Welcome to C-programming Class

Introduction to C99

What is C99?

C99 is an improved version of the C language standard. It introduced many useful features that make C programming easier and more powerful.

Important C99 Features

Declaration of variables anywhere in a block

//single-line comments

long long int

inline functions

Variable Length Arrays (VLA)

stdbool.h for Boolean values

stdint.h for fixed-size integer types

Designated initializers

Declaration of variables anywhere in a block

```
#include <stdio.h>

int main() {
    printf("Start of program\n");
    int a = 10;   // C99 allows declaration here
    printf("Value of a = %d\n", a);
    return 0;
}
```

Variable Linear Array in C99

```
#include <stdio.h>
int main() {
    int n;
    printf("Enter size: ");
    scanf("%d", &n);
    int arr[n];   // VLA in C99
for(int i = 0; i < n; i++) {
        arr[i] = i + 1;
    }
for(int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

Standard libraries in C99

stdio.h

Used for input and output functions.

Common functions: printf(), scanf(), getchar(), putchar(), fgets()

```
#include <stdio.h>
int main() {
    int a;
    printf("Enter a number: ");
    scanf("%d", &a); // takes input
    printf("You entered %d\n", a); // displays output
    return 0;
}
```

Standard libraries in C99

stdlib.h
Used for memory allocation, conversion, random numbers, and program control.
Common functions: malloc(), free(), atoi(), rand(), srand(), exit()

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int *p;
    p = (int *)malloc(sizeof(int)); // allocate memory
    *p = 25;
    printf("Value = %d\n", *p);
    free(p); // release memory
    return 0;
}
```

Standard libraries in C99

assert.h

Used for debugging.

```
#include <stdio.h>
#include <assert.h>

int main() {
    int age = 20;
    assert(age >= 18); // condition must be true
    printf("Valid age\n");
    return 0;
}
```

Standard libraries in C99

math.h

Used for mathematical functions.

Common functions: sqrt(), pow(), sin(), cos()

```
#include <stdio.h>
#include <math.h>

int main() {
    double n = 25;
    printf("Square root = %.2lf\n", sqrt(n));
    return 0;
}
```

Standard libraries in C99

time.h

Used for date and time.

```
#include <stdio.h>
#include <time.h>

int main() {
    time_t t;
    time(&t);
    printf("Current time: %s", ctime(&t));
    return 0;
}
```

Standard libraries in C99

ctype.h

Used to test and convert characters.

Common functions: isalpha(), isdigit(), toupper(), tolower()

```
#include <stdio.h>
#include <ctype.h>

int main() {
    char ch = 'a';
    printf("Uppercase = %c\n", toupper(ch));
    return 0;
}
```

Standard libraries in C99

setjmp.h :- Used for non-local jumps.

```
#include <stdio.h>
#include <setjmp.h>

jmp_buf buf;

int main() {
    if (setjmp(buf) == 0) {
        printf("First time through setjmp\n");
        longjmp(buf, 1); // jump back
    } else {
        printf("Returned using longjmp\n");
    }
    return 0;
}
```

Standard libraries in C99

string.h

Used for string handling.

Common functions: strlen(), strcpy(), strcmp(), strcat()

```
#include <stdio.h>
#include <string.h>
int main() {
    char a[20] = "Hello";
    char b[20] = " World";
    strcat(a, b); // join strings
    printf("%s\n", a);
    return 0;
}
```

Linear Search

Algorithm 4.5: (Linear Search) LINEAR(DATA, N, ITEM, LOC)

Here DATA is a linear array with N elements, and ITEM is a given item of information. This algorithm finds the location LOC of ITEM in DATA, or sets LOC := 0 if the search is unsuccessful.

1. [Insert ITEM at the end of DATA.] Set DATA[N + 1] := ITEM.

2. [Initialize counter.] Set LOC := 1.

3. [Search for ITEM.]

Repeat while DATA[LOC] ≠ ITEM:

Set LOC := LOC + 1.

[End of loop.]

4. [Successful?] If LOC = N + 1, then: Set LOC := 0.

5. Exit.

Linear Search

Algorithm 4.5: (Linear Search) LINEAR(DATA, N, ITEM, LOC)

Here DATA is a linear array with N elements, and ITEM is a given item of information. This algorithm finds the location LOC of ITEM in DATA, or sets LOC := 0 if the search is unsuccessful.

1. [Insert ITEM at the end of DATA.] Set DATA[N + 1] := ITEM.

2. [Initialize counter.] Set LOC := 1.

3. [Search for ITEM.]

Repeat while DATA[LOC] ≠ ITEM:

Set LOC := LOC + 1.

[End of loop.]

4. [Successful?] If LOC = N + 1, then: Set LOC := 0.

5. Exit.

C program on linear search

```
#include <stdio.h>
int main() {
    int arr[] = {10, 20, 30, 40, 50};
    int item = 30;
    int loc = -1;
    int i, n = 5;
    for(i = 0; i < n; i++) {
        if(arr[i] == item) {
            loc = i;
            break;
        }
    }

    if(loc != -1)
        printf("Item found at position %d", loc + 1);
    else
        printf("Item not found");

    return 0;
}
```

C program on linear search

```
#include <stdio.h>
int main() {
    int DATA[100], N, ITEM, LOC;
    int I;
    // Input total number of elements
    printf("Enter number of elements: ");
    scanf("%d", &N);
    // Input array elements
    printf("Enter %d elements:\n", N);
    for (I = 0; I < N; I++) {
        scanf("%d", &DATA[I]);
    }
    // Input item to be searched
    printf("Enter ITEM to search: ");
    scanf("%d", &ITEM);
    // Step 1: [Insert ITEM at end of DATA]
    // In C, array index starts from 0, so last valid extra position is DATA[N]
    DATA[N] = ITEM;
```

C program on linear search

```
    // Step 2: [Initialize counter]
    LOC = 0;
    // Step 3: [Search for ITEM]
    // Repeat while DATA[LOC] != ITEM
    while (DATA[LOC] != ITEM) {
        LOC = LOC + 1;
    }
    // Step 4: [Successful?]
    // If LOC == N, then ITEM was found only at inserted extra position
    // That means ITEM is not present in original array
    if (LOC == N) {
        LOC = -1;
    }
    // Step 5: Output result
    if (LOC == -1) {
        printf("ITEM is not present in DATA\n");
    } else {
        printf("ITEM found at location %d\n", LOC + 1); // +1 for position
    }
    return 0;
}
```

C program on linear search

```
    // Step 2: [Initialize counter]
    LOC = 0;
    // Step 3: [Search for ITEM]
    // Repeat while DATA[LOC] != ITEM
    while (DATA[LOC] != ITEM) {
        LOC = LOC + 1;
    }
    // Step 4: [Successful?]
    // If LOC == N, then ITEM was found only at inserted extra position
    // That means ITEM is not present in original array
    if (LOC == N) {
        LOC = -1;
    }
    // Step 5: Output result
    if (LOC == -1) {
        printf("ITEM is not present in DATA\n");
    } else {
        printf("ITEM found at location %d\n", LOC + 1); // +1 for position
    }
    return 0;
}
```

Binary search

Let DATA be the following sorted 13-element array:

DATA: 11, 22, 30, 33, 40, 44, 55, 60, 66, 77, 80, 88, 99

We apply the binary search to DATA for different values of ITEM.

(a) Suppose ITEM = 40. The search for ITEM in the array DATA is pictured in Fig. 4.6, where the values of DATA[BEG] and DATA[END] in each stage of the algorithm are indicated by circles and the value of DATA[MID] by a square. Specifically, BEG, END and MID will have the following successive values:

1. Initially, BEG = 1 and END = 13. Hence

MID = INT[(1 + 13)/2] = 7 and so DATA[MID] = 55

2. Since 40 < 55, END has its value changed by END = MID – 1 = 6. Hence

MID = INT[(1 + 6)/2] = 3 and so DATA[MID] = 30

3. Since 40 > 30, BEG has its value changed by BEG = MID + 1 = 4. Hence

MID = INT[(4 + 6)/2] = 5 and so DATA[MID] = 40

We have found ITEM in location LOC = MID = 5.

Binary Search:-

Algorithm 4.6: (Binary Search) BINARY(DATA, LB, UB, ITEM, LOC)

Here DATA is a sorted array with lower bound LB and upper bound UB, and ITEM is a given item of information. The variables BEG, END and MID denote, respectively, the beginning, end and middle locations of a segment of elements of DATA. This algorithm finds the location LOC of ITEM in DATA or sets LOC = NULL.

1. [Initialize segment variables.]
Set BEG := LB, END := UB and MID = INT((BEG + END)/2).
2. Repeat Steps 3 and 4 while BEG ≤ END and DATA[MID] ≠ ITEM.
3. If ITEM < DATA[MID], then:
Set END := MID – 1.
Else:
Set BEG := MID + 1.
[End of If structure.]
4. Set MID := INT((BEG + END)/2).
[End of Step 2 loop.]
5. If DATA[MID] = ITEM, then:
Set LOC := MID.
Else:
Set LOC := NULL.
[End of If structure.]
6. Exit.

Binary search

```
#include <stdio.h>
int main() {
    int DATA[100];          // array to store elements
    int LB, UB;             // lower bound and upper bound
    int ITEM;               // item to be searched
    int LOC;                // location of item
    int BEG, END, MID;      // segment variables
    int N, i;               // N = number of elements, i = loop variable

   // Input number of elements
    printf("Enter number of elements: ");
    scanf("%d", &N);

    // Input sorted array elements
    printf("Enter %d elements in sorted order:\n", N);
    for(i = 0; i < N; i++) {
        scanf("%d", &DATA[i]);
    }
```

Binary search

```
// Input item to search
    printf("Enter ITEM to search: ");
    scanf("%d", &ITEM);

    // Set lower bound and upper bound
    LB = 0;          // first index of array
    UB = N - 1;      // last index of array

    // Step 1: Initialize segment variables
    BEG = LB;                  // beginning of search segment
    END = UB;                  // end of search segment
    MID = (BEG + END) / 2;       // middle position

    // Step 2: Repeat Steps 3 and 4 while BEG <= END and DATA[MID] != ITEM
    while(BEG <= END && DATA[MID] != ITEM) {

        // Step 3: If ITEM is less than middle element, search left half
        if(ITEM < DATA[MID]) {
            END = MID - 1;       // move END to left side
        }
        else {
            BEG = MID + 1;       // otherwise search right side
        }
```

Binary search

```
// Step 4: Find new MID
        MID = (BEG + END) / 2;
    }
    // Step 5: Check whether ITEM is found
    if(BEG <= END && DATA[MID] == ITEM) {
        LOC = MID;              // item found at MID
    }
    else {
        LOC = -1;               // NULL in algorithm, here -1 means not found
    }
    // Step 6: Display result
    if(LOC != -1) {
        printf("ITEM found at location %d\n", LOC + 1); // +1 for position
    }
    else {
        printf("ITEM not found\n");
    }
    return 0;
}
```

Bubble Sort

Bubble sort compares adjacent elements and swaps them if they are in the wrong order.

Suppose the following numbers are stored in an array A:

32, 51, 27, 85, 66, 23, 13, 57

(a) Compare A_1 and A_2. Since 32 < 51, the list is not altered.

(b) Compare A_2 and A_3. Since 51 > 27, interchange 51 and 27 as follows:

32, 27, 51, 85, 66, 23, 13, 57

(c) Compare A_3 and A_4. Since 51 < 85, the list is not altered.

(d) Compare A_4 and A_5. Since 85 > 66, interchange 85 and 86 as follows:

32, 27, 51, 66, 85, 23, 13, 57

Bubble Sort

Bubble sort compares adjacent elements and swaps them if they are in the wrong order.

Suppose the following numbers are stored in an array A:

32, 51, 27, 85, 66, 23, 13, 57

Bubble Sort

Bubble sort compares adjacent elements and swaps them if they are in the wrong order.

Suppose the following numbers are stored in an array A:

32, 51, 27, 85, 66, 23, 13, 57

(e) Compare A_5 and A_6. Since 85 > 23, interchange 85 and 23 as follows:

32, 27, 51, 66, 23, 85, 13, 57

(f) Compare A_6 and A_7. Since 85 > 13, interchange 85 and 13 to yield:

32, 27, 51, 66, 23, 13, 85, 57

(g) Compare A_7 and A_8. Since 85 > 57, interchange 85 and 51 to yield:

32, 27, 51, 66, 23, 13, 57, 85

Bubble Sort

Suppose the following numbers are stored in an array A:

32, 51, 27, 85, 66, 23, 13, 57

At the end of this first pass, the largest number, 85, has moved to the last position. However, the rest of the numbers are not sorted, even though some of them have changed their positions.

For the remainder of the passes, we show only the interchanges.

Pass 2. (27, 33,) 51, 66, 23, 13, 57, 85

27, 33, 51, (23, 66,) 13, 57, 85

27, 33, 51, 23, (13, 66,) 57, 85

27, 33, 51, 23, 13, (57, 66,) 85

At the end of Pass 2, the second largest number, 66, has moved its way down to the next-to-last position.

Bubble Sort

Suppose the following numbers are stored in an array A:

32, 51, 27, 85, 66, 23, 13, 57

Bubble Sort

Suppose the following numbers are stored in an array A:

32, 51, 27, 85, 66, 23, 13, 57

At the end of this first pass, the largest number, 85, has moved to the last position. However, the rest of the numbers are not sorted, even though some of them have changed their positions.

For the remainder of the passes, we show only the interchanges.

Pass 2. (27, 33,) 51, 66, 23, 13, 57, 85

27, 33, 51, (23, 66,) 13, 57, 85

27, 33, 51, 23, (13, 66,) 57, 85

27, 33, 51, 23, 13, (57, 66,) 85

At the end of Pass 2, the second largest number, 66, has moved its way down to the next-to-last position.

Bubble sort

Algorithm 4.4: (Bubble Sort) BUBBLE(DATA, N)

Here DATA is an array with N elements. This algorithm sorts the elements in DATA.

1. Repeat Steps 2 and 3 for K = 1 to N – 1.

2. Set PTR := 1. [Initializes pass pointer PTR.]

3. Repeat while PTR ≤ N – K: [Executes pass.]

(a) If DATA[PTR] < DATA[PTR + 1], then:

Interchange DATA[PTR] and DATA[PTR + 1].

[End of If structure.]

(b) Set PTR := PTR + 1.

[End of inner loop.]

[End of Step 1 outer loop.]

4. Exit.

Bubble sort Program in C

```
#include <stdio.h>
int main() {
    int arr[5] = {5, 1, 4, 2, 8};
    int i, j, temp;

    for(i = 0; i < 4; i++) {
        for(j = 0; j < 4 - i; j++) {
            if(arr[j] > arr[j + 1]) {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
    printf("Sorted array: ");
    for(i = 0; i < 5; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

Bubble sort Program in C

God Bless you all

Thanks …