Outline

- Variables
- Constants
- Statements
- Expressions
- Operators

Computer Memory

- A *byte* is the fundamental unit of computer data storage.
- Each byte of memory has a unique *address* by which it is identified.

0	41
1	42
2	43
3	30
	…
12560	5F
12561	60
12562	C1
	…

Random
Access
Memory

Temperature Display Program

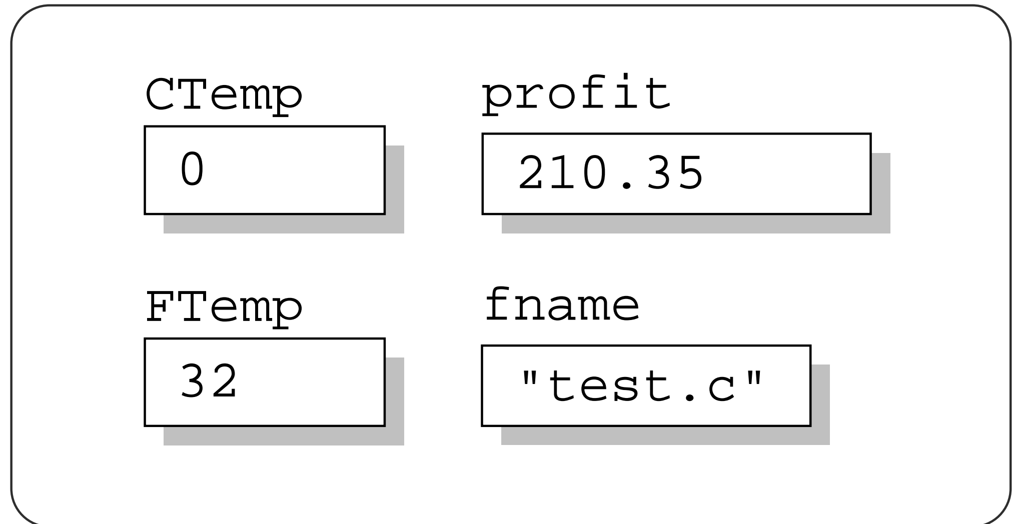
27 ? c

```
main()
{
  int   FTemp, CTemp;

  for(;;) {
    FTemp = GetCurrentTemperature();
    CTemp = FahrenheitToCelsius( FTemp );
    DisplayPanel( CTemp );
    Wait( 5 );
  }
}
```

Variables

- A variable is a named data storage location in your computer's memory.



Computer Memory

Variable Names

- The name can contain letters, digits, and the underscore character (_).
- The first character of the name can not be a digit.
- Case (upper- and lower-case letters) matters.
- C keywords cannot be used as variable names.

```
Count        ✍
count        ✍
kilo2pound✍
year_1994    ✍
year1994     ✍
_income      ✍
double       ✍
1pen         ✍
U&I          ✍
```

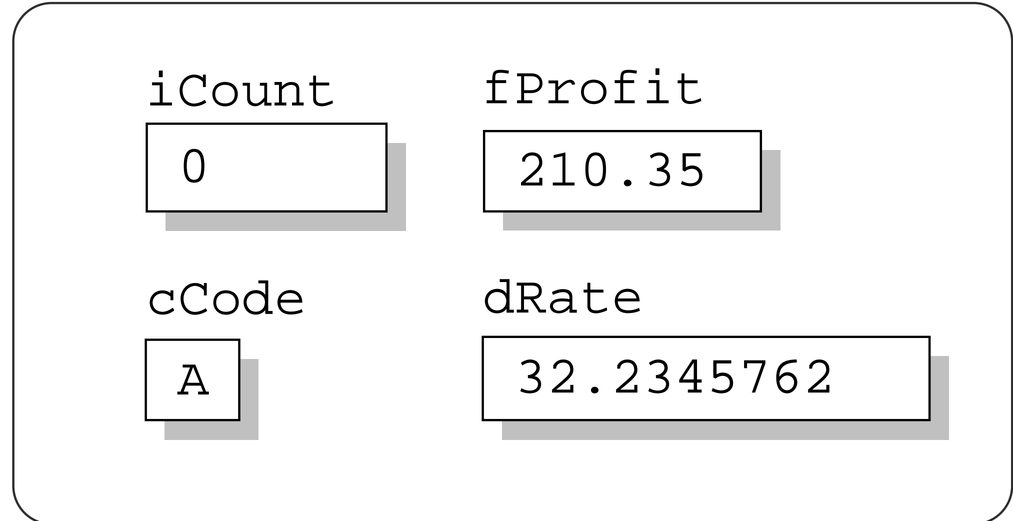
Do & Don't

- Do use variable names that are descriptive.
- Do adopt and stick with a style for naming your variables.
- Don't start your variable names with and underscore unnecessarily.
- Don't name your variables with all capitals unnecessarily.

```
interest_rate
interestRate
InterestRate
fInterestRate
```

Data Types

- Why do we need different types of variables ?
  - Different types have different sizes.
  - Different precisions.



C's Numeric Data Types

Variable type	Keyword	Bytes required	Range
character	`char`	1	-128 to 127
integer	`int`	2	-32768 to 32767
short integer	`short`	2	-32768 to 32767
long integer	`long`	4	-2,147,438,648 to 2,147,438,647
unsigned character	`unsigned char`	1	0 to 255
unsigned integer	`unsigned int`	2	0 to 65535
unsigned short integer	`unsigned short`	2	0 to 65535
unsigned long integer	`unsigned long`	4	0 to 4,294,967,295
single-precision floating-point	`float`	4	1.2E-38 to 3.4E38
double-precision floating-point	`double`	8	2.2E-308 to 1.8E308

Sizeof()

```
/* SIZEOF.C                                      */
/* program to tell the size of C data types */
#include <stdio.h>

main()
{
  printf("An int  is %d bytes\n", sizeof( int )   );
  printf("A short is %d bytes\n", sizeof( short ) );
  printf("A long  is %d bytes\n", sizeof( long )  );
}
```

Variable Declarations

- inform the compiler the name and type of a variable
- optionally initialize the variable to a specific value

```
typeName   variableName ;
```

```
int     numItems;
short   i, j, k;
float   total = 0.0;
char    name[30] = "Somchai";
```

```
int     weight = 10000000;
unsigned int value = -2.5;
```

Be Careful

Do & Don't

- Do initialize variables when you declare them.
- Don't use a variable that has not been initialized.
- Don't use `float` or `double` variable if you are only storing integers (although they will work, using them is inefficient).
- Don't try to put numbers into variable types that are too small to hold them.
- Don't put negative numbers into variables with an `unsigned` type.

Constants

- The value stored in a constant can not be changed during program execution.
- Literal constants
  - a value that is typed directly into the source code

```
area = 3.1416 * radius * radius;
```

- Symbolic constants
  - a

```
area = PI * radius * radius;
```

Literal Constants : Integer

- Integer constants
  - `245`
  - `-45`
  - `102983L`
  - `29`   (decimal)
  - `071`   (octal)
  - `0x1D` (hexadecimal)

Literal Constants : Floating-Point

- Floating-point constants
  - 123.78
  - 0.294
  - -0.00034
  - 1.346e20 (1.346×10^{20})
  - 0.25E-3 (0.25×10^{-3})
  - -9.28E-2

Literal Constants : Character

- Character constants
  - `'c'`
  - `\t    (tab)`
  - `\n    (new line)`
  - `\\    (back slash)`
  - `\'    (single quote)`
  - `\"    (double quote)`
  - `\014  (\ddd)`
  - `\0    (null)`

ASCII Character Set

	0	1	2	3	4	5	6	7
0	*NUL*	*DLE*	*SP*	0	@	P	`	p
1	*SOH*	*DC1*	!	1	A	Q	a	q
2	*STX*	*DC2*	"	2	B	R	b	r
3	*ETX*	*DC3*	#	3	C	S	c	s
4	*EOT*	*DC4*	$	4	D	T	d	t
5	*ENQ*	*NAK*	%	5	E	U	e	u
6	*ACK*	*SYN*	&	6	F	V	f	v
7	*BEL*	*ETB*	'	7	G	W	g	w
8	*BS*	*CAN*	(	8	H	X	h	x
9	*HT*	*EM*	)	9	I	Y	i	y
A	*LF*	*SUB*	*	:	J	Z	j	z
B	*VT*	*ESC*	+	;	K	[	k	{
C	*FF*	*FS*	,	<	L	\	l	|
D	*CR*	*GS*	-	=	M	]	m	}
E	*SO*	*RS*	.	>	N	^	n	~
F	*SI*	*US*	/	?	O	_	o	*DEL*

S. Prasitjutrakul

Literal Constants : String

- String constants
  - `"I am a string constant"`
  - `""    (null string)`
  - `"I am \t a boy \n"`
  - `"Double quote \" and single quote \'"`

I		a	m		\t		a		b	o	y		\n	\0

\0

Symbolic Constants

- A constant that is represented by a name.
  - `#define` directive

```
#define  PI  3.14159
area = PI * radius * radius;
```

  - `const` keyword

```
const float PI = 3.14159;
area = PI * radius * radius;
```

A Short C Program

```
#include  <stdio.h>

const float  PI = 3.14159;
float fRadius, fArea;

main()
{
  printf("Radius = ");
  scanf( "%f", &fRadius );
  fArea = PI * fRadius * fRadius;
  printf("Area   = %f\n", fArea);
}
```

A Short C Program

```
$ a.out
Radius = 10
Area   = 314.159
$
```

Statements

- A complete direction instructing the computer to carry out some task.
- C statements always end with a semicolon.

```
area = PI * radius * radius;

area=PI*radius*radius;

area    = PI    *radius*  radius        ;

area =  PI
 * radius              *
               radius
    ;
```

Whitespace

Literal String vs. Whitespace

```
printf( "Hello World !" );

printf(
  "Hello World !"
);

printf( "Hello
  World !" );

printf( "Hello \
World !" );
```

Compound Statements

- A group of one or more C statements enclosed in braces (also called a *block*).
- No semicolon after the right brace that ends a block.

```
{
  printf( "Hello " );
  printf( "World !" );
}

{
  int    temp;
  temp = x;
  x    = y;
  y    = temp;
}
```

Do & Don't

- Do stay consistent with how you use whitespace in statements.
- Do put block braces on their own line to make the code easier to read.
- Do line up block braces so that it is easy to find the beginning and end of block.
- Don't spread a single statement across multiple lines it there is no need. Try to keep it on one line.

Expressions

- In C, an expression is anything that evaluates to a numeric value.

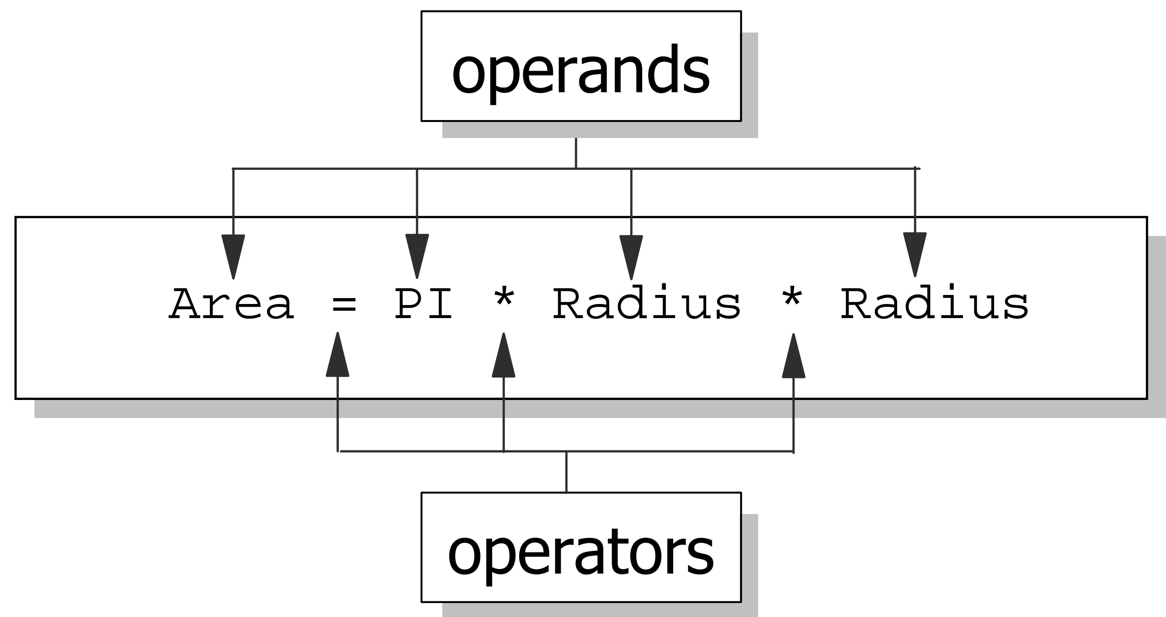
```
PI
20
-1.25
Area

25 + 72
PI * Radius * Radius
Area = PI * Radius * Radius

y = x = z + 5
q = 5 + ( y = x = z + 5 )
```

Operators

- An operator is a symbol that instructs C to perform some operation or action on one or more operands.
- An operands is something that an operator acts on. In C, all operands are expressions.



Assignment Operator

- The assignment operator is the equal sign =
- `x = y`
  - does not mean "`x` is equal to `y`".
  - means "assign the value of `y` to `x`", rather.

```
variable = expression ;
```

- The right side can be any expression.
- The left side must be a variable name.

Arithmetic Operators

Operator	Symbol	Action	Example
Addition	+	Adds its two operands	x + y
Substraction	–	Substracts the second operand from the first	x – y
Multiplication	*	Multiplies its two operands	x * y
Division	/	Divides the first operand by the second operand	x / y
Modulus	%	Used with integers only. Gives the remainder when the first operand is divides by the second operand	x % y
Increment	++	Increment operand by one	++x x++
Decrement	––	Decrement operand by one	––x x––

S. Prasitjutrakul

Increment & Decrement

<table>
<tr><td>++x;
x++;</td><td>x = x + 1;</td></tr>
<tr><td>--x;
x--;</td><td>x = x - 1;</td></tr>
<tr><td>x = 10;
y = ++x;</td><td>x = 10;
x = x + 1;
y = x;
(x = 11, y = 11)</td></tr>
<tr><td>x = 10;
y = x++;</td><td>x = 10;
y = x;
x = x + 1;
(x = 11, y = 10)</td></tr>
<tr><td>x = 10;
y = --x;</td><td>x = 10;
x = x - 1
y = x;
(x = 9, y = x)</td></tr>
<tr><td>x = 10;
y = x--;</td><td>x = 10;
y = x;
x = x - 1;
(x = 10, y = 9)</td></tr>
</table>

Operator Precedence

Operators	Relative Precedence
++ --	1
* / %	2
+ -	3

```
x = 3 + 4 * 5;
x = q + f * ++y

/* left-to-right order */
x = y / z * p;

/* use parentheses */
x = ((3 + 4) * (9 - 7)) / 8;
```

Operator Precedence

- C, like most languages, does not specify in what order the operands of an operator are evaluated.

```
x = (expr 1) + (expr 2);

y = 5;
++y;
x = 10 * y + 10 / y;

y = 5;
x = 10 * ++y + 10 / y;

printf( "%d %d", ++n, 2*n );
```

Do & Don't

- Do use parentheses to make the order of expression evaluation clear.
- Don't overload an expression. Make the expression clearer by breaking it into two or more statements.

```
x = ++y + z-- * 2 / q ;
```

Relational Operators

- Relational operators are used to compare expressions.
- True or False.

```
(x + 5) > y

 x > y - 5

(y - 5) < x
```

Relational Operators

Operator	Symbol	Question Asked	Example
Equal	`==`	Is op1 equal to op2 ?	`x == y`
Greater than	`>`	Is op1 greater than op2 ?	`x > y`
Less than	`<`	Is op1 less than op2 ?	`x < y`
Greater than or equal	`>=`	Is op1 greater than or equal to op2 ?	`x >= y`
Less than or equal	`<=`	Is op1 less than or equal to op2 ?	`x <= y`
Not equal	`!=`	Is op1 not equal to op2	`x != y`

Expression	Evaluated as
`5 == 1`	0 (false)
`5 > 1`	1 (true)
`5 != 1`	1 (true)
`(5+10) == (3*5)`	1 (true)

Do & Don't

- Do learn how C interprets true and false.
- Don't confuse ==, the relational operator, with =, the assignment operator.

```
y = (x = 5);        /* y = 5 */

y = (x == 5);       /* y = 1 */

y = (x == 6);       /* y = 0 */

y = (2 == 2) + (7 != 8);
```

Logical Operators

- "If it's in the evening or a weekend and not my vacation."

Operator	Symbol	Example
and	&&	*expr1 && expr2*
or	||	*expr1 || expr2*
not	!	! *expr*

expr1	*expr2*	*(expr1 && expr2)*	*(expr1 || expr2)*	*(!expr1)*
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

Logical Operators : Examples

```
(x > y) || (x < y)
(x == y) && (x != y)

(x <= y) && (x >= y)
(x != y) || (x == y)

(x != y) && (!(x == y))

(x)
(x != 0)
(!x)
(x == 0)
```

Logical Operators : Precedence

Arithmetic > Relational > Logical

```
! (NOT)  && (AND)  || (OR)

!(x > y) || (x < y)
(x == y) && !(x != y)

x==5 || x==7 && ! x==8
```