Types of variable

- We must *declare* the *type* of every variable we use in C.
- Every variable has a *type* (e.g. `int`) and a *name*.
- This prevents some bugs caused by spelling errors (misspelling variable names).
- Declarations of types should always be together at the top of main or a function (see later).
- Other types are `char`, `signed`, `unsigned`, `long`, `short` and `const`.

Identifiers and Keywords

- Identifiers
  - Names given to various program elements (variables, constants, functions, etc.)
  - May consist of *letters*, *digits* and the *underscore* ('_') character, with no space 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

  my-name

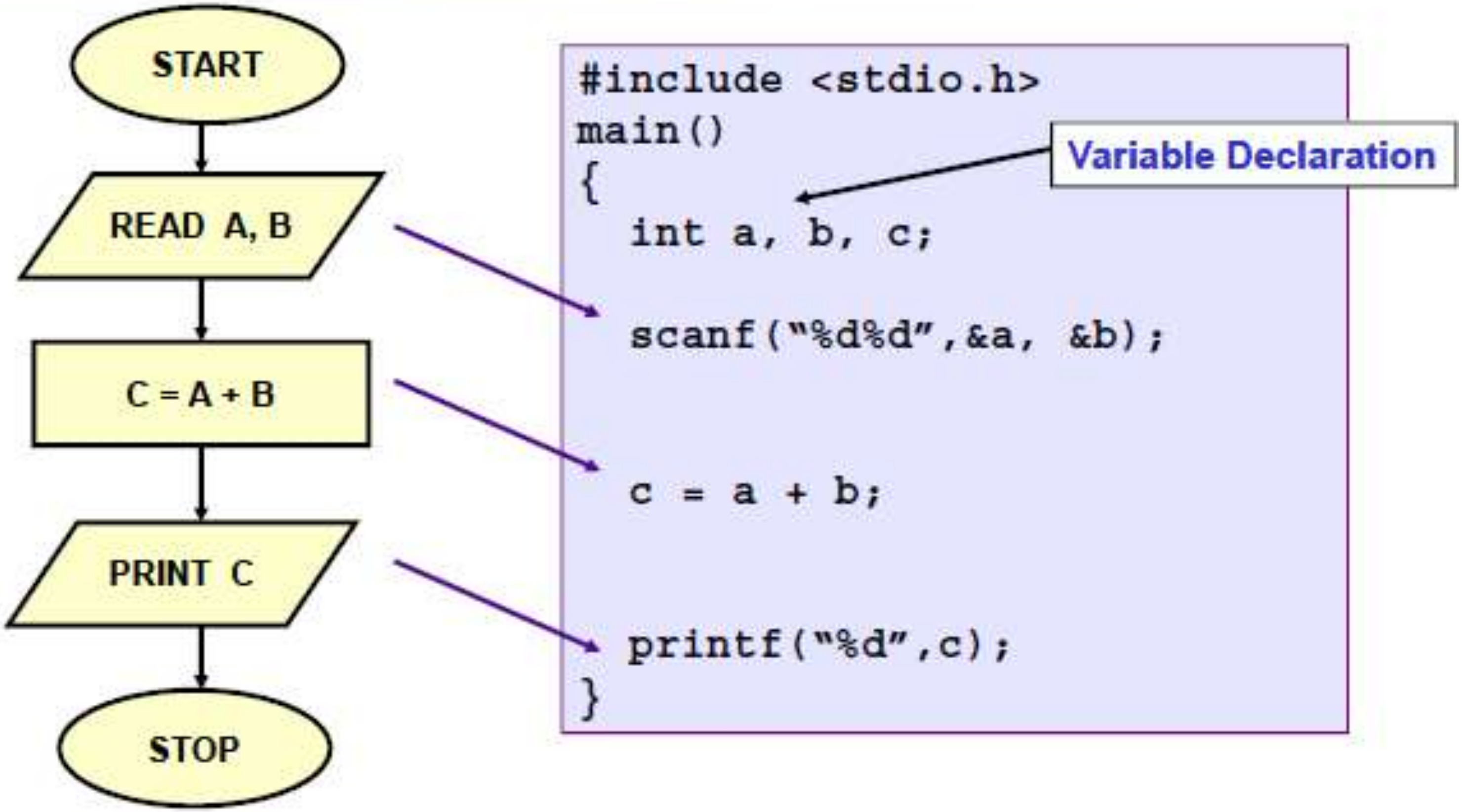
  “hello”

  simple interest

  (area)

  %rate

Another Example: *Adding two numbers*



```
#include <stdio.h>
main()
{
  int a, b, c;

  scanf("%d%d",&a, &b);


  c = a + b;


  printf("%d",c);
}
```

Variable Declaration

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 list of argument *declarations*.
  - A *compound statement*, which comprises the remainder of the function.

Desirable Programming Style

- Clarity
  - The program should be clearly written.
  - It should be easy to follow the program logic.

- Meaningful variable names
  - Make variable/constant names meaningful to enhance program clarity.
    - ‘area’ instead of ‘a’
    - ‘radius’ instead of ‘r’

- Program documentation
  - Insert comments in the program to make it easy to understand.
  - Never use too many comments.

- Program indentation
  - Use proper indentation.
  - Structure of the program should be immediately visible.

Indentation Example: *Good Style*

```
#include <stdio.h>

/* FIND THE LARGEST OF THREE NUMBERS */

main()
{
        int   a, b, c;

        scanf(“%d%d%d”, &a, &b, &c);

        if  ((a>b) && (a>c))
                printf(“\n Largest is %d”, a);
        else
                if  (b>c)
                                printf(“\n Largest is %d”, b);
                else
                                printf(“\n Largest is %d”, c);
}
```

Indentation Example: *Bad Style*

```
#include <stdio.h>

/* FIND THE LARGEST OF THREE NUMBERS */
main()
{
int   a, b, c;
scanf("%d%d%d", &a, &b, &c);
if  ((a>b) && (a>c))
printf("\n Largest is %d", a);
    else
if  (b>c)
  printf("\n Largest is %d", b);
else
printf("\n Largest is %d", c);
}
```

Data Types in C

int :: integer quantity

Typically occupies 4 bytes (32 bits) in memory.

char :: single character

Typically occupies 1 bye (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

Contd.

- 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

Some Examples of Data Types

- int

  0, 25, -156, 12345, –99820

- char

  ‘a’, ‘A’, ‘*’, ‘/’, ‘ ’

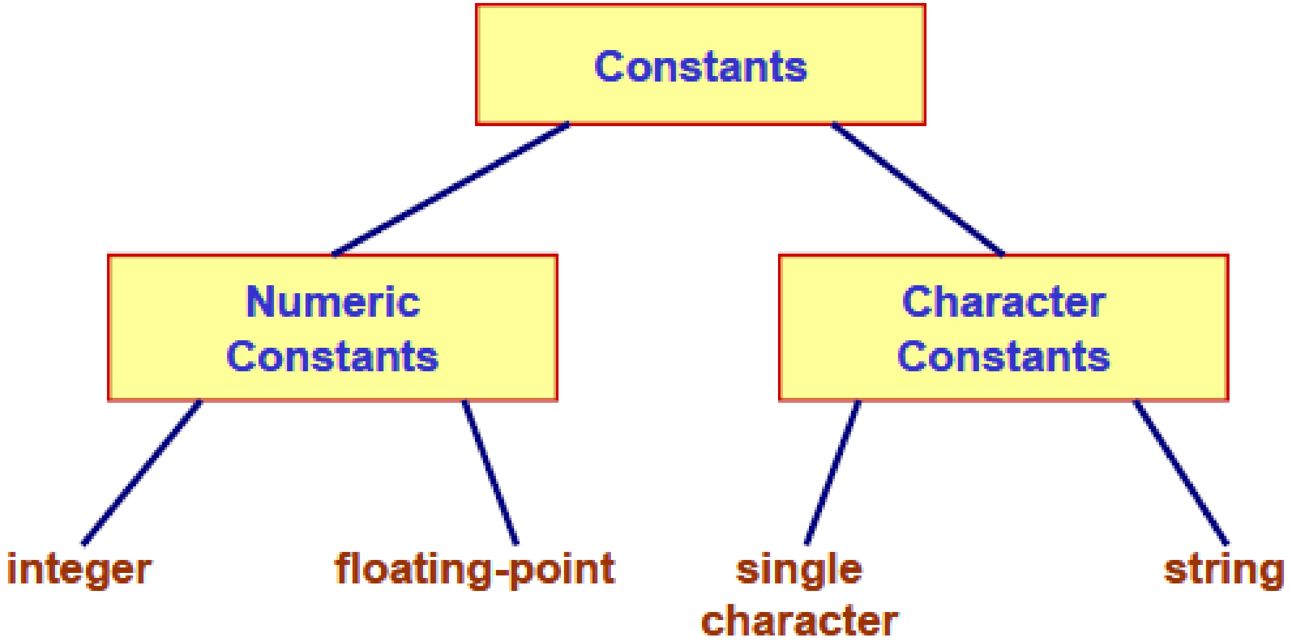
- float

  23.54, –0.00345, 25.0

  2.5E12, 1.234e-5

E or e means “10 to the power of”

Constants



Constants

Numeric Constants

Character Constants

integer

floating-point

single character

string

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.

- Maximum and minimum values (for 32-bit representations)

  Maximum :: 2147483647

  Minimum :: – 2147483648

Floating-point Constants

- Can contain fractional parts.

- Very large or very small numbers can be represented.

  23000000 can be represented as 2.3e7

- Two different notations:

  1. Decimal notation

     25.0, 0.0034, .84, -2.234

  2. Exponential (scientific) notation

     3.45e23, 0.123e-12, 123E2

e means “10 to the power of”

Single 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

String Constants

- Sequence of characters enclosed in double quotes.
  - The characters may be letters, numbers, special characters and blank spaces.

- Examples:

  “nice”, “Good Morning”, “3+6”, “3”, “C”

- Differences from character constants:
  - ‘C’ and “C” are not equivalent.
  - ‘C’ has an equivalent integer value while “C” does not.

Declaration of Variables

- There are two purposes:
  1. It tells the compiler what the variable name is.
  2. 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;
  ```

An Example

```
#include <stdio.h>
main()
    {
        float  speed, time, distance;

        scanf (“%f %f”, &speed, &time);
        distance = speed * time;
        printf (“\n The distance traversed is: \n”, distance);
    }
```

Assignment Statement

- Used to assign values to variables, using the assignment operator (=).

- General syntax:

```
variable_name  =  expression;
```

- Examples:

```
velocity = 20;
b = 15;  temp = 12.5;
A = A + 10;
v = u + f * t;
s = u * t + 0.5 * f * t * t;
```

Contd.

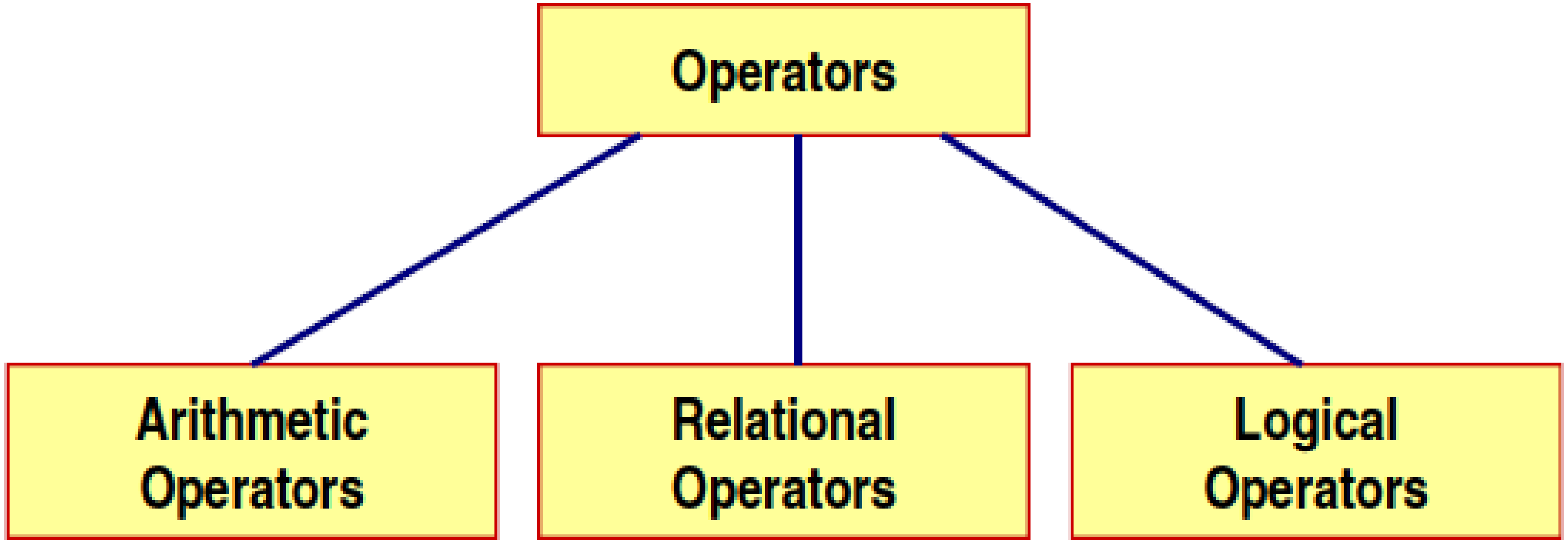
- A value can be assigned to a variable at the time the variable is declared.

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

- Several variables can be assigned the same value using multiple assignment operators.

```
a = b = c = 5;
flag1 = flag2 = ‘y’;
speed = flow = 0.0;
```

Operators in Expressions



Operators

Arithmetic Operators

Relational Operators

Logical Operators

Arithmetic Operators

- Addition :: +
- Subtraction :: –
- Division :: /
- Multiplication :: *
- Modulus :: %

Examples:

```
distance = rate * time ;
netIncome = income - tax ;
speed = distance / time ;
area = PI * radius * radius;
y = a * x * x + b*x + c;
quotient = dividend / divisor;
remain =dividend % divisor;
```

Contd.

- Suppose x and y are two integer variables, whose values are 13 and 5 respectively.

x + y	18
x – y	8
x * y	65
x / y	2
x % y	3

Operator Precedence

- In decreasing order of priority
  1. Parentheses :: ( )
  2. Unary minus :: –5
  3. Multiplication, Division, and Modulus
  4. Addition and Subtraction

- For operators of the *same priority*, evaluation is from *left to right* as they appear.

- Parenthesis may be used to change the precedence of operator evaluation.

Examples: Arithmetic expressions

a + b * c – d / e → a + (b * c) – (d / e)

a * – b + d % e – f → a * (– b) + (d % e) – f

a – b + c + d → (((a – b) + c) + d)

x * y * z → ((x * y) * z)

a + b + c * d * e → (a + b) + ((c * d) * e)

Integer Arithmetic

- When the operands in an arithmetic expression are integers, the expression is called *integer expression*, and the operation is called *integer arithmetic*.

- Integer arithmetic always yields integer values.

Real Arithmetic

- Arithmetic operations involving only real or floating-point operands.

- Since floating-point values are rounded to the number of significant digits permissible, the final value is an approximation of the final result.

  1.0 / 3.0 * 3.0 will have the value 0.99999 and not 1.0

- The modulus operator cannot be used with real operands.

Mixed-mode Arithmetic

- When one of the operands is integer and the other is real, the expression is called a *mixed-mode* arithmetic expression.

- If either operand is of the real type, then only real arithmetic is performed, and the result is a real number.

  25 / 10 → 2

  25 / 10.0 → 2.5

- Some more issues will be considered later.

Type Casting

```
int a=10, b=4, c;
float x, y;

c = a / b;
x = a / b;
y = (float) a / b;
```

The value of c will be 2

The value of x will be 2.0

The value of y will be 2.5

Relational Operators

- Used to compare two quantities.

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

Examples

10 > 20 is false

25 < 35.5 is true

12 > (7 + 5) is false

- When arithmetic expressions are used on either side of a relational operator, the arithmetic expressions will be evaluated first and then the results compared.

  a + b > c – d is the same as (a+b) > (c+d)

Logical Operators

- There are two logical operators in C (also called logical connectives).

  && ➔ Logical AND

  | | ➔ Logical OR

- What they do?
  - They act upon operands that are themselves logical expressions.
  - The individual logical expressions get combined into more complex conditions that are true or false.

Logical Operators

- Logical AND
  - Result is true if both the operands are true.
- Logical OR
  - Result is true if at least one of the operands are true.

X	Y	X && Y	X || Y
FALSE	FALSE	FALSE	FALSE
FALSE	TRUE	FALSE	TRUE
TRUE	FALSE	FALSE	TRUE
TRUE	TRUE	TRUE	TRUE

A Look Back at Arithmetic Operators:

The Increment and Decrement

Increment (++) and Decrement (--)

- Both of these are unary operators; they operate on a single operand.
- The increment operator causes its operand to be increased by 1.
  - Example: a++, ++count
- The decrement operator causes its operand to be decreased by 1.
  - Example: i--, --distance

Pre-increment versus post-increment

- **Operator written before the operand (++i, --i))**
  - Called pre-increment operator.
  - Operator will be altered in value ***before*** it is utilized for its intended purpose in the program.

- **Operator written after the operand (i++, i--)**
  - Called post-increment operator.
  - Operator will be altered in value ***after*** it is utilized for its intended purpose in the program.

Examples

Initial values :: a = 10; b = 20;

x = 50 + ++a;	a = 11, x = 61
x = 50 + a++;	x = 60, a = 11
x = a++ + --b;	b = 19, x = 29, a = 11
x = a++ – ++a;	??

*Called **side effects**:: while calculating some values, something else get changed.*

Input / Output

- printf
  - 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.

Input / Output

- scanf
  - Performs input from the standard input device, which is the keyboard by default.
  - It requires a format string and a list of variables into which the value received from the input device will be stored.
  - It is required to put an ampersand (&) before the names of the variables.

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