First Year First Semester Course
M.Tech. (CS) [Batch 2021-23]

Lecture #02

Introduction to Programming

Introduction to C: Syntax, Basic Constructs

Course Instructor:

Dr. Monidipa Das

DST-INSPIRE Faculty

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

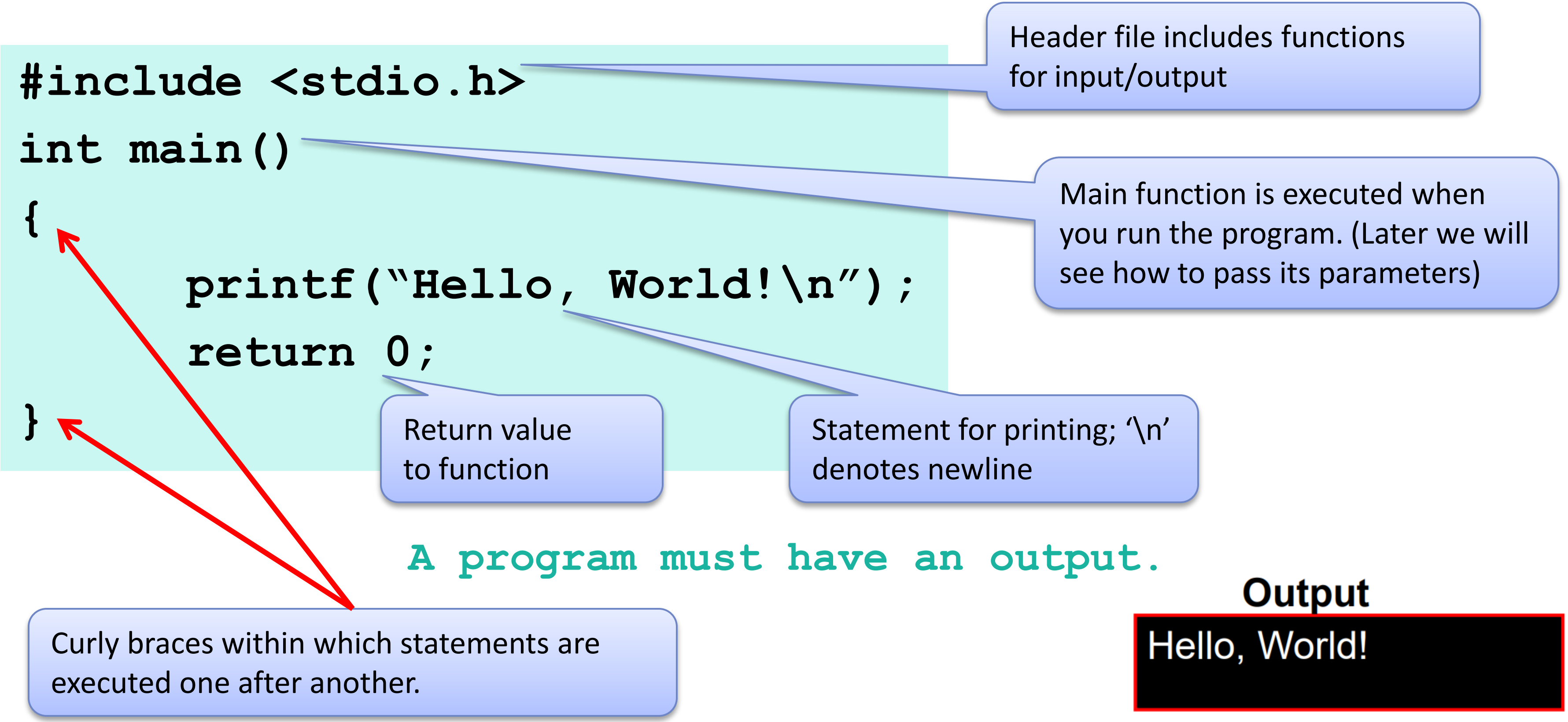
Indian Statistical Institute (ISI) Kolkata, India

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

Curly braces within which statements are executed one after another.

A program must have an output.

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;
}
```

```
2. $ cc hello.c
$

3. $ ./a.out
Hello World
```

OR

```
2. $ gcc –o hello hello.c

3. $ ./hello
```

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;
}
```

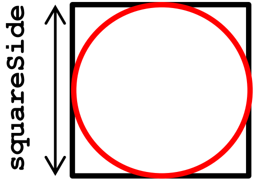
When you run the program

```
```

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:

<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><</td><td>></td><td>/</td></tr>
</table>

whitespace (space, tab)

A C program should not contain anything else

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**

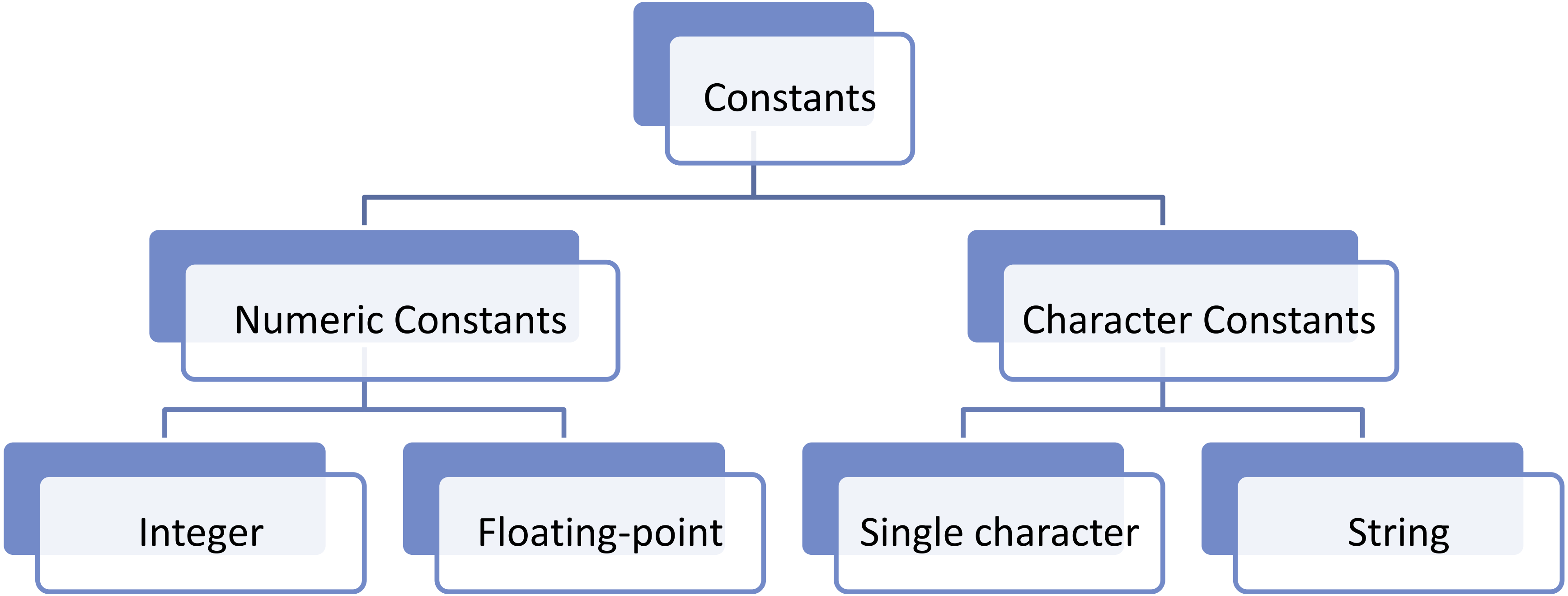
```
10abc
“hello”
simple interest
(area)
%rate
```

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:

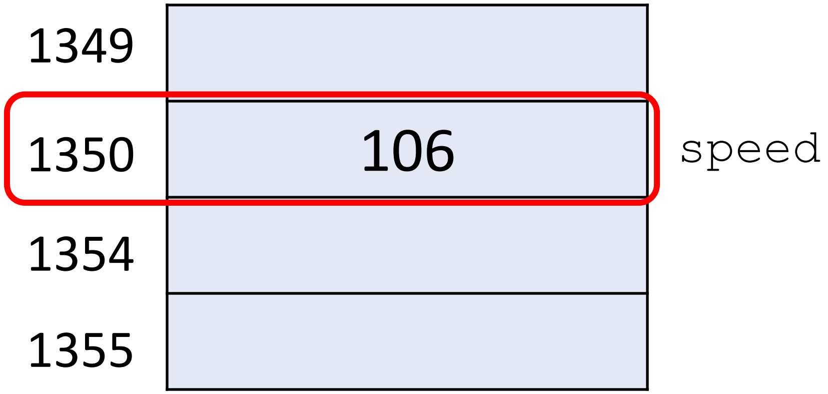
  <data-type> <variable-list>;

- 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.



1349		
1350	106	speed
1354		
1355		

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

speed → 106

&speed → 1350

An Example

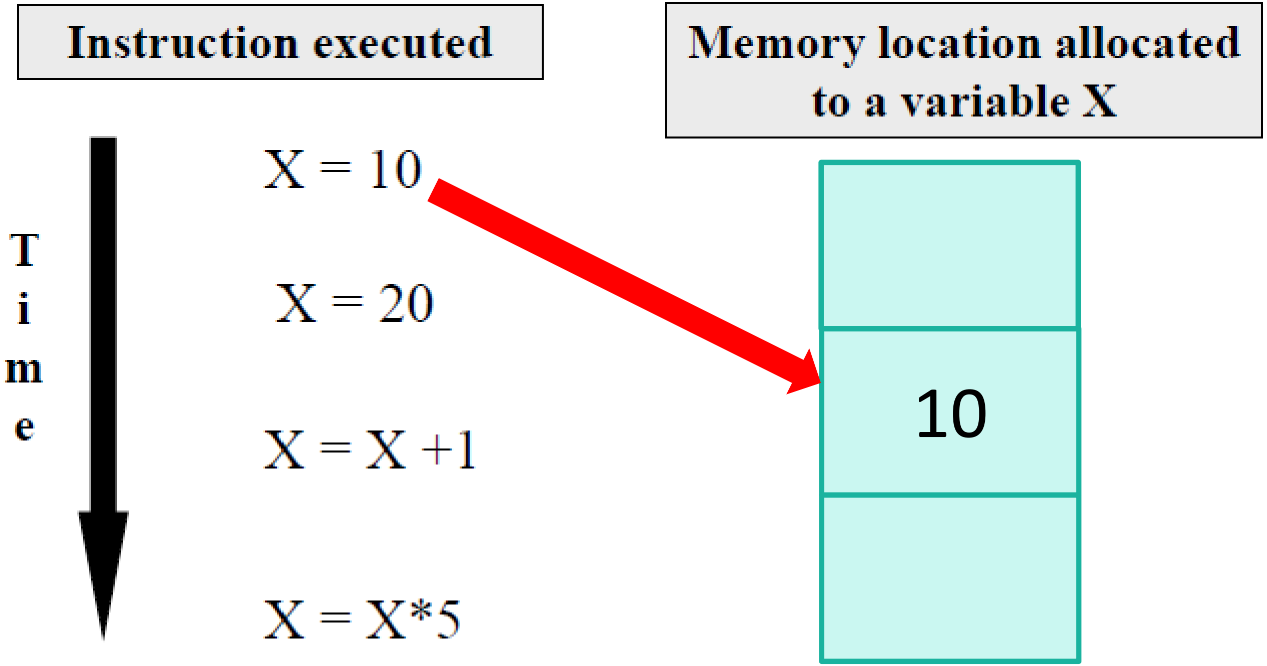
```
#include <stdio.h>
int main()
{
      float speed, time, distance;
      scanf (“%f %f”, &speed, &time);
      distance = speed * time;
      printf (“\n The distance traversed is: %f\n”,
distance);
      return 0;
}
```

Declaration of variable time

Address of time

Content of time

Variables in Memory: More Example



Instruction executed

Memory location allocated to a variable X

Time

X = 10

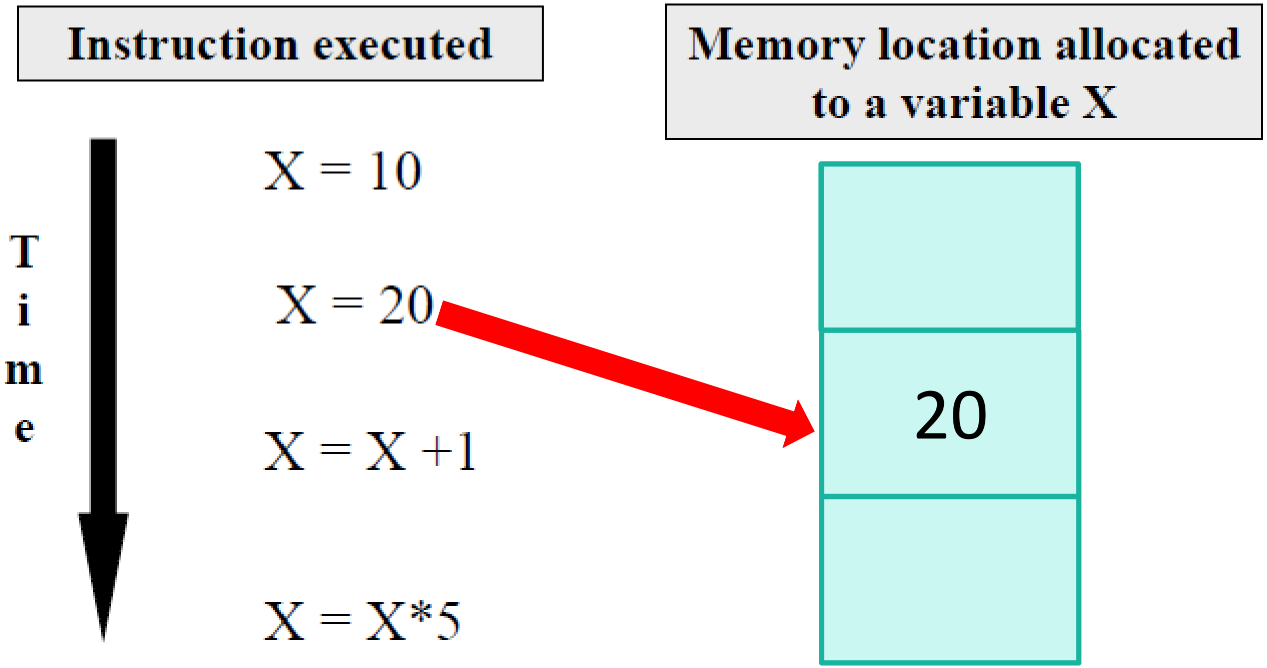
X = 20

X = X +1

X = X*5

10

Variables in Memory: More Example



Instruction executed

Memory location allocated to a variable X

Time

X = 10

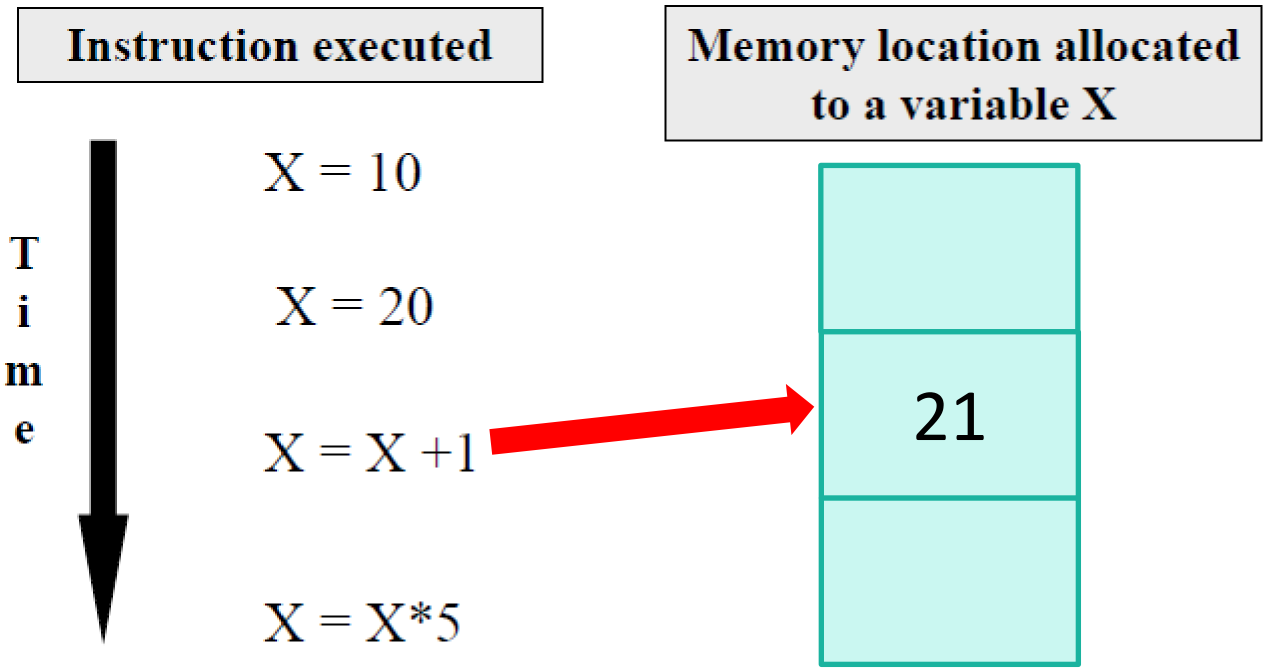
X = 20

X = X +1

X = X*5

20

Variables in Memory: More Example



Instruction executed

Memory location allocated to a variable X

Time

X = 10

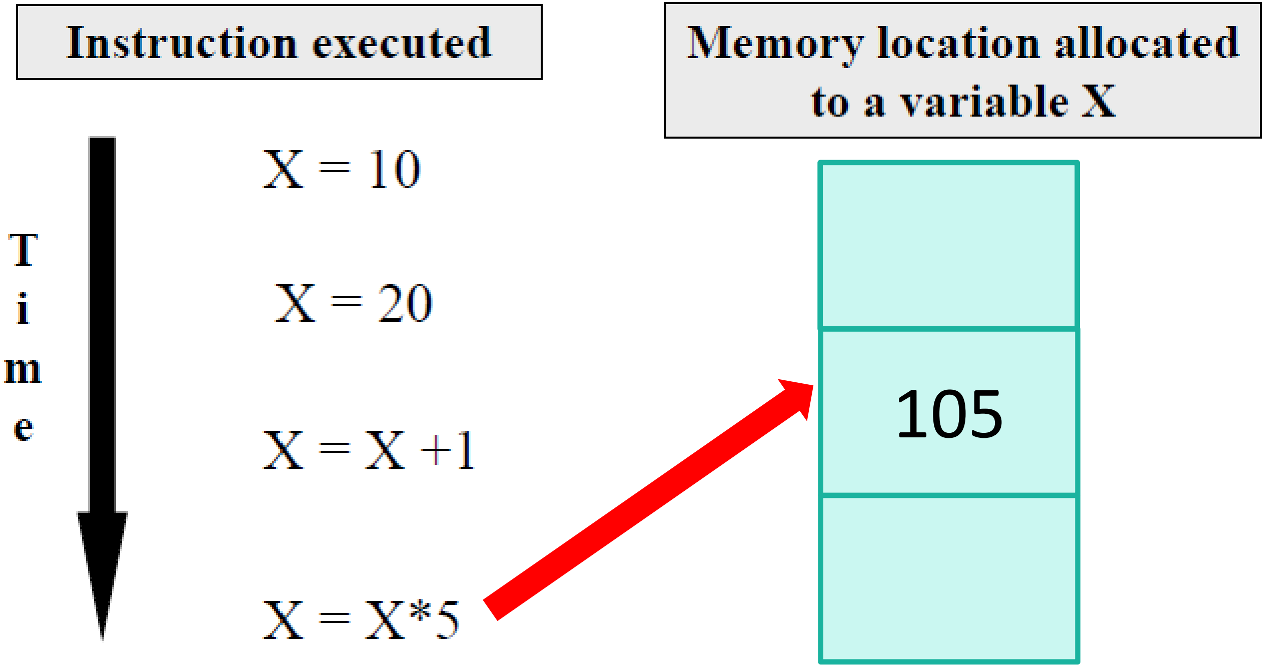
X = 20

X = X +1

X = X*5

21

Variables in Memory: More Example

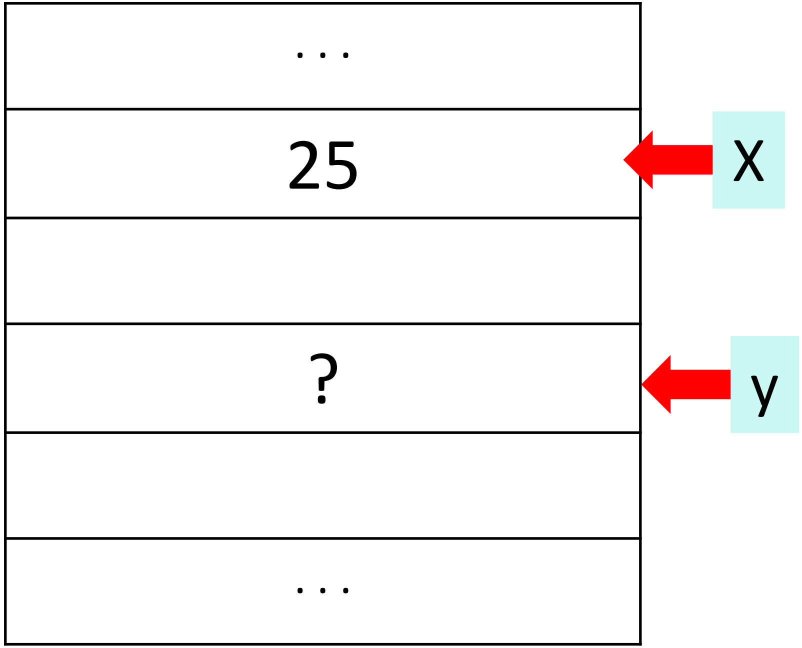


Instruction executed

Memory location allocated to a variable X

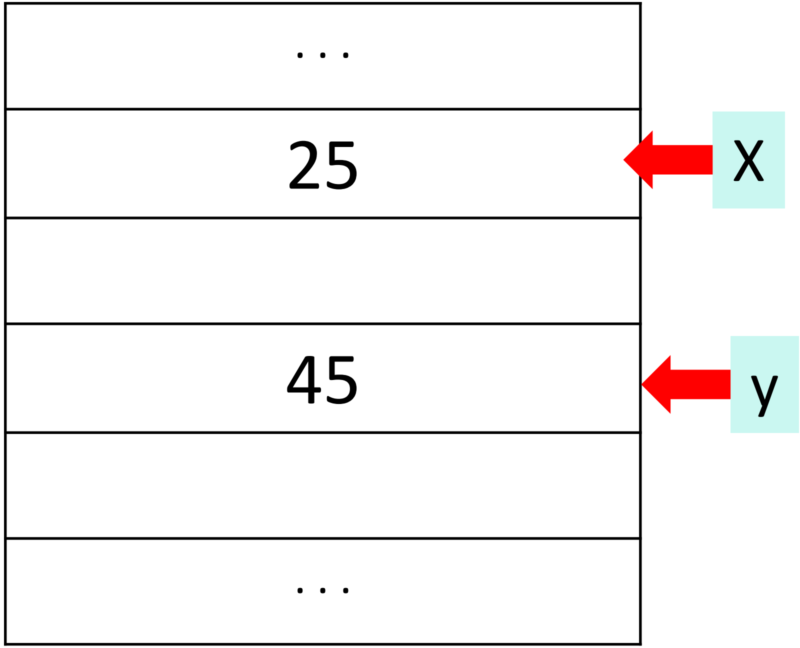
Variables in Memory: More Example

```
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
```



. . .

25

X

45

y

. . .

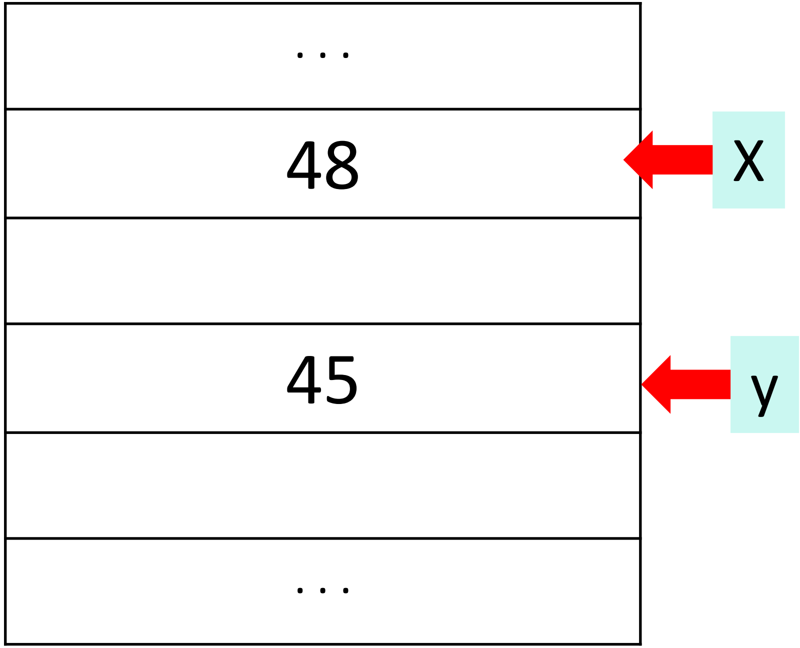
Variables in Memory: More Example

X = 25

Y=45

X = Y+3

Y=X/6



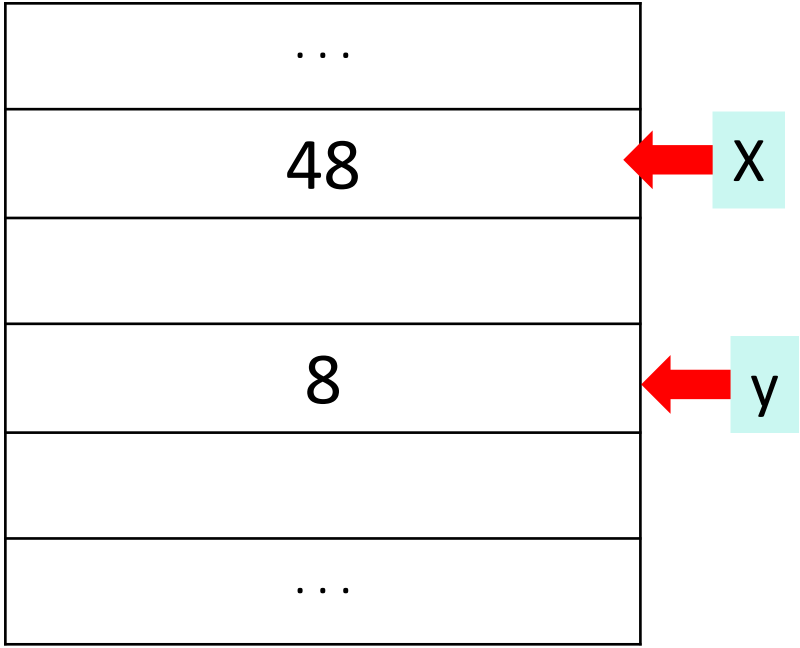
Variables in Memory: More Example

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

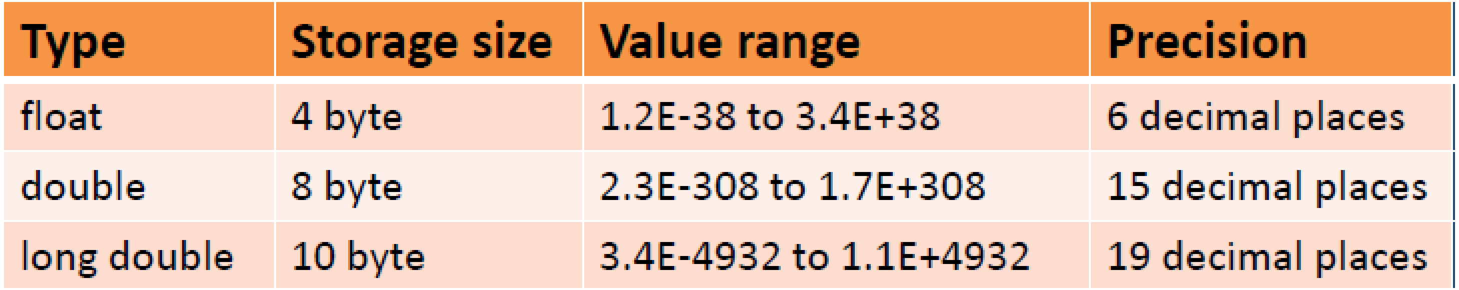
Augmented Data Type

- 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;
```

Questions?