Tutorial Sheet (October 12, 2018)

ESC101 – Fundamentals of Computing

Announcement

1. **AT Drop**: Today is the last day for dropping advanced track. Drop requests must be mailed to instructor, group members, mentors
2. **Course Drop**: Today is the last day for dropping the course. Drop request must be made on proper (online) format.

The Five Golden Rules of Pointers

1. **RULE 1**: All pointers store addresses using 8 bytes. It doesn't matter whether it is a pointer to variable, to array, to another pointer etc
2. **RULE 2** (Reference): &a gives address of variable a. It doesn't matter whether variable a is char, long, or even a pointer.
3. **RULE 3**: (Dereference): Whenever an expression expr generates an address, *(expr) gives value stored at that address. The value is interpreted using the type of expr
4. **RULE 4**: (Arithmetic): Pointer arithmetic always done with respect to the size of the datatype. char* arithmetic is w.r.t. 1 byte blocks, int* w.r.t. 4 byte blocks, double* uses 8 bytes blocks.
5. **RULE 5** (Arrays): Name of array points to first element of array. It doesn't matter whether it is a malloc-ed array or static array.

Two Kinds of Arrays

1. **Static arrays**: can be of constant length (e.g. int a[10];) or variable length (e.g. int n; scanf("%d", &n); int a[n];)
2. **Dynamic arrays**: can be of constant length (e.g. int *a = (int*)malloc(10 * sizeof(int));) or variable length (e.g. int n; scanf("%d",&n); int *a = (int*)malloc(n * sizeof(int));)

The name of an array always points to first element of array i.e. the expression a will generate the address of a[0] i.e. we have a == &a[0]

However, whereas for dynamic arrays, a is a pointer variable we freely control (we can say free(a); or even a++;) we do not have control over a if it is a static array (we cannot say free(a); and cannot say a++;). In fact, whereas &a gives us address of the pointer variable a for dynamic arrays, for static arrays &a will just return back &a[0] once more.

Functions in C

Allow us to think clearly when solving problems, as well as write code that is neater, easier to read by others, and easier to debug.

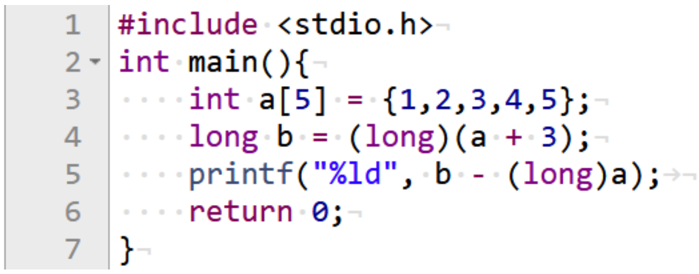
1. When we use a function, we say we *called the function*.
2. When we give an input to a function, we say that we *passed an argument to the function*.
3. When the function generates an output, we say that it *returned an output*.
4. <u>Function Name</u>: must be a valid identifier. Functions must be declared before using them.
5. <u>Function Arguments</u>: can be simple variables (int, char, double) or even pointers or arrays. Functions can have zero (void) or multiple arguments and the type of every one must be declared.
6. <u>Function Return</u>: functions can return zero (void) or one value. That value can be an int, long, float, double, char or a pointer.
7. Values returned by a function can be freely used just as variables of that datatype could have been used.

The Four Golden Rules of Functions

1. **RULE 1** (Input as variable): when a variable is passed as input to a function, the value stored inside that variable gets passed as an argument. For pointer arguments, the address stored inside the pointer variable is passed.

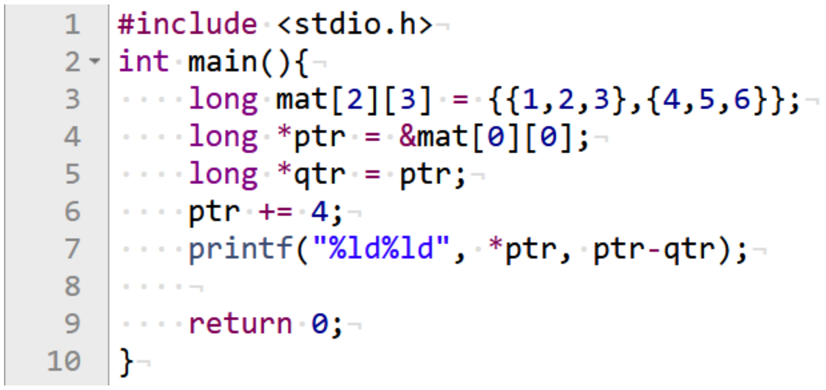
2. **RULE 2** (Input as expression): when an expression is passed as input, the value generated by that expression gets passed as argument. If value generated is an address (e.g. &a or ptr + 2), that address gets passed.
3. **RULE 3** (Type mismatch): if there is a type mismatch between what is passed and what is expected, typecasting attempted.
4. **RULE 4** (Argument replication): all values passed to functions are copied onto fresh variables used by that function alone. If passing a pointer, say ptr, a new pointer variable created and address stored inside ptr is copied onto new pointer variable.

Some Practice Questions (find the output)



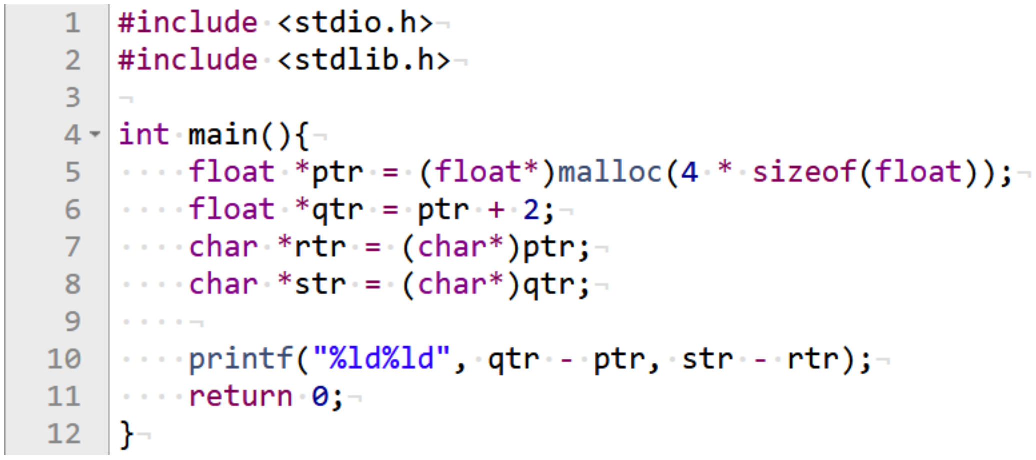
```c
#include <stdio.h>
int main(){
    int a[5] = {1,2,3,4,5};
    long b = (long)(a + 3);
    printf("%ld", b - (long)a);
    return 0;
}
```

Ans: 12



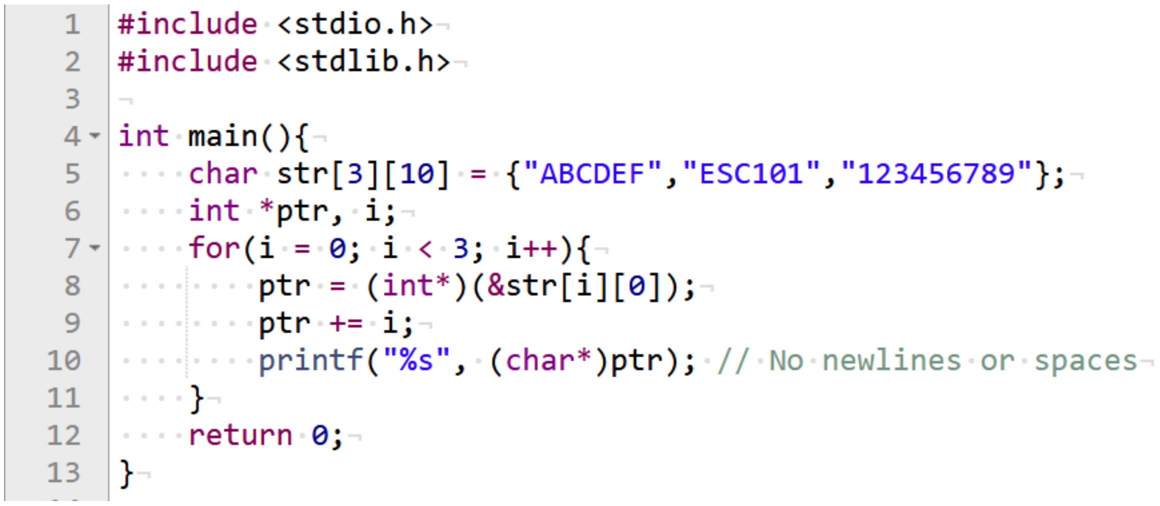
```c
#include <stdio.h>
int main(){
    long mat[2][3] = {{1,2,3},{4,5,6}};
    long *ptr = &mat[0][0];
    long *qtr = ptr;
    ptr += 4;
    printf("%ld%ld", *ptr, ptr-qtr);
    
    return 0;
}
```

Ans: 54



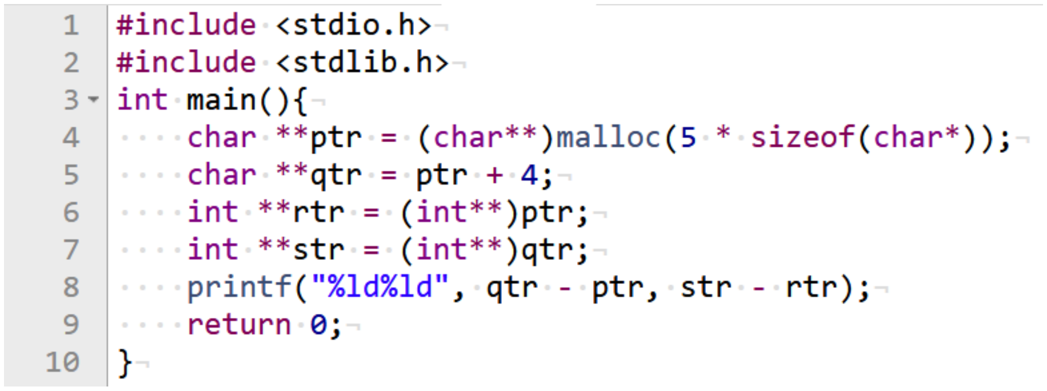
```c
#include <stdio.h>
#include <stdlib.h>

int main(){
    float *ptr = (float*)malloc(4 * sizeof(float));
    float *qtr = ptr + 2;
    char *rtr = (char*)ptr;
    char *str = (char*)qtr;
    
    printf("%ld%ld", qtr - ptr, str - rtr);
    return 0;
}
```

Ans: 28



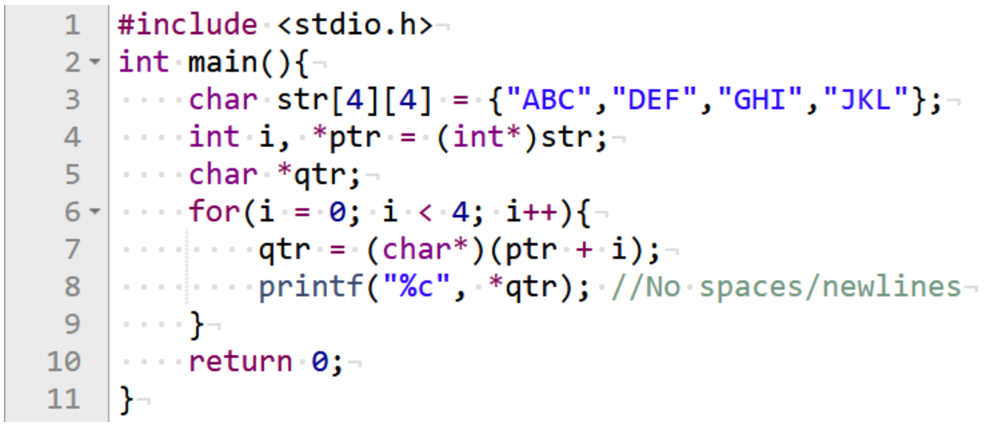
```c
#include <stdio.h>
#include <stdlib.h>

int main(){
    char str[3][10] = {"ABCDEF","ESC101","123456789"};
    int *ptr, i;
    for(i = 0; i < 3; i++){
        ptr = (int*)(&str[i][0]);
        ptr += i;
        printf("%s", (char*)ptr); // No newlines or spaces
    }
    return 0;
}
```

Ans: ABCDEF019



```c
#include <stdio.h>
#include <stdlib.h>
int main(){
    char **ptr = (char**)malloc(5 * sizeof(char*));
    char **qtr = ptr + 4;
    int **rtr = (int**)ptr;
    int **str = (int**)qtr;
    printf("%ld%ld", qtr - ptr, str - rtr);
    return 0;
}
```

Ans: 44



```c
#include <stdio.h>
int main(){
    char str[4][4] = {"ABC","DEF","GHI","JKL"};
    int i, *ptr = (int*)str;
    char *qtr;
    for(i = 0; i < 4; i++){
        qtr = (char*)(ptr + i);
        printf("%c", *qtr); //No spaces/newlines
    }
    return 0;
}
```

Ans: ADGJ

Some Pitfalls and recognizing compiler error messages

```c
int *a, *b;
a = (int*)malloc(2 * sizeof(int));
b = a;
free(b); // OK - will free array a
b = (int*)malloc(3 * sizeof(int));
b++;
free(b); // Runtime error!
```

1. When using free(), make sure to give it the exact address which malloc/calloc/realloc gave when memory was allocated.
2. Give functions same number of arguments as promised. If type mismatch in arguments, type-casting will take place – may cause errors, loss of info.
3. Execution of a function stops immediately after any return statement encountered. We can have multiple return statements in a function. However, all of them must return the same datatype as the return type of the function (for void return type, and empty return statement i.e. return; should be used.
4. When defining functions, you can name your input variables anything you like (even if you have already used them inside other functions like main()). Do not expect variables inside two different functions to share values just because their names are the same (unless it is a global variable).