should be at the beginning of the main() or other functions
- A value can also be assigned to a variable at the time the variable is declared.

```
int   speed = 30;
char  flag = ‘y’;
```

Type Modifiers in C

- Signed
  - Range $[-2^{N-1}, 2^{N-1}-1]$
- Unsigned
  - Range $[0,2^{N}-1]$
- Short
  - Can use half the size of the normal data type
- Long
  - Can use double the size of the normal data type

Composite data types

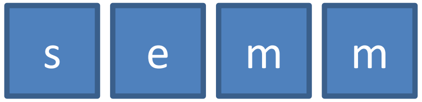
- signed short int = signed short = short (%hi)
- signed long int = signed long = long (%li)
- unsigned int (%u)
- float (%f)
- double (%f or %lf)
- long double (%Lf)

Floating Point Representation

- Have to represent three things
  - sign
  - Number
  - Exponent
- Assign some bits of memory for each
  - 1 bit for sign
  - m for exponent
  - n for mantissa

Conceptual Example

- Consider a 4 bit memory
  - What can you assign with unsigned int?
    - 0,1,.....15
  - What can you assign with signed int?
    - Use twos complement notation
    - -8,-7,.... ,7
  - What can you assign with float?



s e m m

$(-1)^s * 1.m * 2^{e-0}$

1.0, 1.1, 1.2, 1.3
2.0, 2.2, 2.4, 2.6
-1.0, -1.1, -1.2, -1.3
-2.0, -2.2, -2.4, -2.6

Edge Cases in float Representations

- Has upper bound
- Has lower bound
- Needs special handling for special numbers
  - Zero (when e and m are all zeros)
  - Infinity
- Exact matches can be problematic
  - Is x = 0.902323?

temp_conversion.c

Compile and Run

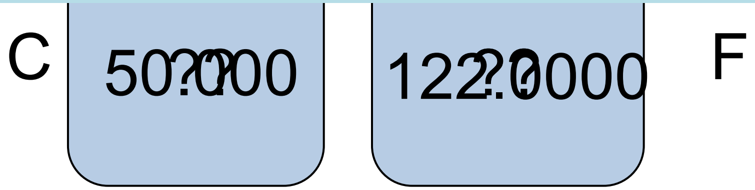
```
# include <stdio.h>
int main()  {
      float C;
      float F;
      C=50;
      F = ((9*C)/5) + 32;
      printf(“The temperature”);
      printf( “ %f ”, C);
      printf(“Celsius equals”);
      printf(“ %f ”, F);
      printf(“Fahrenheit”);
      return 0;
}
```

- Microprocessors represent real numbers using ***finite precision***, i.e., using *limited number of digits after decimal point.*

- Typically uses scientific notation:

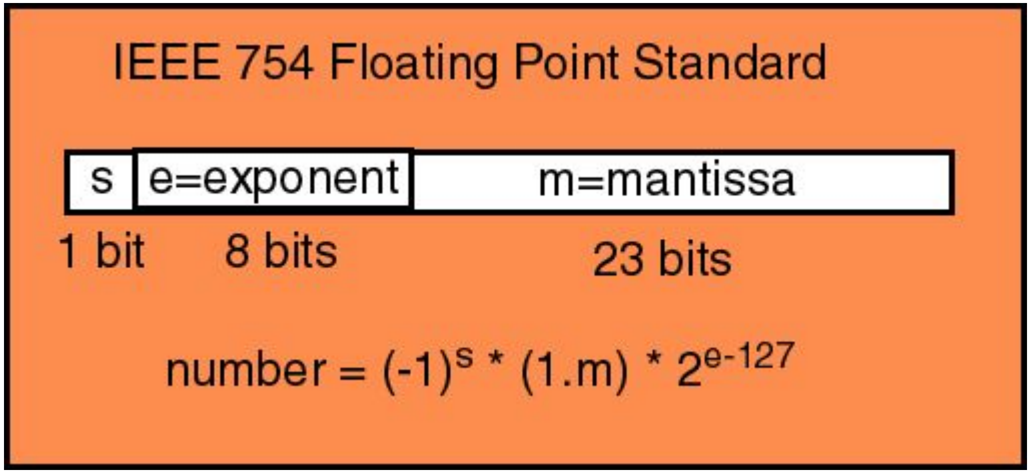
12.3456789 represented as 1.23456789E+1.

“%f” signifies that the corresponding variable is to be printed as a real number in decimal notation.



```
The temperature  50.000000  Celsius equals  122.000000  Fahrenheit
```

IEEE 754 Floating Point Representation

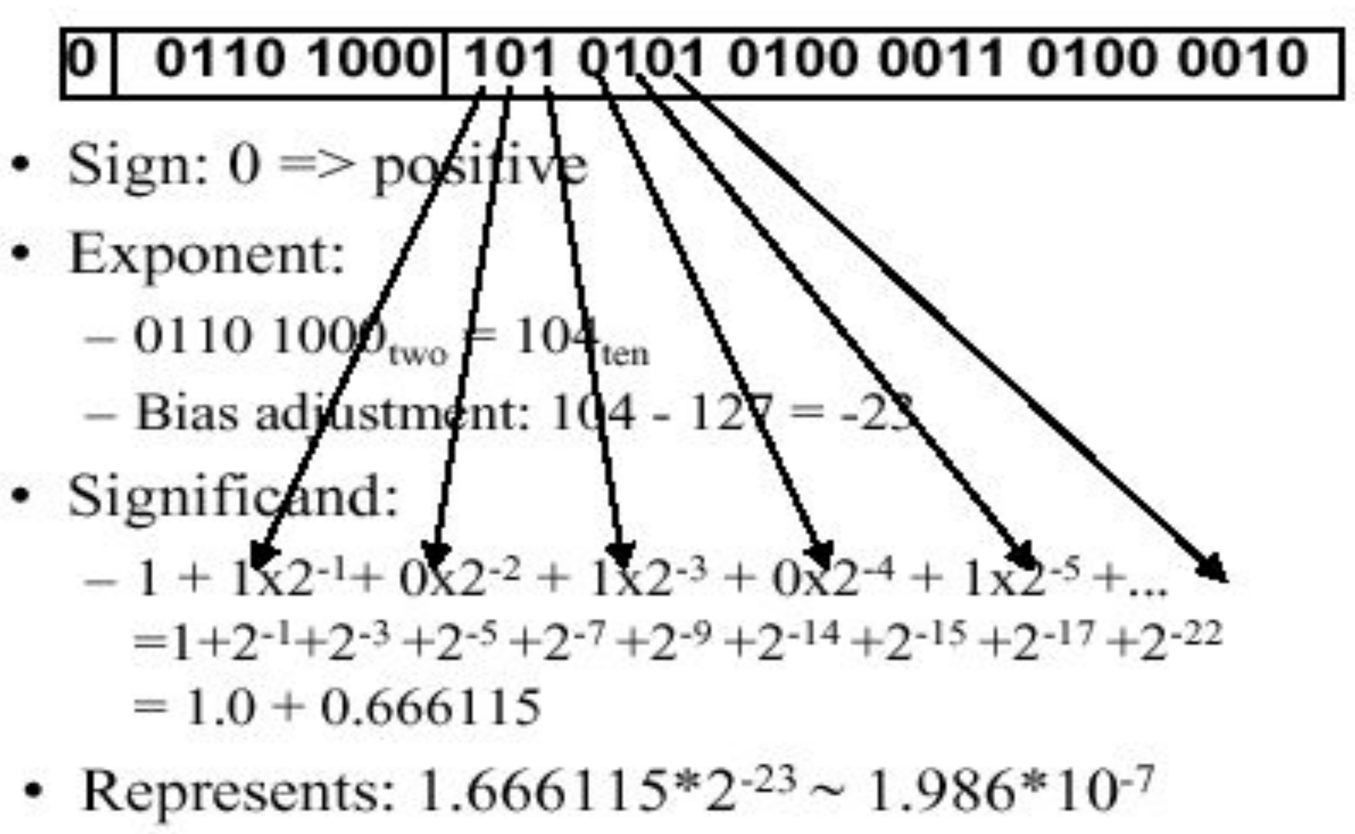


IEEE 754 Floating Point Standard

s	e=exponent	m=mantissa
1 bit	8 bits	23 bits

$$\text{number} = (-1)^s * (1.m) * 2^{e-127}$$

Single-precision Floating Point Representation



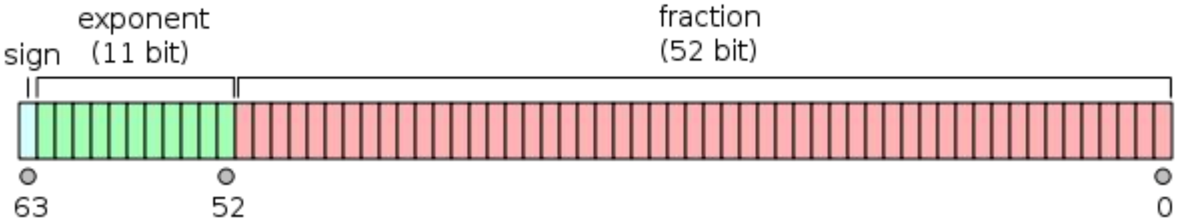
0	0110 1000	101 0101 0100 0011 0100 0010

- Sign: 0 => positive
- Exponent:
  - 0110 $1000_{\text{two}} = 104_{\text{ten}}$
  - Bias adjustment: 104 - 127 = -23
- Significand:
  - $1 + 1\text{x}2^{-1} + 0\text{x}2^{-2} + 1\text{x}2^{-3} + 0\text{x}2^{-4} + 1\text{x}2^{-5} + \ldots$
    $=1+2^{-1}+2^{-3}+2^{-5}+2^{-7}+2^{-9}+2^{-14}+2^{-15}+2^{-17}+2^{-22}$
    $= 1.0 + 0.666115$
- Represents: $1.666115*2^{-23} \sim 1.986*10^{-7}$

This is what you're using when you are invoking *float*

Double Precision

- Same logic as single precision, but with 64 bits of memory



Type Conversion (Type Casting)

- Converting values of one type to other.
  - Example: int to float and float to int (also applies to other types)
- Can be implicit or explicit

```
int k =5;
float x = k;              // good implicit conversion, x gets 5.0


float y = k/10;       // poor implicit conversion


float z = ((float) k)/10; // Explicit conversion, z gets 0.5
```

Typecasting Application

```
int main() {
    int total=100, number=50;
    float percentage=0.0;
    percentage=(number/total)*100;
    printf("%.2f",percentage);
    return 0;
}
```

Output: 0.00

```
int main() {
    int total=100, number=50;
    float percentage=0.0;
    percentage= (float) number/total*100;
    printf("%.2f",percentage);
    return 0;
}
```

Output: 50.00

Loss of Information!

- Type conversion may result in lost information
- Larger sized type (e.g. float) converted to smaller sized type (e.g. int) is undefined/ unpredictable

float to int: type conversion (result ok)

```
#include<stdio.h>
int main() {
     float x;  int y;      /* define two variables */
     x = 5.67;
     y = (int) x;          /* convert float to int    */
     printf(“%d”, y);
     return 0;
}
```

```
Output :    5
```

float x;

…

(int) x;

converts the real value stored in x into an integer. Can be used anywhere an int can.

float to int Type Conversion (not ok!)

- float is a larger box, int is a smaller box. Assigning a float to an int may lead to loss of information and unexpected values.

The floating point number 1E50 is too large to fit in an integer box.

```
# include <stdio.h>
int main() {
    float x; int y;
    x  = 1.0E50; //
    y = (int) x;
printf(“%d”, y);
   return 0;
}
```

Output:
2147483647

Careful when converting from a ‘larger’ type to ‘smaller’ type. Undefined.

Rocket Science

- First launch of the Ariane 5 rocket on 4th June 1996
- Rocket lost its flight path and disintegrated 40 seconds into launch
- Cost → $370 million
- Fundamental cause of disaster
  - Float to int data type casting

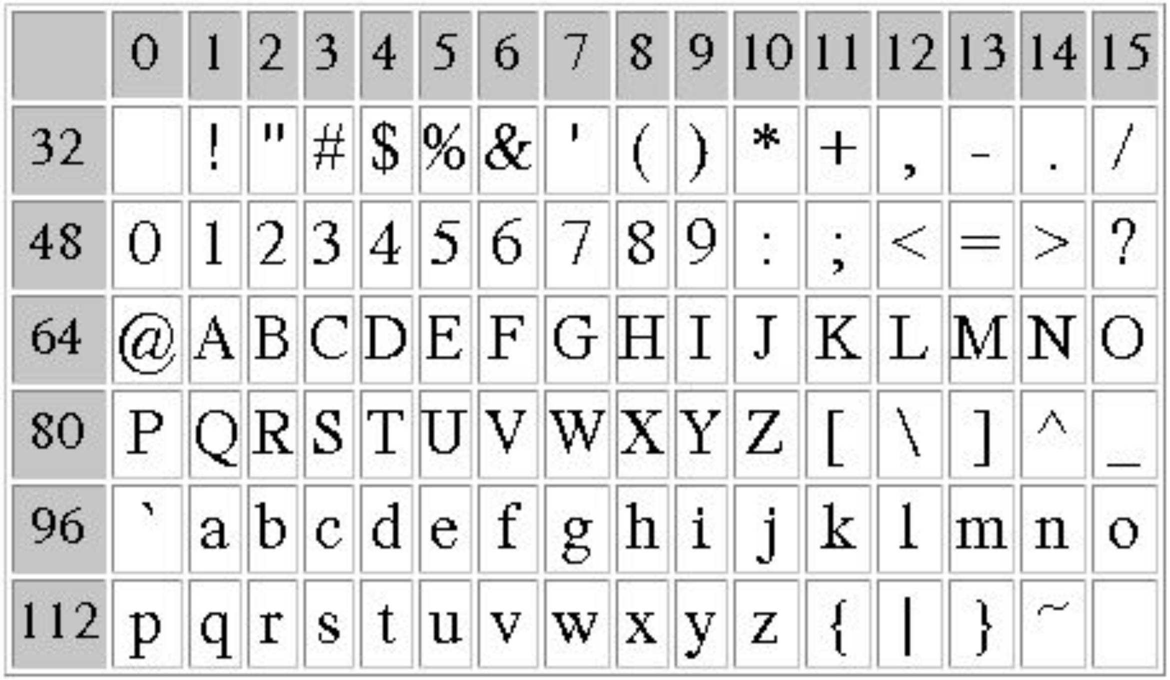
Rocket science

- Rocket’s horizontal velocity was greater than older rocket’s (Ariane 4)
- Inertial navigation system
  - A single calculation required conversion from double to int
  - Worked fine with the smaller value in Ariane 4
  - Gave a garbage result for the large value in Ariane 5
- Commanded engine to make an impossibly large course correction
- Veered off course
  - Auto-destruct sequence initialized

char to int

- Range: 0 to 255
- You should NOT try to remember ASCII values
  - Encoding/programming languages provide alternatives to use them
- C treats characters as integers corresponding to their ASCII value
- While displaying with %c placeholder, the ASCII value is converted to its corresponding character

ASCII character set



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
32		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
48	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
96	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

Translates letters to numbers for the computer to understand

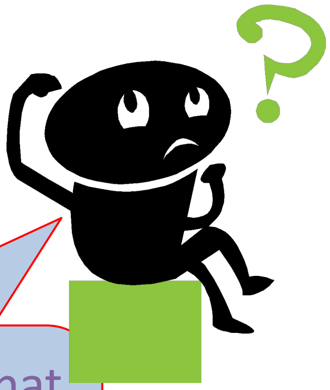
char □□ int

- conversion between character and integer datatypes can be exploited to write programs.

```
printf("%d\n", 'A');
printf("%d\n", '7');
printf("%c\n", 70);
printf("%c\n", 321);
```

```
Output:
65
55
F
```

321 is outside range! What do you think will be the output of printf("%c\n",321);
Try it out



char-int operations

- Interconversion between character and integer datatypes can be exploited to write programs.

```
printf("%c\n", ‘C’+5);
printf("%c\n", ‘D’ - ‘A’ + ‘a’ );
printf("%d\n", ‘3’ + 2);
```

```
Output:
H
d
53
```

- Placeholder determines the output.
- Use with caution.
- Avoid arithmetic operation such as * and / on characters.
- Common Mistake: Incorrect data type of placeholder.

Input: scanf function

- Performs input from keyboard
- It requires a format string and a list of variables into which the value received from the keyboard will be stored
- format string = individual groups of characters (usually '%' sign, followed by a conversion character), with one character group for each variable in the list

```
int a, b;
float c;
scanf("%d %d %f", &a, &b, &c);
```

Variable list (note the & before a variable name)

- Commonly used conversion characters

c	for char type variable
d	for int type variable
f	for float type variable
lf	for double type variable

- Examples

```
        scanf ("%d", &size) ;
    scanf ("%c", &nextchar) ;
    scanf ("%f", &length) ;
    scanf (“%d%d”, &a, &b);
```

Reading a single character

- A single character can be read using scanf with %c
- It can also be read using the getchar() function

```
char c;
c = getchar();
```

- Program waits at the getchar() line until a character is typed, and then reads it and stores it in c

Output: printf function

- Performs output to the standard output device (typically defined to be the screen)
- It requires a format string in which we can specify:
  - The text to be printed out
  - Specifications on how to print the values

    ```
    printf ("The number is %d\n", num);
    ```

  - The format specification %d causes the value listed after the format string to be embedded in the output as a decimal number in place of %d
  - Output will appear as: The number is 125

Contd.

- General syntax:

  printf (format string, arg1, arg2, …, argn);

  - format string refers to a string containing formatting information and data types of the arguments to be output
  - the arguments arg1, arg2, … represent list of variables/expressions whose values are to be printed
- The conversion characters are the same as in scanf

Contd.

- Examples:

```
printf  (“Average of %d and %d is %f”, a, b, avg
printf  (“Hello \nGood \nMorning \n”);
printf (“%3d %3d %5d”, a, b, a*b+2);
printf (“%7.2f  %5.1f”, x, y);
```

- Many more options are available for both printf and scanf
  - Read from the book
  - Practice them in the lab

Next Class

- C program structure
  - Variables
  - Declarations
  - Expressions
  - Statements