Programming in C/C++

Daniel Chatfield

November 26, 2014

1. What is the difference between $'a'$ and $"a"$?

   Answer:

   $'a'$ denotes a single character literal where as $"a"$ denotes a string literal (char array) containing 2 character literals (an a and a null terminator).

2. Will the following terminate, and if so under what circumstances?

   ```
   char i,j; for (i=0; i<10,j<5; i++,j++);
   ```

   Answer:

   The variable j is not assigned to and thus the value it takes is implementation specific and can be simply garbage. The first part of the second expression ($i < 10$) is unused and the for loop condition is simply $j < 5$. The program will terminate whenever $j < 5$ becomes $false$ which will always happen. The number of loops that will occur before the for loop exits is determined by the starting value of j which is indeterminate.

3. (a) Write an implementation of bubble sort for a fixed array of integers.
   An array of integers can be defined as int i[] = 1,2,3,4;
   The 2nd integer in an array can be printed using $printf("\%dnn", i[1])$;.

   Answer:

   ```
   #include <stdio.h>

   int main(void) {

       int items[] = {4, 6, 5, 1, 8, 2, 8, 6};

       int n = sizeof(items)/sizeof(items[0]);

       for (int j = n; j > 1; j--) {
           for (int i = 0; i < j-1; i++) {
   ```

```
            if (items[i] > items[i+1]) {
                int rv = items[i];
                items[i] = items[i+1];
                items[i+1] = rv;
            }
        }
    }

    for (int i = 0; i < n; i++) {
        if(i != 0) {
            printf(",");
        }
        printf("%d", items[i]);
    }

    return 0;
}
```

(b) Modify your answer to the previous question to sort characters into lexicographical order.
The 2nd character in a character array i can be printed using printf("%cnn", i[1]);.

Answer:

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

int main(void) {

    char str[] = "Computer Laboratory";


    for (int j = strlen(str); j > 1; j--) {
        for (int i = 0; i < j-1; i++) {
            if (strcmp(&str[i], &str[i+1]) > 0) {
                int rv = str[i];
                str[i] = str[i+1];
                str[i+1] = rv;
            }
        }
    }

    // Test it
```

```
    for (int i = 0; i < strlen(str); i++) {
        printf("%c", str[i]);
    }

    return 0;
}
```

4. Write a function definition which matches the following declaration. The implementation should return the number of lower-case letters in a string

```
int cntlower(char str[]);
```

Answer:

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

int cntlower(char str[]) {
    int counter = 0;

    for (int i = 0;  i < strlen(str); i++) {
        if(str[i] >= 'a' && str[i] <= 'z')
            counter++;
    }

    return counter;
}

int main(void) {

    char str[] = "Computer Laboratory";

    int counter = cntlower(str);

    printf("%d", counter);

    return 0;
}
```

5. Use function recursion to write an implementation of merge sort for a fixed array of integers; how much memory does your program use for a list of length n?

Answer:

```
#include <stdio.h>

void _mergesort(int arr[], int start, int length, int tmp[]) {
    if(length < 2) {
        return;
    }


    // split into 2 parts
    int half_length = length / 2;
    int middle = start + half_length;


    // sort parts
    _mergesort(arr, start, half_length, tmp);
    _mergesort(arr, middle, length - half_length, tmp);

    //merge the lists
    int i = 0;
    int j = 0;
    int k = 0;

    while (i < half_length && j < length - half_length) {
        int left = arr[start+i];
        int right = arr[middle+j];

        if (left < right) {
            tmp[k] = left;
            i++;
        } else {
            tmp[k] = right;
            j++;
        }
        k++;
    }

    // finish it off
    while (i < half_length) {
        tmp[k] = arr[start+i];
        i++;
        k++;
    }
    while (j < length - half_length) {
```

```
        tmp[k] = arr[middle+j];
        j++;
        k++;
    }

    // copy back into original array
    for(k = 0; k < length; k++) {
        arr[start+k] = tmp[k];
    }
}

void mergesort(int arr[], int length) {
    //not sure why compiler complains without the ampersand
    int tmp[length];
    _mergesort(arr, 0, length, tmp);
}

int main(void) {
    int items[] = {4, 6, 5, 1, 8, 2, 8, 6};
    int n = sizeof(items)/sizeof(int);
    mergesort(items, n);


    for (int i = 0; i < n; i++) {
        if(i != 0) {
            printf(",");
        }
        printf("%d", items[i]);
    }
    return 0;
}
```

It uses $O(n)$ space as it creates a new array of the same size.

6. (a) Define a macro $SWAP(t, x, y)$ that exchanges two arguments of type t.

Answer:

```
#include <stdio.h>

#define SWAP(T,x,y) {T rv = x; x = y; y = rv; }

int main(void) {
    int x = 2;
```

```
    int y = 4;

    SWAP(int,x,y);
    printf("X:%d,Y:%d\n",x,y);
}
```

(b) Does your macro work as expected for SWAP(int, v[i++], w[f(x)])?

Answer:

It doesn't work properly, i++ has a side effect (incrementing i) and thus is not safe to use any more than once in the macro result. The same could be true for f(x).

(c) What other invocations could cause a simple implementation to fail and how can we mitigate them?

Answer:

Not sure about other invocations (other than just others that have side effects). I think by using pointers to everything you should be able to mitigate this but cannot actually get this to work.

7. Define a macro $SWAP(x, y)$ that exchanges two arguments of the same type (e.g. int or char) without using a temporary

Answer:

```
#include <stdio.h>

#define SWAP(x,y) { x > y ? \
    (x = x - y, y = y + x, x = y - x): \
    (y = y - x, x = x + y, y = x - y);}

int main(void) {

    int x = 2;
    int y = 4;

    SWAP(x,y);
    printf("X:%d,Y:%d\n",x,y);
}
```

8. If p is a pointer, what does p[−2] mean? When is this legal?

Answer:

It references the array item that is two before the element pointed to by the p pointer. It is always legal if p is a pointer but if it is not pointing to an element in an array or if the index offset references an element that is out of bounds then you will get garbage.

9. Write a string search function with the following declaration which returns a pointer to the first occurrence of s in f (and NULL otherwise)

```
char *strfind(const char *s, const char *f);
```

Answer:

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

const char* strfind(const char* s, const char* f) {
    while (*f != '\0') {
        if (*s == *f) {
            return f;
        }
        f++;
    }

    return NULL;
}

int main(void) {
    char haystack[] = "Computer Laboratory";
    char needle = 'L';

    const char *b = strfind(&needle, haystack);

    if (b == NULL) {
        printf("NULL");
    } else {
        printf("%c", *b);
    }
}
```

10. If p is a pointer to a structure, write some C code which uses all the following indirections and describe the action of each code snippet

++p->i
p++->i
*p->i
*p->i++
(*p->i)++
*p++->i

Answer:

```
#include <stdio.h>

typedef struct s {
    int i;
    int *j;
} s;

void print(s arr[], s* p, int x) {
    printf("{{%d,%d},{%d,%d},{%d,%d}} p:%d rv:%d\n", arr[0].i,*(arr[
}

int main(void) {

    int x = 4, y = 5, z = 6;

    s arr[] = {{1,&x},{2, &y},{3, &z}};
    s *p = arr;

    int rv = 0;
    print(arr,p,rv);

    rv = ++p->i;
    print(arr,p,rv);

    rv = p++->i;
    print(arr,p,rv);

    rv = *p->j;
    print(arr,p,rv);

    rv = *p->j++;
    print(arr,p,rv);

    rv = (*p->j)++;
    print(arr,p,rv);
```

```
    rv = *p++->j;
    print(arr,p,rv);

    return 0;
}
```

11. Write a program calc which evaluates a reverse Polish expression given on the command line; for example $ calc 2 3 4 + * should print 14

Answer:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char getOperation(const char *str);

typedef struct Stack {
    int arr[100];
    int pos;
} Stack;


void push(Stack*, int);
int pop(Stack*);

int main(int argc, char *argv[]) {
    Stack *stack = malloc(sizeof(Stack));
    memset(stack,0,sizeof(Stack));

    for(int x = 1; x < argc; x++) {
        char* end;
        int val = strtol(argv[x], &end, 10);

        if (!*end) {
            push(stack, val);
        } else {
            char op = getOperation(argv[x]);
            int a;
            int b;
            switch(op) {
                case '+':
                    a = pop(stack);
                    b = pop(stack);
```

```
                    push(stack, a + b);
                    break;
                case '-':
                    a = pop(stack);
                    b = pop(stack);
                    push(stack, b - a);
                    break;
                case '/':
                    a = pop(stack);
                    b = pop(stack);
                    push(stack, b / a);
                    break;
                case '*':
                    a = pop(stack);
                    b = pop(stack);
                    push(stack, b * a);
                    break;
                default:
                    fprintf(stderr,"Invalid input\n");
                    exit(1);

            }
        }

    }

    int x = pop(stack);

    printf("%d", x);

    free(stack);

    return 0;
}

int isDigit(char c) {
    return (c >= '0' && c <= '9');
}

int isNumber(const char *arr) {
    int i;
    for (i = 0; isDigit(arr[i]); i++) {}

    return (i == 0 || arr[i] != '\0');
}
```

```
char getOperation(const char *str) {
    if (strlen(str) != 1) {
        return 'F';
    }

    return str[0];
}

void push(Stack* stack, int val) {
    stack->arr[stack->pos] = val;
    (stack->pos)++;
}

int pop(Stack* stack) {
    if(stack->pos == 0) {
        fprintf(stderr,"Invalid input\n");
        exit(1);
    }
    return stack->arr[--(stack->pos)];
}
```

12. What is the value of i after executing each of the following:

    (a) i = sizeof(char);

    Answer:

    1

    (b) i = sizeof(int);

    Answer:

    4

    (c) int a; i = sizeof a;

    Answer:

    4

    (d) char b[5]; i = sizeof(b);

Answer:

5

(e) char *c=b; i = sizeof(c);

Answer:

8

(f) struct {int d; char e;} s; i = sizeof s;

Answer:

8

(g) void f(int j[5]) { i = sizeof j;}

Answer:

Couldn't get this to work

(h) void f(int j[][10]) i = sizeof j;

Answer:

Couldn't get this to work

13. Popular programming journal Obscure C Techniques for Experts has published a novel way to save space for a doubly-linked list program. Instead of storing two pointers (one next and one previous), this new technique stores a single value: the XOR of previous and next pointers.

(a) (15 points) You have been engaged to provide code examples of this approach for publication.

Ensure your code illustrates the creation and initialisation of such a list as well as the insertion, and deletion, of elements from such a list. Additionally, you must provide examples of a forward or backward traversal of the list permitting examination of each element in turn.

Answer:

Not sure on the whole concept here.

(b) (5 points) Comment on this form of linked list. Consider the comparative

speed, memory overheads, maintenance and other advantages or disadvantages of the XOR doubly-linked list approach when compared with an approach that stores both previous and next pointers.

Answer:

The memory overhead is less as it only needs to store one pointer. A disadvantage is that debugging tools probably don't know how to traverse the list and this might hinder them. Also if pointer to an element is stored in another data structure then you can't use this to traverse the entire list (not actually sure about this but I can't see how you would be able to traverse the list from just the XOR without making wild assumptions about the location of the elements in memory).

14. (a) According to the rules of pre and post increment what would you expect the following expressions to do:

```
i++ + ++i
++i + ++i
```

Answer:

The first one should return $2i + 1$ and i should have the value $i + 2$ afterwards.
The second one should return $2i+3$ and the value of i afterwards should be $i + 2$.

(b) What actually happens if you try it?

Answer:

```
#include <stdio.h>

#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Wunsequenced"
int main(void) {
    int i = 1;
    int x =  i++ + ++i;
    printf("%d, %i\n", x, i); // 4,3
    i = 1;
    x = ++i + ++i;
    printf("%d, %i\n", x, i); // 5,3

}
#pragma clang diagnostic pop
```