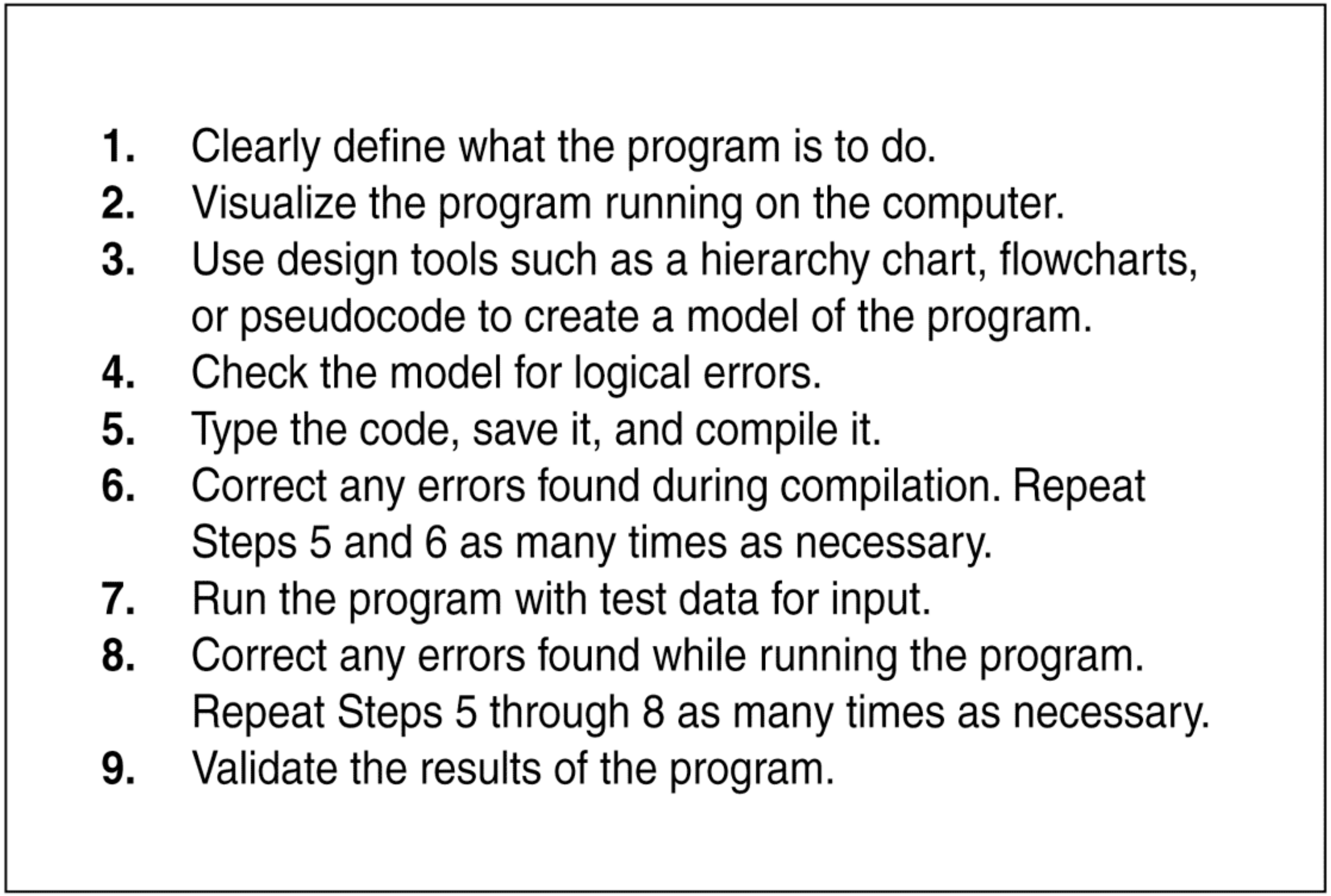
CIS 22A – Lecture 2

Manish Goel

The Programming Process



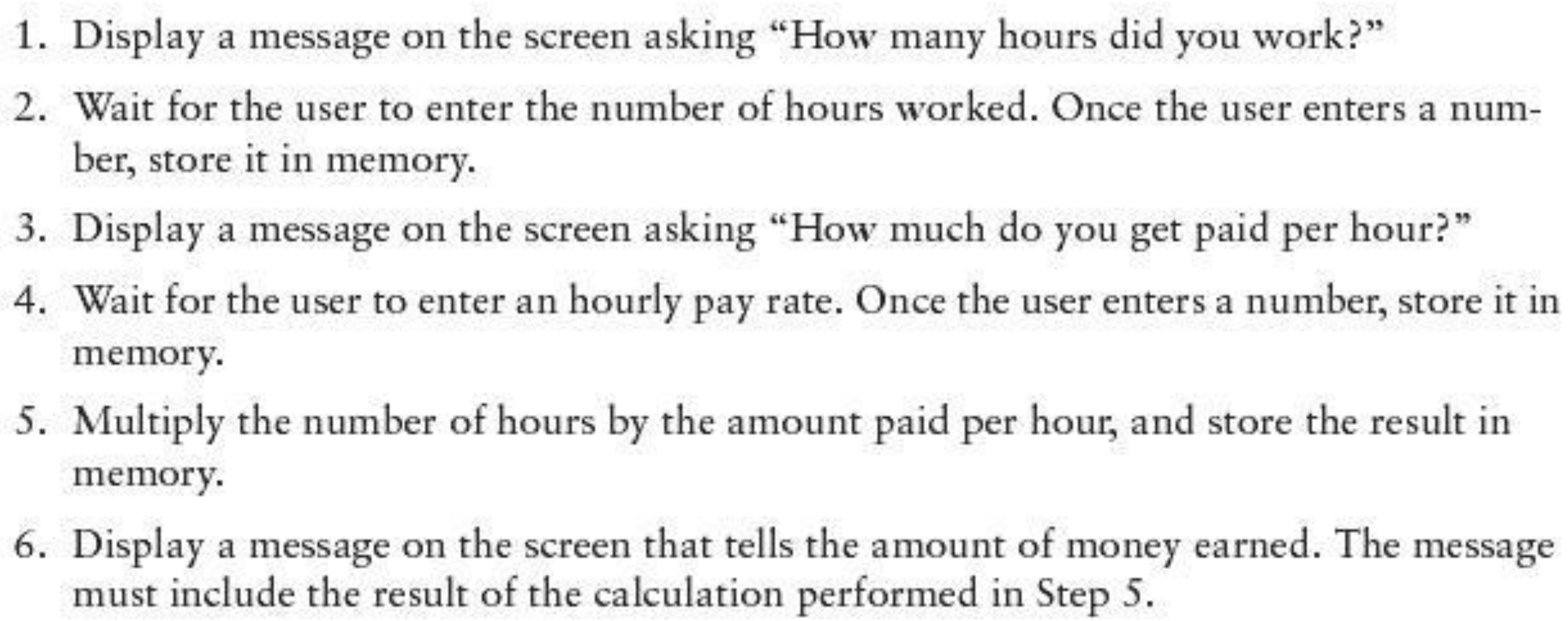
1. Clearly define what the program is to do.
2. Visualize the program running on the computer.
3. Use design tools such as a hierarchy chart, flowcharts, or pseudocode to create a model of the program.
4. Check the model for logical errors.
5. Type the code, save it, and compile it.
6. Correct any errors found during compilation. Repeat Steps 5 and 6 as many times as necessary.
7. Run the program with test data for input.
8. Correct any errors found while running the program. Repeat Steps 5 through 8 as many times as necessary.
9. Validate the results of the program.

Programs and Programming Languages

- A program is a set of instructions that the computer follows to perform a task

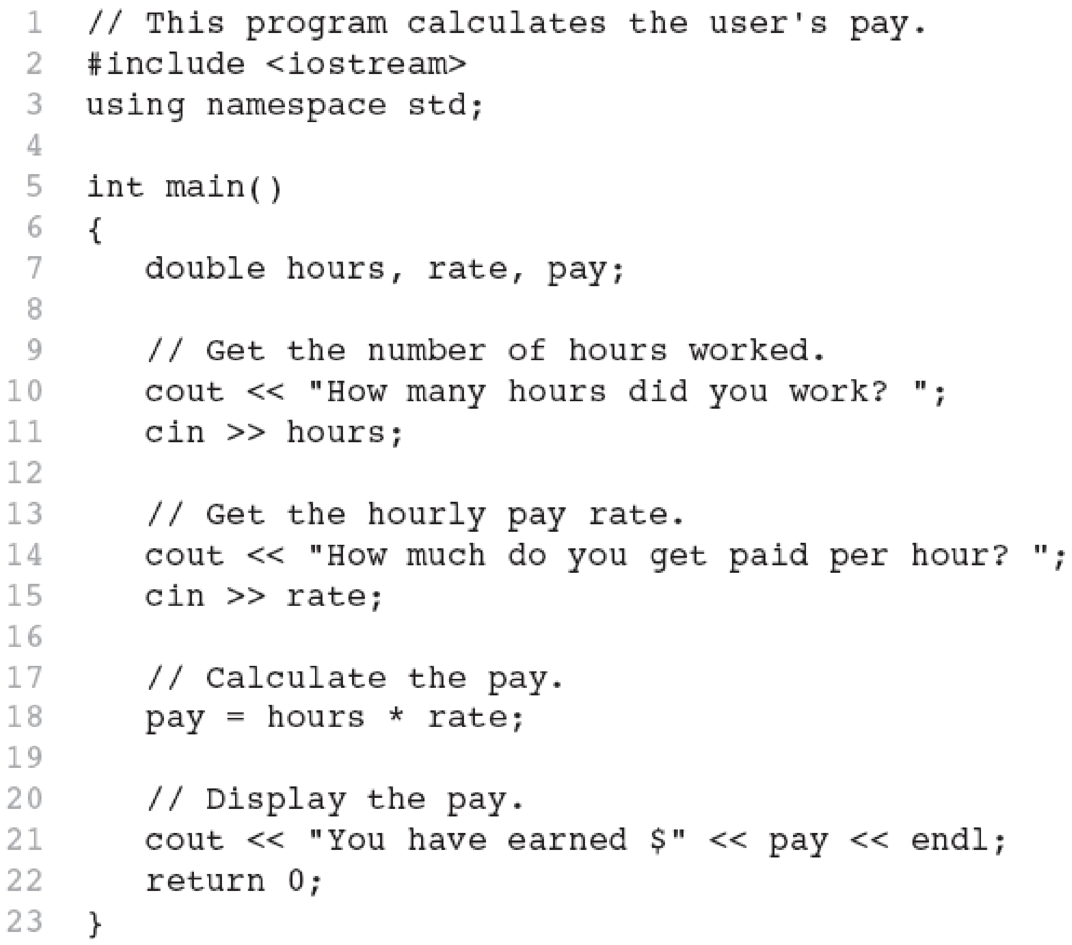
- We start with an *algorithm*, which is a set of well-defined steps.

Example Algorithm for Calculating Gross Pay



1. Display a message on the screen asking “How many hours did you work?”
2. Wait for the user to enter the number of hours worked. Once the user enters a number, store it in memory.
3. Display a message on the screen asking “How much do you get paid per hour?”
4. Wait for the user to enter an hourly pay rate. Once the user enters a number, store it in memory.
5. Multiply the number of hours by the amount paid per hour, and store the result in memory.
6. Display a message on the screen that tells the amount of money earned. The message must include the result of the calculation performed in Step 5.

Program 1-1



```
// This program calculates the user's pay.
#include <iostream>
using namespace std;

int main()
{
   double hours, rate, pay;

   // Get the number of hours worked.
   cout << "How many hours did you work? ";
   cin >> hours;

   // Get the hourly pay rate.
   cout << "How much do you get paid per hour? ";
   cin >> rate;

   // Calculate the pay.
   pay = hours * rate;

   // Display the pay.
   cout << "You have earned $" << pay << endl;
   return 0;
}
```

Key Words

- Also known as reserved words
- Have a special meaning in C++
- Can not be used for any other purpose
- Key words in the Program : `using`, `namespace`, `int`, `double`, and `return`

Operators

- Used to perform operations on data
- Many types of operators:
  - Arithmetic - ex: +, −, *, /
  - Assignment – ex: =

- Some operators in Program1-1:
  `<<  >>  =  *`

Identifiers, Variables and Literals

- Identifier: a programmer-defined name for parts of a program - variables, functions, etc. Identifiers are ‘Reserved’ or ‘User-Defined’

- Variable: a storage location in memory
  - Has a name and a type of data it can hold
  - Must be defined before it can be used

    ```
    int item;
    ```

- Literal: a value that is used as stated or assigned.

  `"hello, there"` (string literal)

  `12` (integer literal)

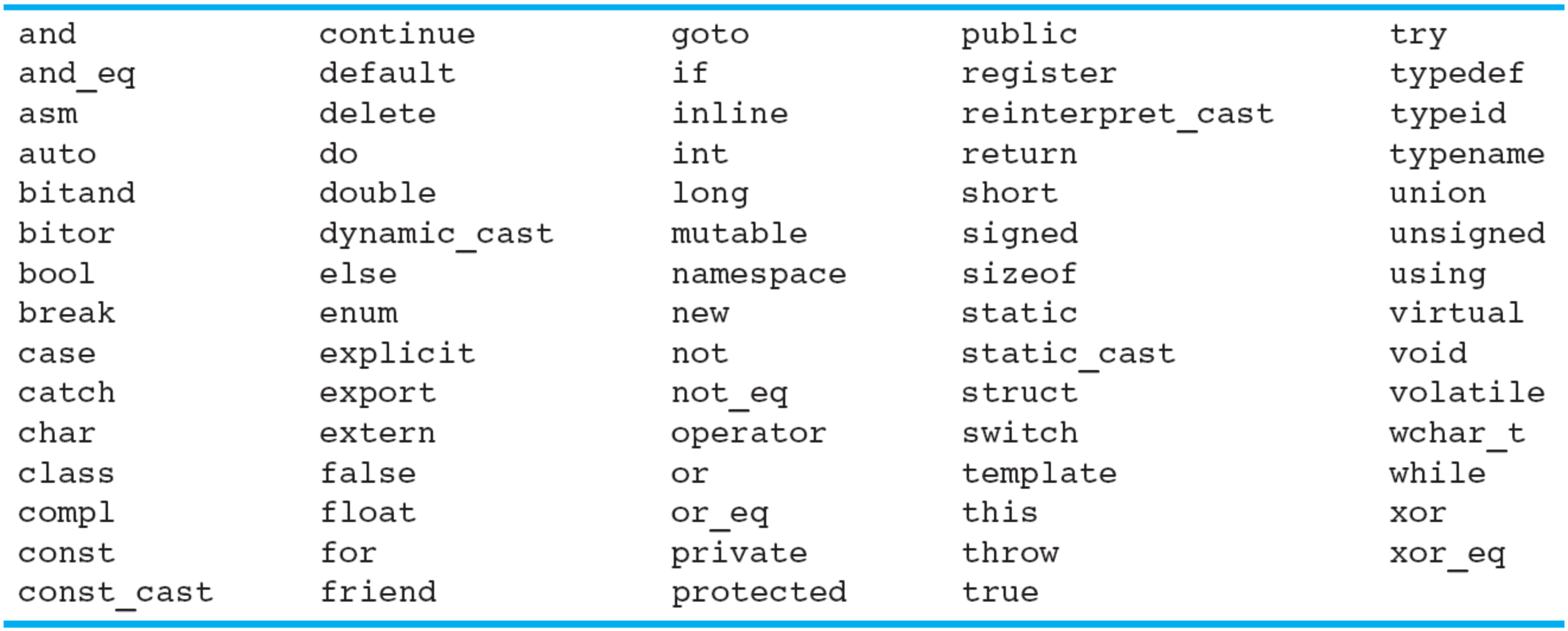
Identifier Rules

- An identifier is made up of letters (lower or upper case), numerals or underscore (_)
- The first character of an identifier must be an alphabetic character or and underscore (_), i.e. it cannot be a number
- Upper- and lowercase characters are distinct
- Identifier ‘names’ key pieces of your program – so should be appropriately descriptive

C++ Key Words

You cannot use any of the C++ key words as an identifier. These words have reserved meaning.

Table 2-4 The C++ Key Words



and	continue	goto	public	try
and_eq	default	if	register	typedef
asm	delete	inline	reinterpret_cast	typeid
auto	do	int	return	typename
bitand	double	long	short	union
bitor	dynamic_cast	mutable	signed	unsigned
bool	else	namespace	sizeof	using
break	enum	new	static	virtual
case	explicit	not	static_cast	void
catch	export	not_eq	struct	volatile
char	extern	operator	switch	wchar_t
class	false	or	template	while
compl	float	or_eq	this	xor
const	for	private	throw	xor_eq
const_cast	friend	protected	true	

Good vs Poor Naming

Allowed	Not Allowed
Main – not same as main	1st – first char cannot be number
MyBankBalance	MyBankBalance$
patientAge	Patients Age
_variable	-variable
qtr_sales	qtr.sales

Naming conventions are a good idea

Type	Convention
Integer or Whole Number	Start with ‘I’ or ‘int’ e.g. intNumber or iN
Decimal Number	Start with ‘f’ or ‘float’ e.g. floatValue or fV
Character or Letter	Start with ‘c’ or ‘char’ e.g. charCode or cC
Line of Text	Start with ‘s’ or ‘str’ e.g. strText or sText
Refers to some type of Object	Start with ‘o’ or ‘obj’ e.g. objCar

Numeric Data Types in C++

- Integer Data Types

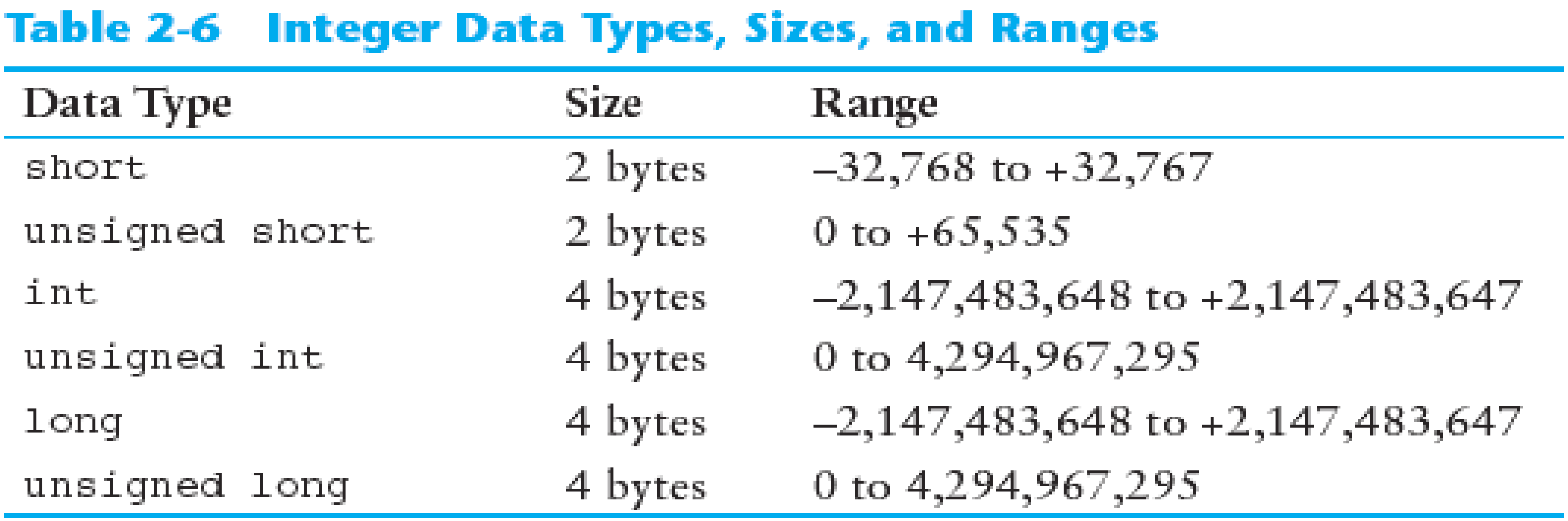


Table 2-6 Integer Data Types, Sizes, and Ranges

Data Type	Size	Range
`short`	2 bytes	–32,768 to +32,767
`unsigned short`	2 bytes	0 to +65,535
`int`	4 bytes	–2,147,483,648 to +2,147,483,647
`unsigned int`	4 bytes	0 to 4,294,967,295
`long`	4 bytes	–2,147,483,648 to +2,147,483,647
`unsigned long`	4 bytes	0 to 4,294,967,295

- Floating Point Data Types

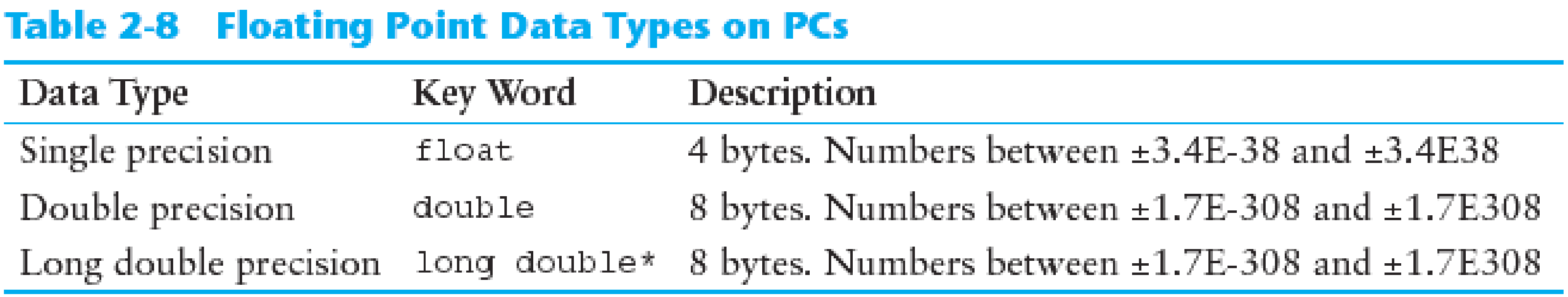


Table 2-8 Floating Point Data Types on PCs

Data Type	Key Word	Description
Single precision	`float`	4 bytes. Numbers between ±3.4E-38 and ±3.4E38
Double precision	`double`	8 bytes. Numbers between ±1.7E-308 and ±1.7E308
Long double precision	`long double*`	8 bytes. Numbers between ±1.7E-308 and ±1.7E308

Character Conversions

Decimal	Hexadecimal	Binary		Decimal	Hexadecimal	Binary
0	0	0000		8	8	1000
1	1	0001		9	9	1001
2	2	0010		10	A	1010
3	3	0011		11	B	1011
4	4	0100		12	C	1100
5	5	0101		13	D	1101
6	6	0110		14	E	1110
7	7	0111		15	F	1111

0110 in binary = $0*2^3 + 1*2^2 + 1*2^1 + 0*2^0$

= $0+4+1+0$ = 6 in decimal

AFC in HEX = $10*16^2 + 15*16^1 + 12*16^0$

= $2560+240+12$ = 2812 in decimal

Bits and Bytes

CPU Bits	Maximum RAM Size
4	2^{4} = 16 Bytes
8	2^{8} = 256 Bytes
16	2^{16} = 65,536 Bytes = 64 KB
32	2^{32} = 4294967296 Bytes = 4096 MB = 4 GB
64	2^{64} = 18446744073709551616 Bytes = 4 GB * 4 GB = 16 Exabyte

- The CPU loads the program into memory – preferable to have more memory than the size of the program so all of it can be loaded at once for fastest execution.

- A 64-bit computer that has the maximum possible RAM could probably store more data than all of Google.

Floating-Point Data

- Decimal numbers can be represented in
  - Fixed point (decimal) notation: `31.4159 or 0.0000625`
  - E notation: `3.14159E1 or 6.25e-5`
- The floating-point data types are:
  `float, double, long double`
- Are `double` by default
- All floating-point numbers are signed
- Can be forced to be float (`3.14159f`) or long double (`0.000625L`)

Named Constants

- Named constant : whose content cannot be changed during program execution
- Also called constant variable
- Identified by keyword `const`
- Often named in uppercase letters
- Used for representing constant values with descriptive names:

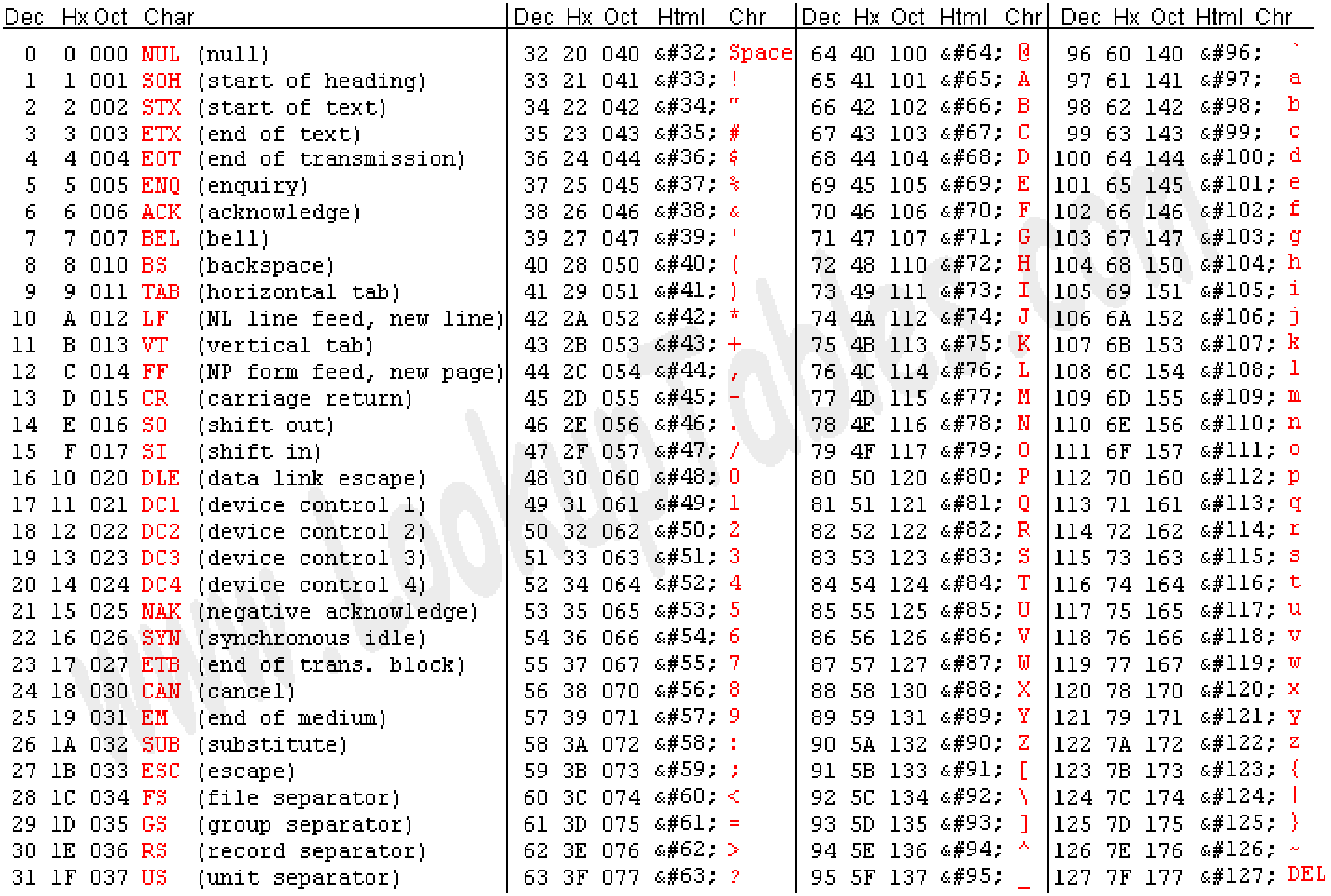
```
   const double TAX_RATE = 0.0675;
   const int NUM_STATES = 50;
```

Other Data Types

BOOL	bit	0 or 1 / T or F / Y or N
CHAR	character	Needed to store single letters Encapsulated by single quotes Upper case diff from lower case
STRING	string	Needed to store English words or phrases Encapsulated by double quotes Strings always have \0 escape sequence at the end

- **‘A’ is a CHAR but “A” is a STRING**

US ASCII Charset



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	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH	(start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX	(start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX	(end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT	(end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ	(enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK	(acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL	(bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS	(backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB	(horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT	(vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR	(carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO	(shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI	(shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE	(data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1	(device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2	(device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3	(device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4	(device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK	(negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN	(synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB	(end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN	(cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM	(end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB	(substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC	(escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS	(file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS	(group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS	(record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US	(unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Escape Sequences

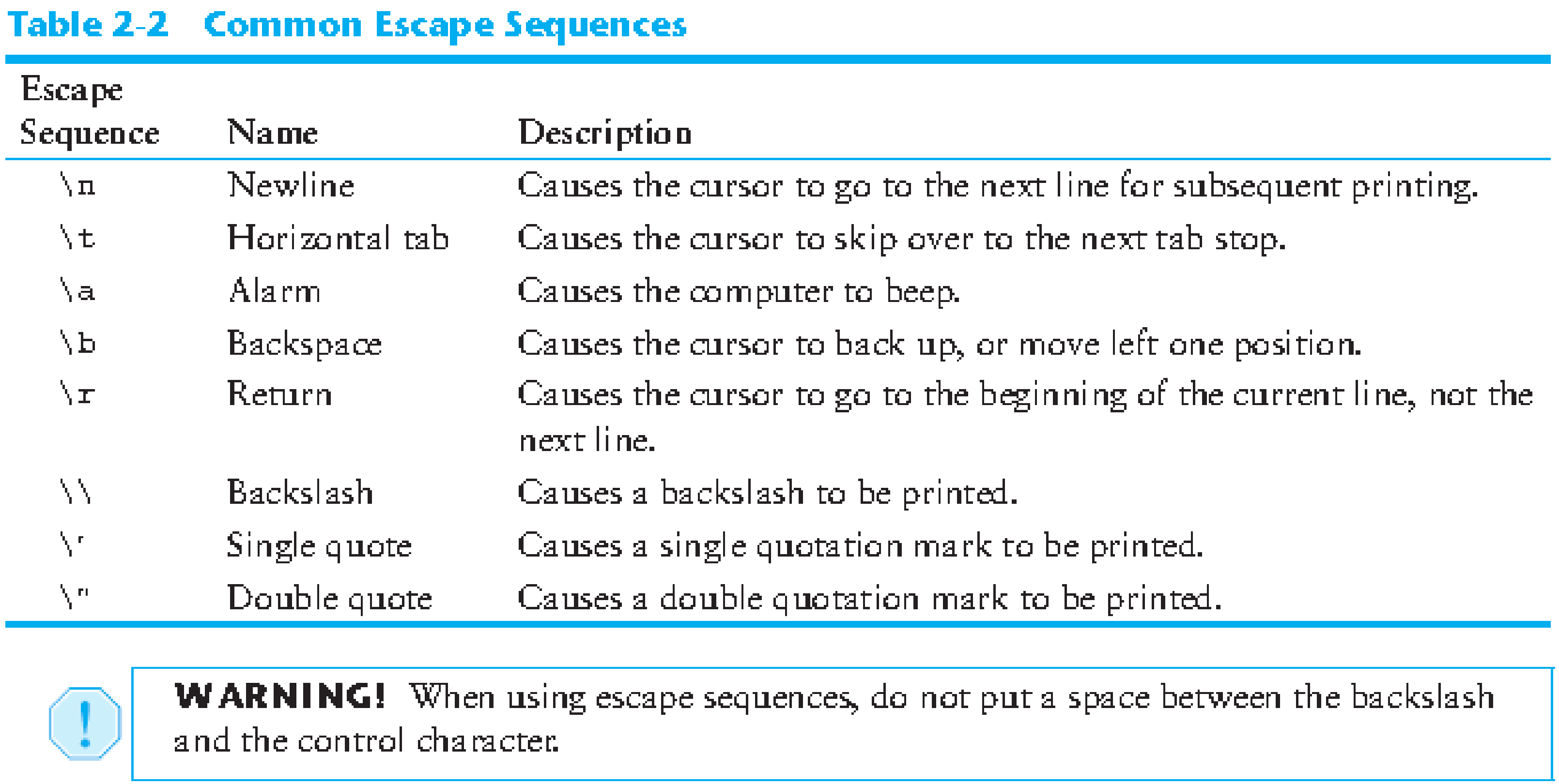


Table 2-2 Common Escape Sequences

Escape Sequence	Name	Description
`\n`	Newline	Causes the cursor to go to the next line for subsequent printing.
`\t`	Horizontal tab	Causes the cursor to skip over to the next tab stop.
`\a`	Alarm	Causes the computer to beep.
`\b`	Backspace	Causes the cursor to back up, or move left one position.
`\r`	Return	Causes the cursor to go to the beginning of the current line, not the next line.
`\\`	Backslash	Causes a backslash to be printed.
`\'`	Single quote	Causes a single quotation mark to be printed.
`\"`	Double quote	Causes a double quotation mark to be printed.

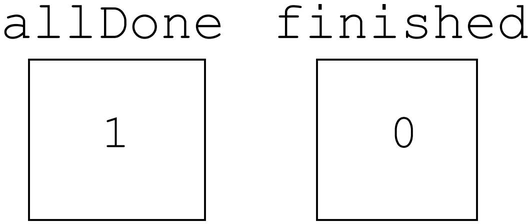
WARNING! When using escape sequences, do not put a space between the backslash and the control character.

The `bool` Data Type

- Represents values that are `true` or `false`
- `bool` variables are stored as small integers
- `false` is represented by 0, `true` by 1:

```
bool allDone = true;

bool finished = false;
```



Adding two numbers together

Starting Out

```
// sample C++ program
#include <iostream>
using namespace std;

int main()
{
  int firstN;    //define
  int secondN;
  int result;

  firstN = 5;    //assign
  secondN = 6;
  result = firstN + secondN;

  cout << “The result is ";
  cout << result;
  return 0;
}
```

Alternate & Concise

```
// sample C++ program
#include <iostream>
using namespace std;

int main()
{
  int fN = 5; //define and
  int sN = 6; //assign
  int res;    //define only

  res = fN + sN;

  cout << “The result is ”
       << res;
  return 0;
}
```

The C++ `string` Class

- Object data type for a series of characters

```
#include <string>
```

- Allows defining `string` variables in programs:

```
string firstName, lastName;
```

- Allows receiving values with assignment operator:

```
firstName = "George";
lastName = "Washington";
```

- Can be displayed via `cout`

```
cout << firstName << " " << lastName;
```

Character Strings

- The series of characters of a string are stored in consecutive memory locations:

  `"Hello"`

- Stored with the null terminator, `\0`, at the end:

- Comprised of the characters between the `"   "`

H	e	l	l	o	\0

Things to remember

- A number can be a character or string literal

```
– char cN = ‘5’;               // character data type
– string strN = “56”;          // string data type
```

- int data type can be stored as floating point with decimals

```
– double dNum = 57;            // stored as 57.0
```

- floating point data type when stored as int get truncated

```
– int iNum = 57.23;            // stored as 57
```

Determining the Size of a Data Type

The `sizeof` operator gives the size of any data type or variable:

```
double amount;
cout << "A double is stored in "
     << sizeof(double) << "bytes\n";
cout << "Variable amount is stored in "
     << sizeof(amount)
    << "bytes\n";
```