Introduction to C Programming Language

by NZP

What you will learn

- Thinking algorithmically
- Understanding the Hello World
- Variable types, const declarations
- Basic I/O
- Conditionals, loops
- Arrays, char strings

Thinking Algorithmically

- Algorithm: A procedure for solving a problem in terms of
  - the actions to be executed,
  - the order in which the actions are to be executed.
- Specifying the order in a computer program is called program control.

Thinking Algorithmically

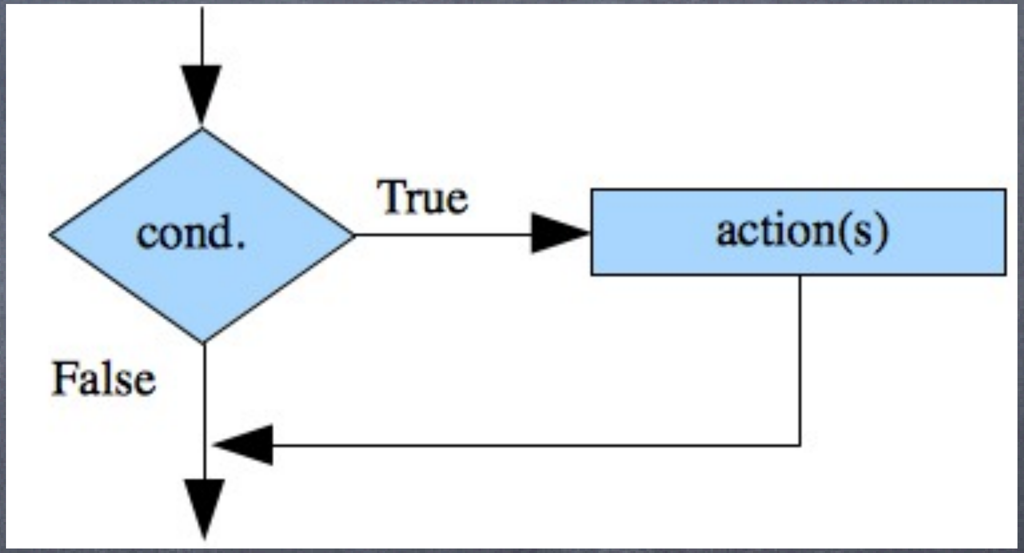
- To develop algorithms, we can use
  - pseudo-codes,
  - flowcharts

Thinking Algorithmically

- All programs can be written in terms of three control structures
  - Sequence: Statements execute one after the other
  - Selection: if, if...else, switch
  - Repetition: for, while, do...while
- We can stack them or nest them to develop algorithms
- All structures are single-entry/single-exit!

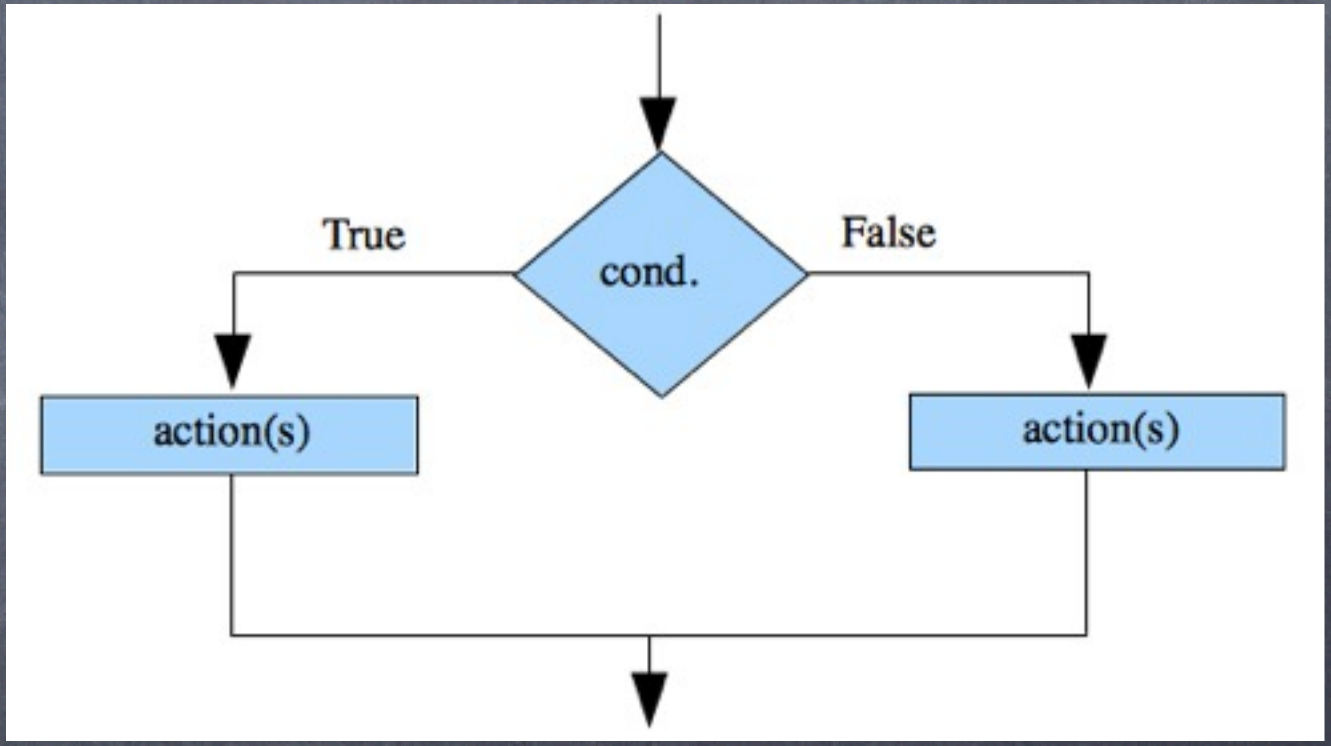
Thinking Algorithmically

- Single-selection if statement:



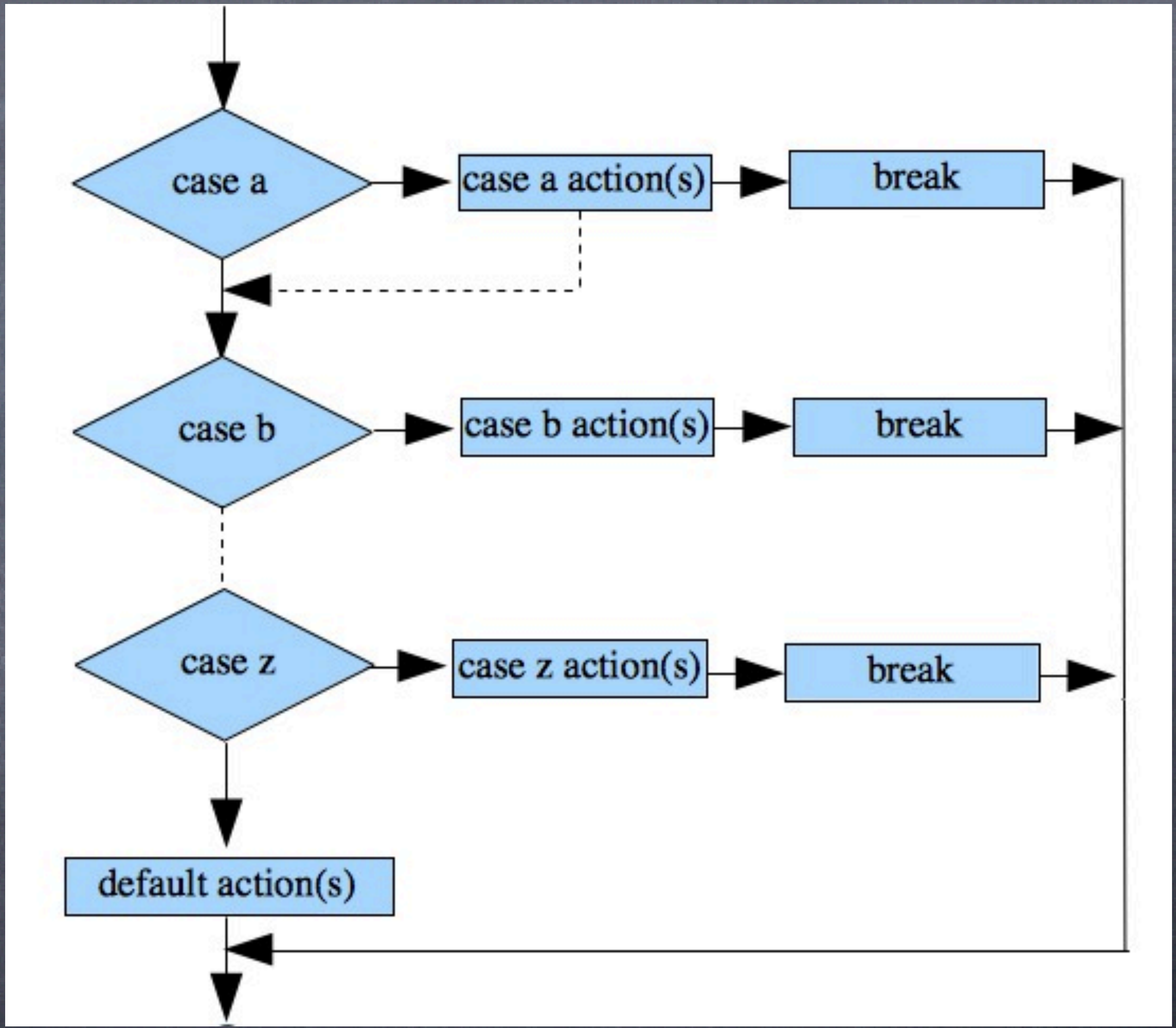
Thinking Algorithmically

- Double-selection if statement:



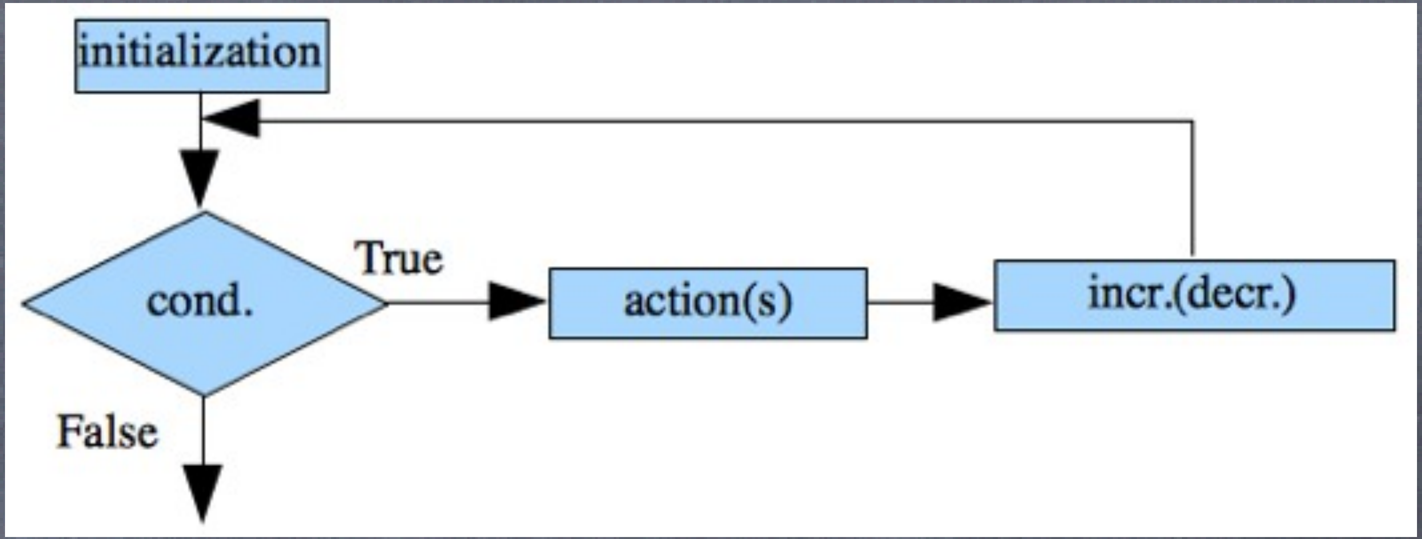
Thinking Algorithmically

- Switch multiple-selection statement:



Thinking Algorithmically

- Repetition (loop) statement:
- In a for loop increment (decrement) is always the last statement to be executed. In while or do..while, it is up to you!



Hello World!

```
/* A first program in C */
#include <stdio.h>
int main(void)
{
    printf("Hello World!\n");
    return 0;
}
```

Components of a C Program

- All C programs consist of one or more functions, each of which contains one or more statements.
- The only function that every C program must have is main()
- Library functions: The standard library contains functions to perform disk I/O, string manipulations, etc.
- Header files: Contain information about the library functions. Added by the #include preprocessor directive
- /*...*/ indicates that the line is a comment.

CASE 1: Counter-Controlled Repetition

Case: A class of ten students took a quiz.The grades (integers [0,100]) are available. Determine the class average on the quiz.

Pseudo-code

```
Set total to zero
Set grade counter to one
While grade counter is less than or equal to ten
  Input next grade
  Add the grade into the total
  Add one to the grade counter
Set the class average to the total divided by ten
Print the class average
```

Source code

```c
#include <stdio.h>

#define LOWER 1
#define UPPER 10
#define INCREMENT 1

int main(void) {
   int counter, grade, total, average;
   /* initialization phase */
   total = 0;
   counter = 1;
   /* processing phase */
   for (counter=LOWER; counter<=UPPER; counter+=INCREMENT) {
       printf("Enter grade:");
       scanf("%d",&grade);
       total += grade;
   }
   /* termination phase */
   average = total / 10;
   printf("Class average is %d\n",average);
   return 0;
}
```

Varaibles types

- All variables must be declared before they can be used!
- C supports five different basic data types:
- printf("%zu",sizeof(int)); /* gives size in bytes */

Type	Keyword	Size (byte)
character data	char	1
signed whole numbers	int	4
floating-point numbers	float	4
double-precision f-p	double	8
valueless	void	n/a

CASE 2: Sentinel-Controlled Repetition

Case: Develop a class averaging program that will process an arbitrary number of grades each time the program is run.

Pseudo-code

```
Set total to zero
Set grade counter to zero
Input first grade
While the user has not yet entered the sentinel
  Add the grade into the total
  Add one to the grade counter
  Input next grade
If the counter is not equal to zero
  Set the average to the total divided by counter
  Print the class average
Else
  Print “No grades were entered”
```

Source code

```
#include <stdio.h>

int main(void) {
   float average;
   int counter, grade, total;
    /* initialization phase */
   total = 0;
   counter = 0;
   /* processing phase */
   printf("Enter grade, -1 to end:");
   scanf("%d",&grade);
   while (grade != -1) {
       total += grade;
       counter += 1;
       printf("Enter grade, -1 to end:");
       scanf("%d",&grade);
   }
   /* termination phase */
   if (counter != 0) {
       average = (float) total / counter;
       printf("Class average is %.2f\n",average);
   }
   else {
       printf("No grades were entered\n");
   }
   return 0;
}
```

Type Cast

Since both total and counter are integers

```
total / counter
```

evaluates to an integer, thus the fractional part is lost!
Use a type cast to overcome this problem

```
(float) total / counter
```

creates a temporary copy of total, and a f-p value divided by an integer gives a floating point value.

%.2f Displays the f-p value rounded to two digits.

Arrays

- 1-D Array: A list of variables that are all the same type and are accessed through a common name
- Array element: An individual variable in an array
- Declaration: Declare a 1-D array by:

  ```
  type var_name[size];
  ```

  where size is the number of elements in the array
- You may use the value of an array element anywhere you would use a simple variable or constant

Arrays

- To print the sum of the values contained in the first three elements of i:

```
printf("%d", i[0] + i[1] + i[2]);
```

- To input a numeric value into an array element

```
scanf("%d", &i[0]);
```

- You may not assign one entire array to another!

```
char a1[10], a2[10];
a2 = a1; /* this is wrong */
```

Initializing an Array

- Value list: Use a comma separated list:

```
int n[10] = { 1,2,3,4,5,6,7,8,9,10 };
```

  - Rule: Fewer initializers than elements, the remaining elements are initialized to ZERO.
  - Rule: Arrays are not automatically initialized to zero. At least the first element must be initialized to ZERO.

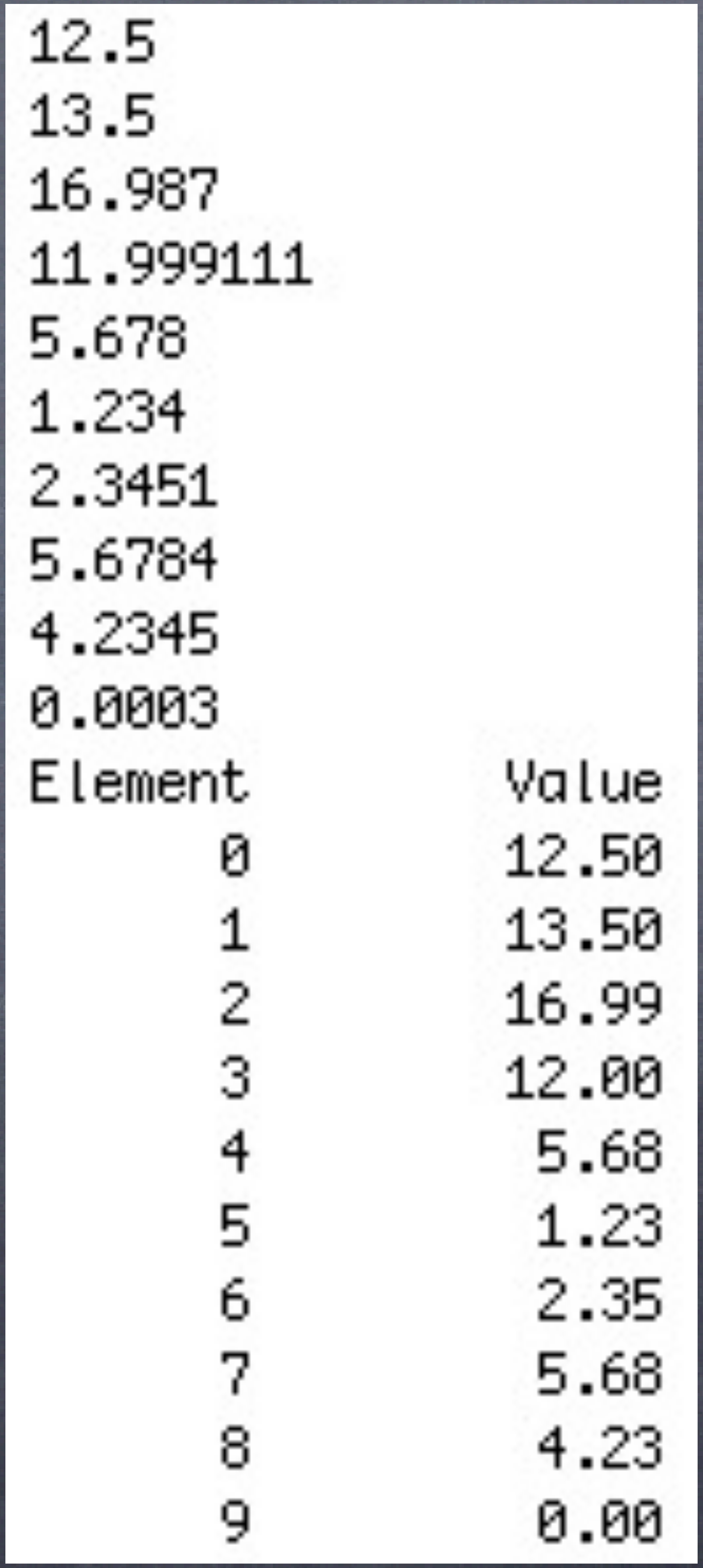
Initializing an Array

Using a repetition structure

```c
#include <stdio.h>
#define SIZE 10

int main(void) {
  float s[SIZE];
  int j;
  for (j=0; j < SIZE; ++j) {
     scanf("%f", &s[j]);
  }
  printf("%s%13s\n", "Element", "Value");
  for (j=0; j < SIZE; j++) {
     printf("%7d%13.2f\n", j, s[j]);
  }
  return 0;
}
```

SAMPLE INPUT & OUTPUT



```
12.5
13.5
16.987
11.999111
5.678
1.234
2.3451
5.6784
4.2345
0.0003
Element            Value
      0            12.50
      1            13.50
      2            16.99
      3            12.00
      4             5.68
      5             1.23
      6             2.35
      7             5.68
      8             4.23
      9             0.00
```

Strings as Char Arrays

- The most common use of the 1-D array is the string.
- string = Null-terminated character array
- A null is zero: '\0'
- Be sure to make room for the null!
- The size of the following string is 5 + 1 = 6
  ```
  char str[]="first";
  char srt[] = {'f','i','r','s','t','\0'}; /* equivalent */
  ```

Strings as Char Arrays

- scanf() can be used to read a string using the %s conversion specifier. But: scanf() stops reading when first white space character is encountered.
- white space character = space, tab, newline

```
#include <stdio.h>

int main(void) {
  char str[80];
  printf("Enter a string: ");
  scanf("%s",str);
  printf("%s\n",str);
  return 0;
}
```