First Year First Semester Course

M.Tech. (CS) [Batch 2021-23]

Lecture #02

Introduction to Programming

Introduction to C: Syntax, Basic Constructs, Operators, Expressions

Course Instructor:

Dr. Monidipa Das

DST-INSPIRE Faculty

Machine Intelligence Unit (MIU), Centre for Artificial Intelligence and Machine Learning (CAIML)

Indian Statistical Institute (ISI) Kolkata, India

Topics of Discussion

- Programming in C
  - Example Program
  - Compilation and Execution
- Syntax and Basic Constructs
  - Identifiers
  - Keywords
  - Data Types
  - Constants
  - Variables
- Operators and Expressions

Programming in C

First C program – print on screen


```
#include <stdio.h>
int main()
{
    printf("Hello, World!\n");
    return 0;
}
```

Header file includes functions for input/output

Main function is executed when you run the program. (Later we will see how to pass its parameters)

Return value to function

Statement for printing; '\n' denotes newline

A program must have an output.

Curly braces within which statements are executed one after another.

Output

```
Hello, World!
```

Three steps to follow

1. Write a C program and save it.
2. Compile the program using the compiler.
3. Execute the program

1. vi hello.c

```
#include <stdio.h>
int main()
{
  printf("Hello World\n");
  return 0;
}
```

OR

2. $ gcc –o hello hello.c

3. $ ./hello

2. $ cc hello.c
$

3. $ ./a.out
Hello World

Introduction to C

- ***C*** is a general-purpose, structured programming language.
- ***C*** can be used for applications programming as well as for systems programming.
- There are only 32 keywords and its strength lies in its built-in functions.
- ***C*** is highly portable
- ***C*** is case sensitive.
- ***C*** is a free-form language.

Structure of a C program

- Every C program consists of one or more functions.
  - One of the functions must be called *main*.
  - The program will always begin by executing the main function.

- Each function must contain:
  - A function *heading*, which consists of the *function name*, followed by an optional list of *arguments* enclosed in parentheses.
  - A *return type*
  - A *compound statement*, which comprises the remainder of the function.

Structure of a C program

- Each compound statement is enclosed within a pair of braces: ‘{‘ and ‘}’
  - The braces may contain combinations of elementary statements and other compound statements.
- Statements are executed one by one in order
- Comments may appear anywhere in a program, enclosed within delimiters ‘/*’ and ‘*/’.
  - Example:

```
a = b + c; /* ADD TWO NUMBERS */
```

A Simple C program

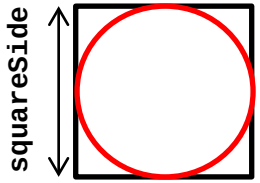

```
#include <stdio.h>
int main()
{   int x, y, sum, max;
    scanf("%d%d", &x, &y);
    sum = x + y;
    if (x > y)
         max = x;
    else
         max = y;
    printf ("Sum = %d\n", sum);
    printf ("Larger = %d\n",
    max);
    return 0;
}
```

When you run the program

Output after you type 15 and 20

```
15 20
Sum = 35
Larger = 20
```

A complete C Program




```
#include <stdio.h>
#define PI 3.1416
double area_of_circle(float);
double area_of_circle (float radius)
{
        return PI*radius*radius;
}
int main()
{
        int squareSide;
        double area;
        scanf("%d", &squareSide);
        area= area_of_circle(squareSide/2.0);
        printf("Area of the circle enclosing the square of side %d is: %lf\n",
squareSide, area);
        return 0;
}
```

Syntax and Basic Constructs in C

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:

A C program should not contain anything else

```
!     #     %     ^     &     *     (
)     -     _     +     =     ~     [
]     \     |     ;     :     ‘     “
{     }     ,     .     ?     <     >     /
whitespace (space, tab)
```

Identifiers

- Names given to the various program elements (variables, constants, functions, etc.)
- May consist of *letters*, *digits* and the *underscore* ('_') character, with no space in between.
- First character must be a letter or *underscore*.
- An identifier can be arbitrary long.
  - Some ***C*** compilers recognize only the first few characters of the name (16 or 31).
- Case sensitive
  - 'area', 'AREA' and 'Area' are all different.

Valid and Invalid Identifiers

- **Valid identifiers**

```
X
abc
simple_interest
a123
LIST
stud_name
Empl_1
Empl_2
avg_empl_salary
```

- **Invalid identifiers**

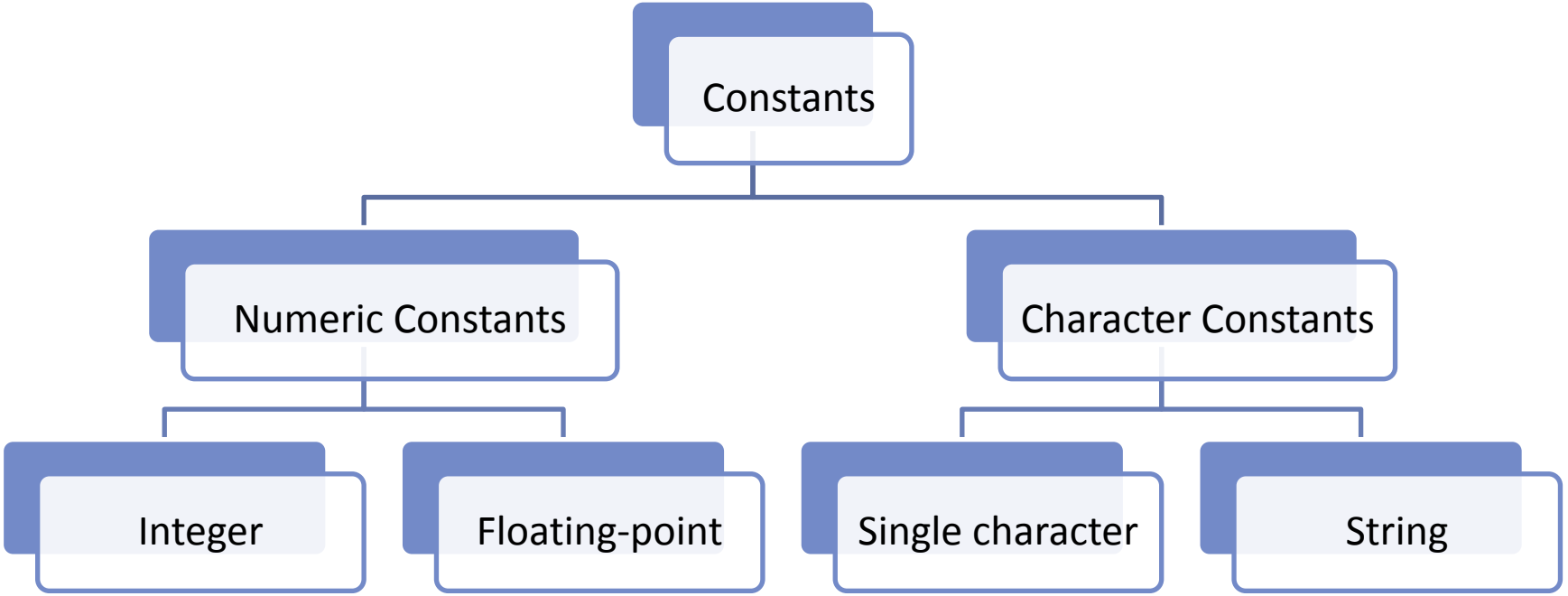
```
10abc
“hello”
simple interest
(area)
%rate
```

Keywords

- Keywords
  - Reserved words that have standard, predefined meanings in ***C***.
  - Cannot be used as identifiers.
  - OK within comments.
  - Standard ***C*** keywords:

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

Constants


Constants
Numeric Constants
Character Constants
Integer
Floating-point
Single character
String

Variables

- It is a data name that can be used to store a data value.
- Unlike constants, a variable may take different values in memory during execution.
- Can have only one value assigned to it at any given point of time during the execution of the program
- Variable names follow the naming convention for **identifiers**.
  - Examples: temp, speed, name1, name2, current

- Variables are stored in memory
- Memory is a list of consecutive storage locations, each having a unique address
- A variable is like a bin
  - The ***content*** 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***

Declaration of Variables

- There are two purposes:
  - It tells the compiler what the variable name is.
  - It specifies what type of data the variable will hold.

- General syntax:

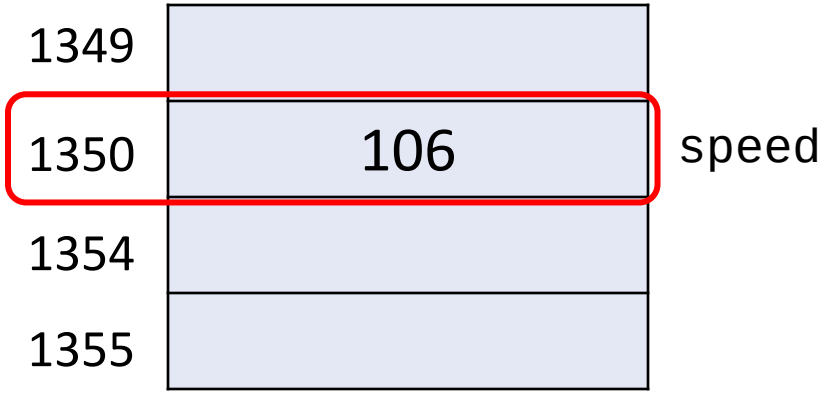
  <data-type> <variable-list>;

- Examples:

```
int velocity, distance;
int a, b, c, d;
float temp;
char flag, option;
```

Address and Content

Every variable has an address (in memory), and its contents.

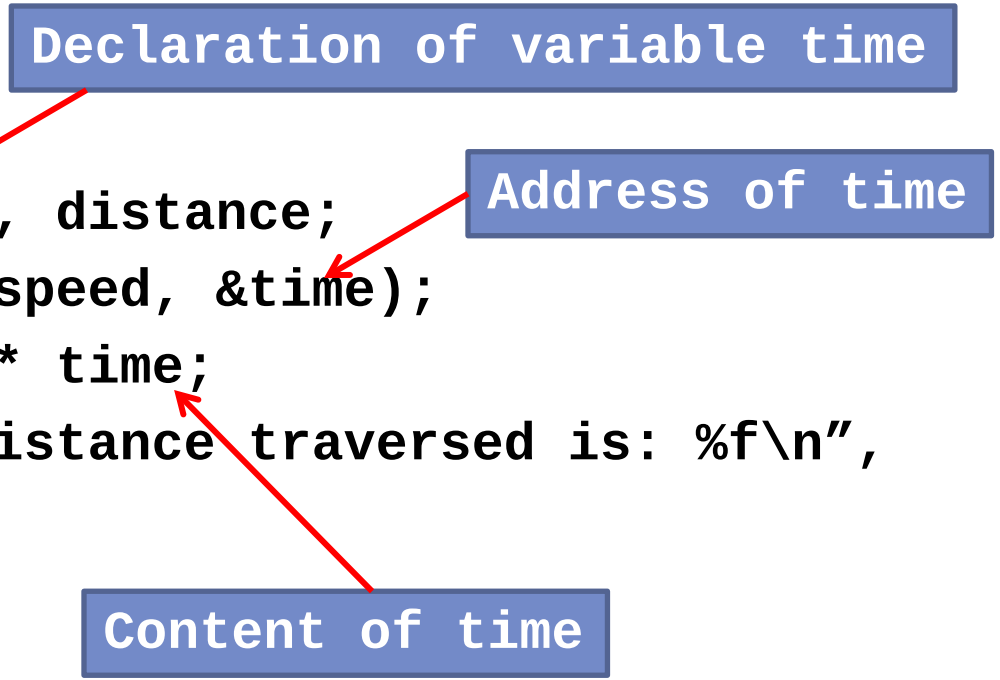



```
int speed;
speed=106;
```

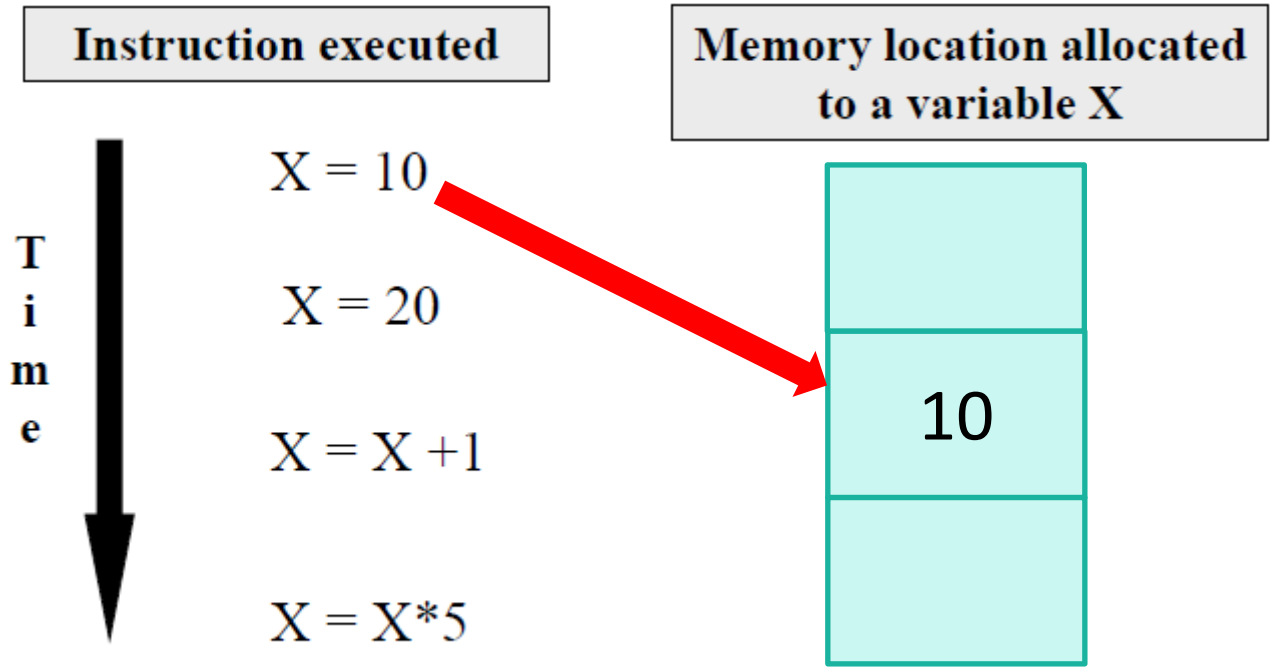
```
speed  ⇒  106
&speed ⇒  1350
```

An Example

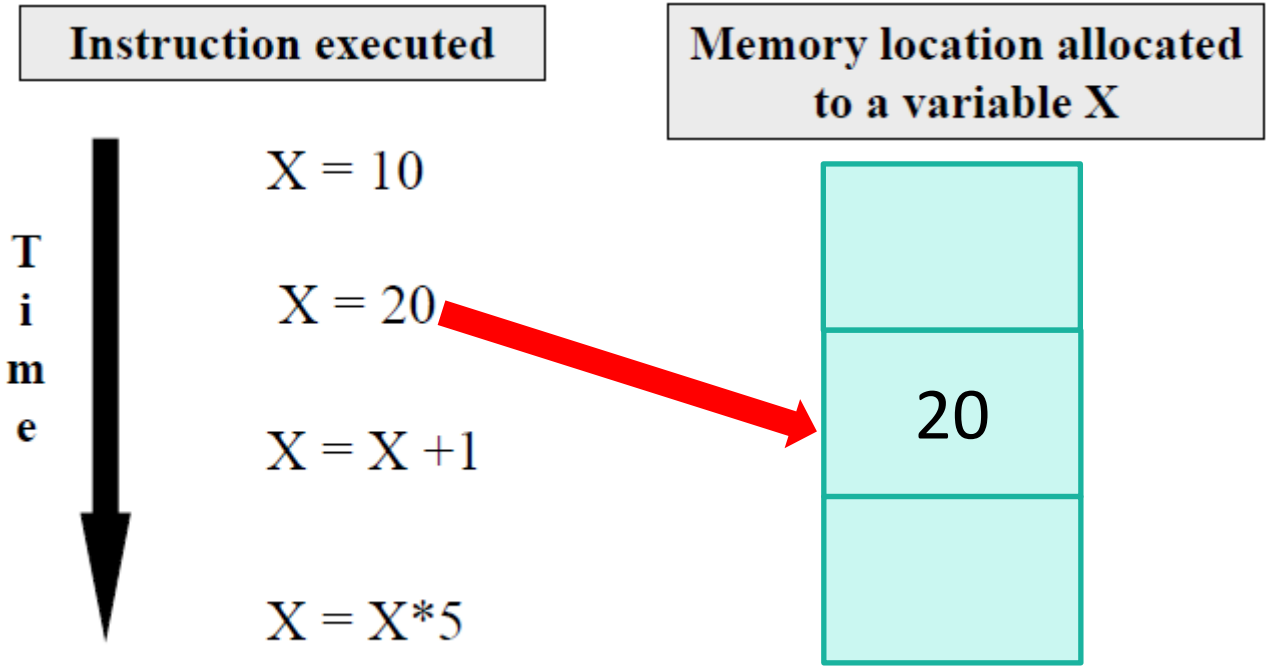

```
#include <stdio.h>
int main()
{
        float speed, time, distance;
        scanf (“%f %f”, &speed, &time);
        distance = speed * time;
        printf (“\n The distance traversed is: %f\n”,
distance);
        return 0;
}
```



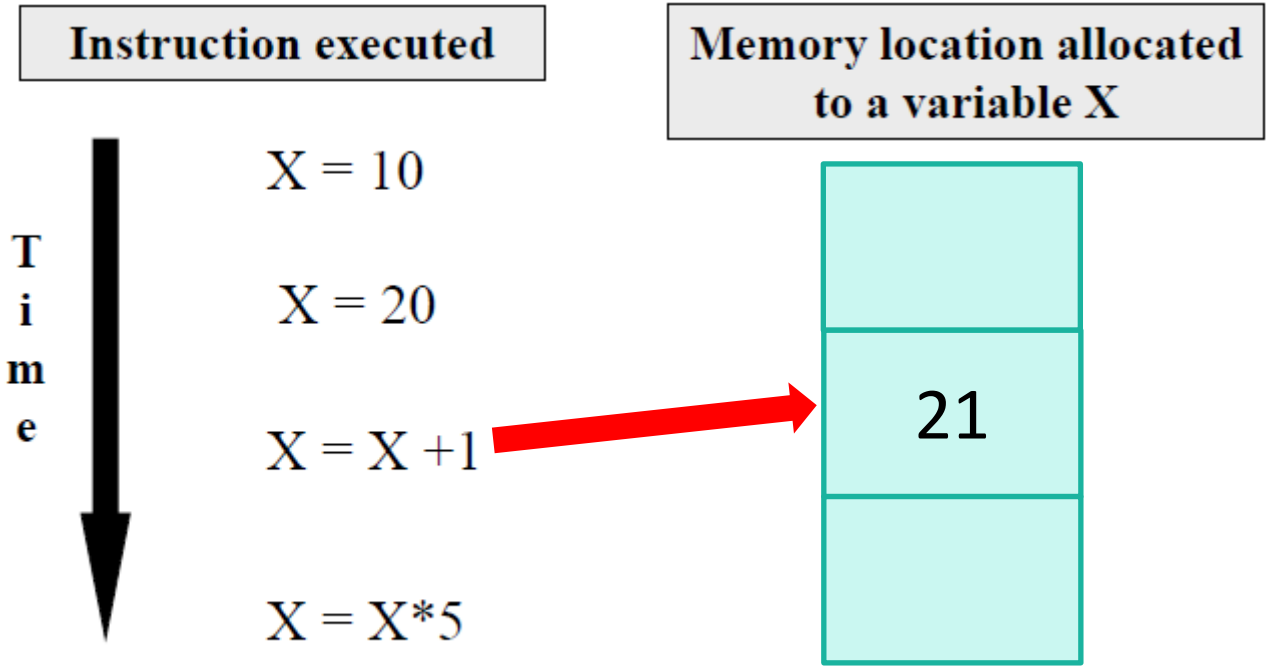
Variables in Memory: More Example



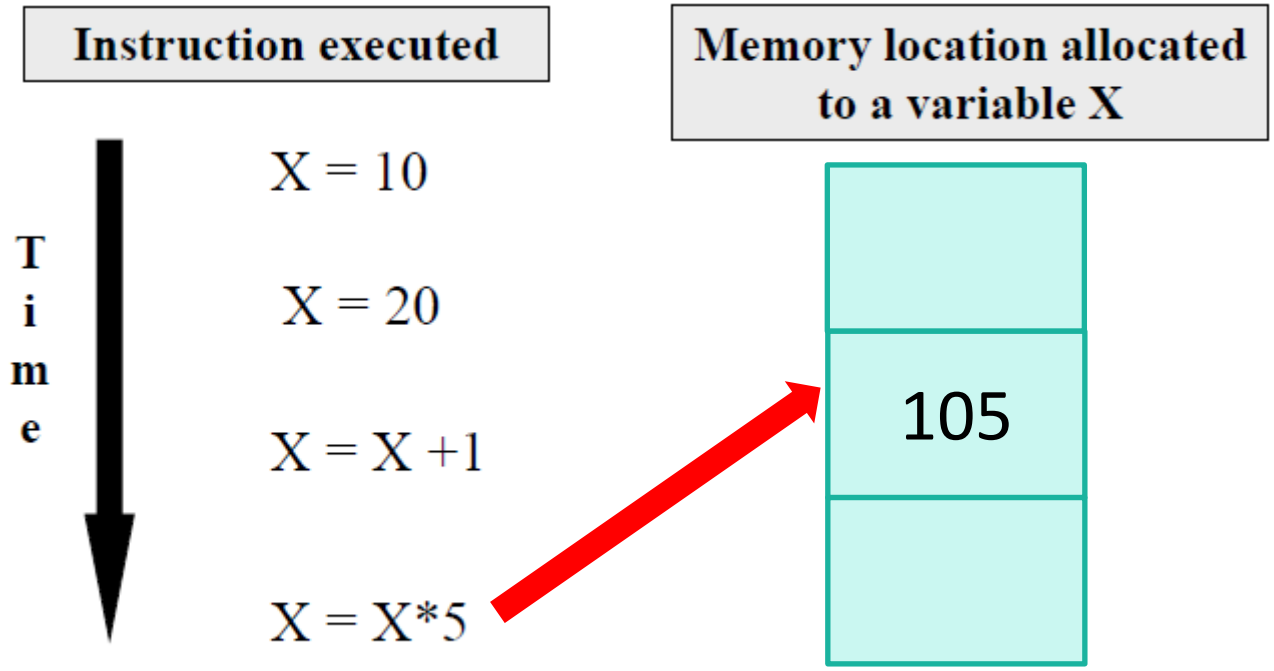
Variables in Memory: More Example



Variables in Memory: More Example



Variables in Memory: More Example



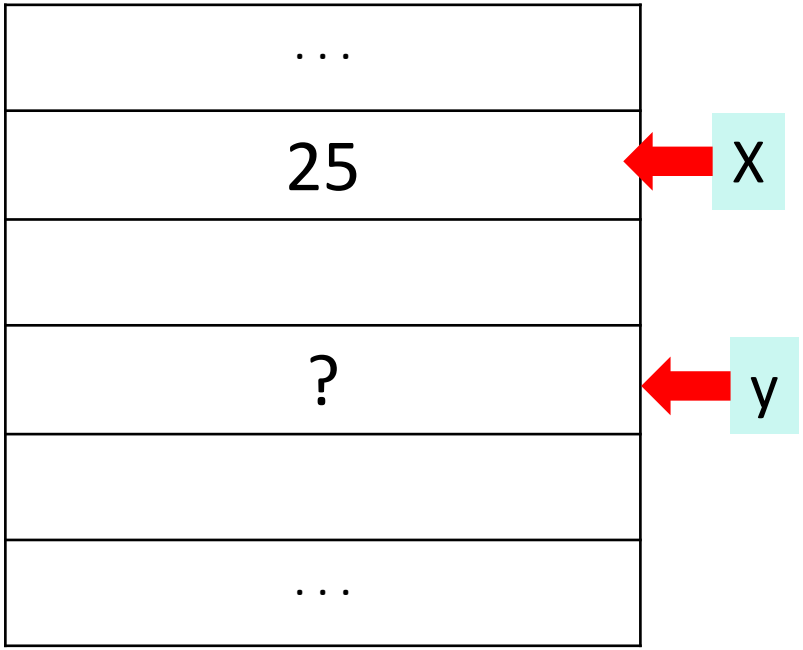
Variables in Memory: More Example

X = 25

Y=45

X = Y+3

Y=X/6



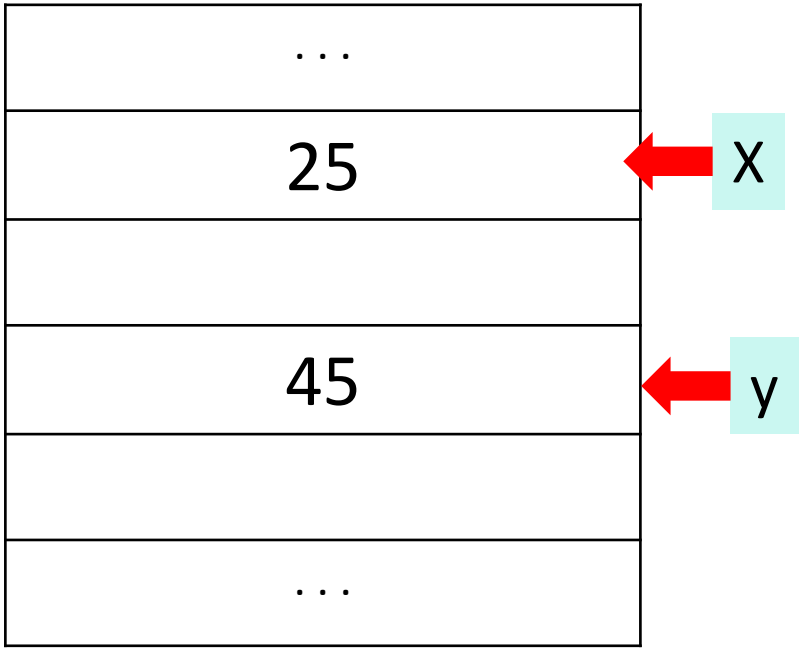
Variables in Memory: More Example

X = 25

Y=45

X = Y+3

Y=X/6



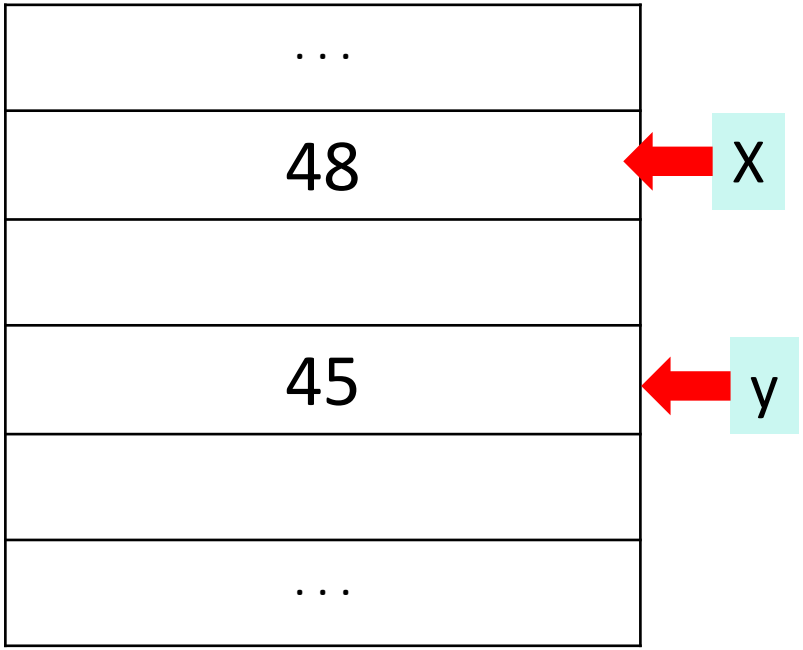
Variables in Memory: More Example

X = 25

Y=45

X = Y+3

Y=X/6



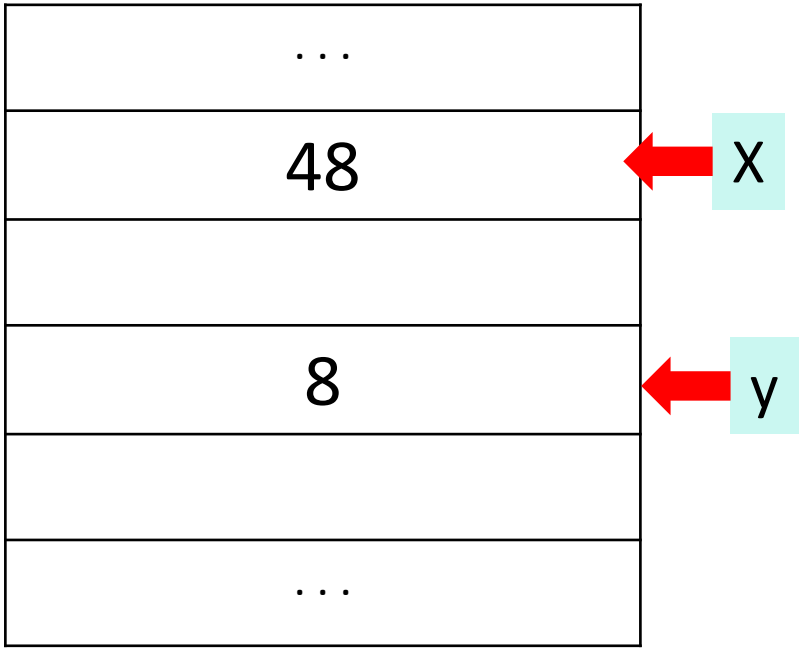
Variables in Memory: More Example

X = 25

Y=45

X = Y+3

Y=X/6



Basic Data Types in *C*

- **int** : integer quantity
  - Typically occupies 4 bytes (32 bits) in memory.
- **char** : single character
  - Typically occupies 1 byte (8 bits) in memory.
- **float** : floating-point number (a number with a decimal point)
  - Typically occupies 4 bytes (32 bits) in memory.
- **double** : double-precision floating-point number
- ***Precision refers to the number of significant digits after the decimal point.***

Size of data types may vary depending on machine/OS type.
You can use the sizeof() operator to get the size
sizeof(char) will give 1,
sizeof(int) will give 4 and so on

Augmented Data Type

- Some of the basic data types can be augmented by using certain data type qualifiers:
  - short
  - long
  - signed
  - unsigned
- Typical examples:
  - short int
  - long int
  - unsigned int

Integer Type

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

Floating-point type

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

The size of the various data types depends on machine configuration

Example


```
#include <stdio.h>
int main()
{
        float x, y;
         int a, b = 20;
        scanf(“%f%f%d”,&x, &y, &a);
        printf(“%f plus %f is %f\n”, x, y, x+y);
        printf(“%d minus %d is %d\n”, a, b, a-b);
        return 0;
}
```

Type casting


```
#include <stdio.h>
int main ()
{
	int n;
	scanf("%d",&n);
	printf("%d\n",1/n);
	return 0;
}
```

```
#include <stdio.h>
int main ()
{
	int n;
	scanf("%d",&n);
	printf("%f\n",1/n);
	return 0;
}
```

The division 1/n is of integers (quotient).
The format %d is for printing integers

Type casting


```
#include <stdio.h>
int main ()
{
        int n;
        scanf("%d",&n);
        printf("%f\n",1.0/n);
        return 0;
}
```

```
#include <stdio.h>
int main ()
{
        int n;
        float x;
        scanf("%d",&n);
        x=(float)1/n;
        printf("%f\n",x);
        return 0;
}
```

Type casting

Integer to Real

```
int a=10;
float b;
b=(float)a;
```

Real to Integer

```
int a;
float b=3.14;
a=(int)b;
```

Real to Real

```
float b;
double c=3.14;
b=(float)c;
```

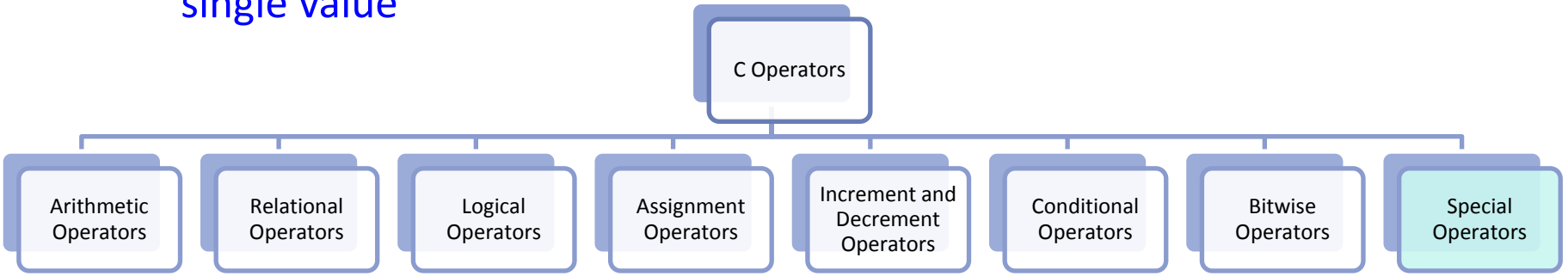
Real to Real

```
float b;
double c;
c=22.0/7.0;
b=(float)c;
```

Operators and Expressions in C

Operators in C

- An operator is a symbol that tells computer to perform certain mathematical or logical manipulations over the data
- Usually a part of mathematical or logical expressions
  - Expression: A sequence of operands and operators that reduces to a single value



Arithmetic Operators

Arithmetic Operators	Meaning
+	Addition or Unary plus
-	Subtraction or Unary minus
*	Multiplication
/	Division
%	Modulo Division

Examples: Arithmetic expressions

- `v = u + f * t;` → `v = u+(f*t);`
- `X = x * y / z;` → `X = (x*y)/z;`
- `A = a + b – c * d / e;` → `A = ((a+b)-((c*d)/e));`
- `A = -b * c + d % e;` → `A = (((-b)*c)+(d%e));`

Note: The modulo division operator % cannot be used on floating point data

Relational Operators

Relational Operators	Meaning
<	is less than
<=	is less than or equal to
>	is greater than
>=	is greater than or equal to
==	is equal to
!=	is not equal to

- Used to compare two quantities
- The value of a relational expression is either *one* or *zero*
- Frequently used in decision statement, such as **if**, **while** to decide the course of action
- **Arithmetic operators have higher priority over relational operators**

Relational Operators

An expression containing a relational operator is termed as ***relational expression***
Generic form: ***arithmetic-expression1*** ***relational-operator*** ***arithmetic-expression2***

```
#include<stdio.h>
int main()
{int x = 20;
int y=3;
float a=20.3;
if ( x > y )                          /* 20 > 3 → True */
        printf (“%d is larger\n”, x);
if ( x + x > y * 6 )                  /* 20+20 > 3*6 → (20+20)>(3*6) → True */
        printf(“Double of %d is larger than 6 times %d\n”,x,y);
if ( x > a )
        printf(“%d is larger than %f\n”,x, a);
else
        printf(“%d is smaller than %f”,x, a);
return 0;}
```

Output

```
20 is larger
Double of 20 is larger than 6 times 3
20 is smaller than 20.299999
```

Logical Operators

- C has following *three* logical operators

  && meaning logical AND

  || meaning logical OR

  ! meaning logical NOT

  - The second operand will not be evaluated if the first one is zero
  - The second operand will not be evaluated if the first one is non-zero

The logical operators && and || are used when we want to test more than one conditions and make decisions

E.g. a>b && b!=0 ← *Logical expression* or *Compound relational expression*

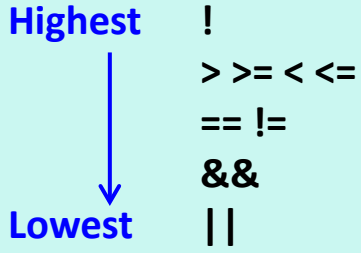
Logical Operators

Operand-1 (op1)	Operand-2 (op2)	op1 && op2	op1 || op2
FALSE (or 0)	FALSE (or 0)	FALSE (or 0)	FALSE (or 0)
FALSE (or 0)	TRUE (or non-zero)	FALSE (or 0)	TRUE (or 1)
TRUE (or non-zero)	FALSE (or 0)	FALSE (or 0)	TRUE (or 1)
TRUE (or non-zero)	TRUE (or non-zero)	TRUE (or 1)	TRUE (or 1)

```
int x = 20;
int y=3;
float a=20.3;

if((x>y) && (x>a))
                printf("X is largest");
if((x>y) || (x>a))
                printf("X is not smallest");
```

Relative precedence of the relational and logical operators



Assignment Operator

- Used to assign the result of an expression to a variable
- Assignment operator: '='

- Shorthand assignment operator

$$v\ op = exp;$$

v is a variable, exp is an expression, and op is a C binary arithmetic operator

$v\ op = exp;$ is equivalent to $v = v\ op\ (exp);$

E.g.: $x \mathrel{+}= y + 1;$ is equivalent to $x = x + (y + 1);$

Assignment Operator

Statement with simple assignment operator	Statement with shorthand assignment operator
$a = a + b$	$a \mathrel{+}= b$
$a = a - b$	$a \mathrel{-}= b$
$a = a * (b + 1)$	$a \mathrel{*}= b + 1$
$a = a/(b + 1)$	$a \mathrel{/}= b + 1$
$a = a\%b$	$a\% = b$

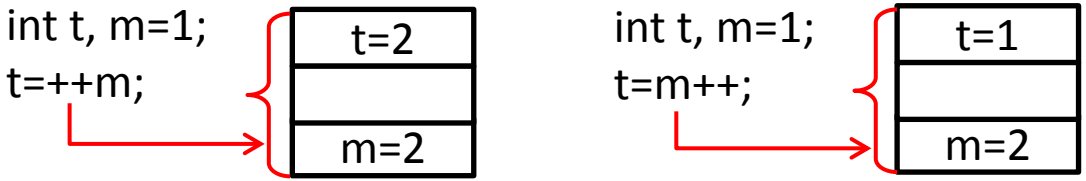
- Shorthand assignment operator advantages
  - Concise; Easier to read and write
  - Statement becomes more efficient

Increment and Decrement Operators

- Increment (++) Operation means op=op+1;
  - Prefix operation (++i) or Postfix operation (i++)
- Decrement (--) Operation means op=op-1;
  - Prefix operation (--i) or Postfix operation (i--)

- Precedence
  - Prefix operation : First increment / decrement and then used in evaluation
  - Postfix operation : Increment / decrement operation after being used in evaluation

- Example



More examples

Initial values: a = 10; b = 20;

x = 50 + ++a;

a = 11, x = 61

Initial values: a = 10; b = 20;

x = 50 + a++;

x = 60, a = 11

Initial values: a = 10; b = 20;

x = a++ + --b;

b = 19, x = 29, a = 11

```
#include <stdio.h>
int main()
{
        int a=10,x;
        x= a++ - ++a;
        printf("\n x=%d, a=%d\n",x,a);

        x= a++ + ++a;
        printf("\n x=%d, a=%d\n",x,a);

return 0;
}
```

What would be the output?

Conditional Operators

- Ternary operator
- Conditional Expression: $exp1\ ?\ exp2\ :\ exp3$
- *exp1, exp2,* and *exp3* are expressions

- *Example*

```
a=10;
b=15;
x= (a>b) ? a : b
```

Here, *x* will be assigned the value of *b*

Equivalent to:

```
if(a > b)
        x=a;
else
        x=b;
```

Bitwise Operators

- Used for manipulation of data at bit level
- These may not be applied to **float** and **double**

Bitwise Operator	Meaning
&	bitwise AND
|	bitwise OR
^	bitwise exclusive OR
≪	Shift left
≫	Shift right

Special Operators

- Comma operator
  - Example: `value =(x = 21, y = 14, x + y)`
  - Usually used in for loop and while loop
  - Comma operator has the lowest precedence of all operators
- sizeof operator
  - Compile time operator
  - Returns the number of bytes the operand occupies
  - Example: `m=sizeof(x); n=sizeof(char);`
- Pointer operator (**&** and ***)
- Member selection operator (**.** and **->**)

Operator Precedence and Associativity

- Precedence
  - Each operator in C has a precedence associated with it
  - Decides the order in which different operators are applied
  - There are distinct levels of precedence
  - The operators at the higher level of precedence are evaluated first

- Associativity
  - Decides the order in which multiple occurrence of the same level operator are applied
  - *Left to right* or *Right to left*

Operator Precedence and Associativity

Operator	Description	Precedence level	Associativity
( ) [ ] .	Parentheses: grouping or function call Brackets (array subscript) Dot operator (Member selection via object name)	1	Left to Right
+ - ++ -- ! ~ * & (datatype) sizeof	Unary plus Unary minus Prefix increment/decrement Logical NOT One's complement Indirection Address (of operand) Type cast Determine size in bytes on this implementation	2	Right to Left
* / %	Multiplication Division Modulus	3	Left to Right
+ -	Addition Subtraction	4	Left to Right

Operator Precedence and Associativity [contd.]

<< >>	Left shift Right shift	5	Left to Right
< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	6	Left to Right
== !=	Equal to Not equal to	7	Left to Right
&	Bitwise AND	8	Left to Right
^	Bitwise XOR	9	Left to Right
|	Bitwise OR	10	Left to Right
&&	Logical AND	11	Left to Right
||	Logical OR	12	Left to Right
?:	Conditional operator	13	Right to Left
= *= /= %= += -= &= ^= |= <<= >>=	Assignment operators	14	Right to Left
,	Comma operator	15	Left to Right

Questions?

For the beginners

- Write a C program to read two integer numbers $\boldsymbol{a}$ and $\boldsymbol{b}$ from the keyboard. Now print on screen the values of $\boldsymbol{a}+\boldsymbol{b}$, $\boldsymbol{a}-\boldsymbol{b}$, $\boldsymbol{a}*\boldsymbol{b}$, $\boldsymbol{a}/\boldsymbol{b}$, and $\boldsymbol{a}\%\boldsymbol{b}$

- Write a C program to read two integer numbers $\boldsymbol{a}$ and $\boldsymbol{b}$ from the keyboard. Now print on screen the values of $\boldsymbol{a}\&\boldsymbol{b}$, $\boldsymbol{a}|\boldsymbol{b}$, $\boldsymbol{a}\text{^}\boldsymbol{b}$, $\boldsymbol{a} << \boldsymbol{b}$, and $\boldsymbol{a} >> \boldsymbol{b}$

- Write a C program to read temperature in degree Celsius and print the corresponding Fahrenheit value

Solutions

Write a C program to read two integer numbers a and b from the keyboard. Now print on screen the values of $a + b$, $a - b$, $a * b$, a/b, and $a\%b$


```
#include<stdio.h>
int main()
{
        int a,b;
        printf("Enter the value of a:");
        scanf("%d",&a);
        printf("Enter the value of b:");
        scanf("%d",&b);
        printf("a+b=%d, a-b=%d, a*b=%d, a/b=%d, a%%b=%d",a+b,a-b,a*b,a/b,a%b);
        return 0;
}
```

Output:

```
Enter the value of a:10
Enter the value of b:7
a+b=17, a-b=3, a*b=70, a/b=1, a%b=3
```

Write a C program to read two integer numbers a and b from the keyboard. Now print on screen the values of a&b, a|b, a^b, a << b, and a >> b


```
#include<stdio.h>
int main()
{
        int a,b;
        printf("Enter the value of a:");
        scanf("%d",&a);
        printf("Enter the value of b:");
        scanf("%d",&b);
        printf("a&b=%d, a|b=%d, a^b=%d, a<<b=%d, a>>b=%d",a&b,a|b,a^b,a<<b,a>>b);
        return 0;
}
```

Output:

```
Enter the value of a:10
Enter the value of b:2
a&b=2, a|b=10, a^b=8, a<<b=40, a>>b=2
```

Write a C program to read temperature in degree Celsius and print the corresponding Fahrenheit value


```
#include<stdio.h>
int main()
{
        float c, f;
        printf("Enter the temperature value in degree Celsius: ");
        scanf("%f",&c);
        f=c*(9.0/5)+32;
        printf("The temperature in degree Fahrenheit is %f",f);
        return 0;
}
```

Output

```
Enter the temperature value in degree Celsius:32
The temperature in degree Fahrenheit is 89.599998
```