**Dynamic Arrays**

> Write a function **duplicate** that uses dynamic storage allocation to create a copy of a string so that, for instance, the call

```
p = duplicate(str);
```

would allocate space for a string of the same length as **str**, copy the contents of **str** into the new string, and return a pointer to it (which is assigned to p). Have **duplicate** return a NULL pointer if memory allocation fails.

```
char *duplicate(char *s)
{
    char * t = malloc(strlen(s) + 1);
    strcpy(t,s);
    return t;
}
```

> Complete the code below for a function that creates and returns a dynamic array containing the contents of string repeated n times. The dynamic array should be just large enough to contain the resulting string. The function returns a pointer to the dynamic array. Your function should use only string functions and one local variable of type int. No local variables are allowed.

If s contains "Hi" and n has value 3, then the returned string will contain "HiHiHi".

```
char * multiply_string(char *s, int n)

{
   char * t = NULL; // new string
   int i = 0;

   if (n == 0)
     return NULL;

   t = malloc(n*strlen(s) + 1);
   strcpy(t,s);
   for(i = 0; i < n-1; ++i)
     strcat(t,s);
   return t;
}
```

> Complete the code below for a function that creates and returns a dynamic array containing the contents of string s1 followed by the contents of string2. The dynamic array should be just large enough to contain the resulting string. The function returns a pointer to the dynamic array. Your function should use only string functions. No local variables other than t are allowed.

If `s1` contained the string "`Hello`" and `s2` the string "`Goodbye`", then the string that is returned by the function would contain the string "`HelloGoodbye`".

```
char * join_strings(char *s1, char *s2)
{
   char * t; // new string
   t = malloc(strlen(s) + strlen(t) + 1);
   strcpy(t,s1);
   strcat(t,s2);
   return t;
}
```

**Structs**

> (a) Enter a typedef for the type Color, which is a struct with three integer fields named red, green and blue, respectively.

```
typedef struct {
    int red;
    int green;
    int blue;
} Color;
```

(b) Write a single statement that declares a variable of type Color named magenta having value 255 for red, 0 for green and 255 for blue.

```
Color magenta = {255,0,255};
```

(c) Complete the code for the following function modifies a Color variable by dividing all its fields by 2.

```
void color_cropper(Color *c)
{
    c->red /= 2;
    c->green /= 2;
    c->blue /= 2;
}
```

> (a) Write a `typedef` for a `struct` with three integer fields: `red`, `green` and `blue` of type `int`. The new type name should be Color.

**See above**.

(b) Write a single statement that declares and initializes a length-two array of `Color` objects, with the first having `red`, `blue`, `green` values `200`,`158`,`190` and the second having values `150`, `200`, `240`.

```
Color A[] = {{200,158,190}, {150,200,240}};
```

(c) Write a function `minColor` with three parameters of type `Color`; the first two are input parameters and the third is an output parameter. The function should use the third parameter to return a `Color` object whose field values are the minimum of the corresponding field values of the first two parameters.

```
void minColor(Color A, Color B, Color *C)
{
    if (A.red >= B.red)
      C->red = A.red;
    else
      C->red = B.red;
    if (A.green >= B.green)
      C->green = A.green;
    else
      C->green = B.green;
    if (A.blue >= B.blue)
      C->blue = A.blue;
    else
      C->blue = B.blue;
}
```

> (a) Write a `typedef` for a `struct` type `vector3` with fields `x_coord`, `y_coord`, `z_coord`, all of type `double`.

```
typedef struct {
   double x_coord;
   double y_coord;
   double z_coord;
} vector3;
```

(b) Write a single statement that declares and initializes an array of two vector3 objects, the first representing the vector (2.1,3.0,-1.5) and the second representing the vector (1.5, 2.1,5.0).

```
vector3 V[] = {{2.1,3.0,-1.5},{1.5,2.1,5.0}};
```

(c) Write a function with return type void and three vector3 parameters; the first two are input parameters and the third is an output parameter use to return the vector sum of the first two.

```
void VectorSum(vector3 u, vector3 v, vector3 *w)
{
   w->x_coordinate = u.x_coordinate + v.x_coordinate;
   w->y_coordinate = u.y_coordinate + v.y_coordinate;
   w->z_coordinate = u.z_coordinate + v.z_coordinate;
}
```

## Files

**>** You are to write a function that transfers the contents of one file to another, replacing each tab character with four space characters. The parameters are strings containing the names of the two files. If any file fails to open, the function should abort the program.

```
void untabify(char *target, char *source)
// Copies the contents of the file with name in the string source
// to the file with name in the string target, replacing all tab
// characters the source file by four spaces
{
    char ch = 0;
    FILE * in = fopen(source,"r");
    assert(in != NULL);
    FILE * out = fopen(target,"w");
    assert(out != NULL);

    ch = fgetc(in);
    while(!feof(in))
    {
       if (ch != '\t') {
           fputc(ch,out);
      else {
           fprintf(out,"    ");
      }
       ch = fgetc(in);
    }
    fclose(in);
    fclose(out);
}
```

> Complete the code for the function below that inputs values from a file into an array until the array is full or there are no more integers available in the file. The parameters are the array `A`, the length `alen` of the array and a string `fname` containing the name of the file. You should use the `feof` function to detect the end of input from the file. The function returns the number of values that were input from the file.

```
int FileToArray(int A[],int alen,char * fname) /* A has length alen */
{
  FILE * infile = fopen(fname,"r");
  assert(infile != NULL);
  int next_num = 0;
  int count = 0;

  fscanf("%d",&next_num);
  while(!feof(infile) && i < alen) {
      A[count] = next_num;
      ++count;
      fscanf("%d",&next_num);
  }

  return count;
}
```

**Linked Lists**

> Complete the code for a function that replaces each occurrence of value x by value y in a linked list with first node pointer p. Assume the info type is int.

```
void replaceLinkedList(Node* p, int x, int y)
{
   Node *tmp = p;
   while(tmp != NULL) {
      if (tmp->info == x){
         tmp->info = y;
      }
      tmp = tmp->next;
}
```

> For this problem you are to write two functions the count the number of nodes in a linked list with first node pointer p. The first function must be recursive, the second non-recursive.

(a)

```
int recursive_node_count(Node * p)
{
   if (p == NULL) {
      return 0;
   }
   return 1 + recursive_node_count(p->next);
}
```

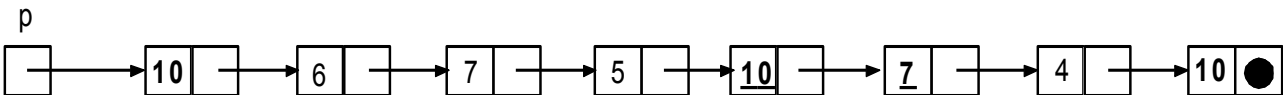
(b)

```
int iterative_node_count(Node *p)
{
   int count = 0;
   Node *tmp = p;
   while(tmp != NULL) {
      ++count;
      tmp = tmp->next;
   }
   return count;
}
```

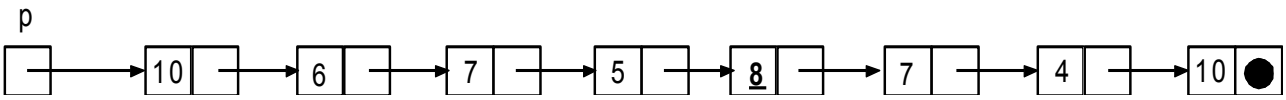
> Complete the code for the function below so that the specifications in the comments are satisfied.

```
void conditional_replace(Node *p, int old, int new,
                         int follower);
/*  Every occurrence of value old in the linked list with first
node pointer p will be replaced by the value new provided the
node after that node contains the value follower. */
```

Example: old has value 10, new has value 8, follower has value 7. Before the function call, the list is:




Then, after the function call, the list would look like this

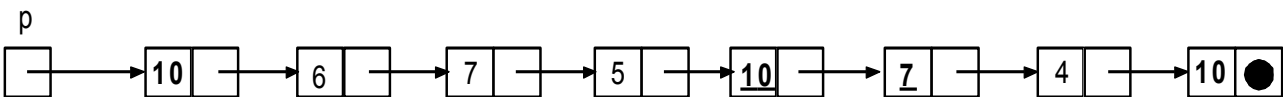



```
void conditional_replace(Node *p, int old, int new,
                         int follower)
{
   Node *tmp = p;
   while(tmp != NULL) {
     if (tmp->info == old){
        if (tmp->next != NULL && tmp->next->info == follower){
          tmp->info = new;
        }
     tmp = tmp->next;
   }
}
```

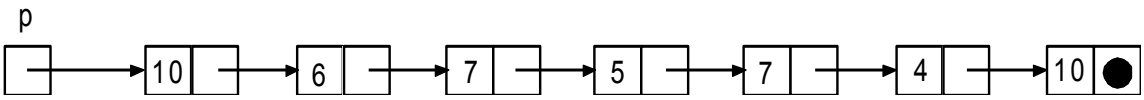
> Complete the code for the function below so that the specifications in the comments are satisfied.

```
void conditional_delete(Node *p, int old, int follower);
/*  Every node containing the value old in the linked list with
first node pointer p will be deleted from the list provided the
next node after the one that contains value old contains the
value follower. */
```

Example: old has value 10 and follower has value 7. Before the function call, the list is:



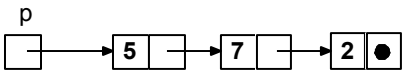

Then, after the function call, the list would look like this



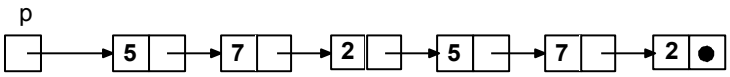

```
void conditional_delete(Node *p, int old, int follower)
{
   /* Must assume that p->info != old */
   if (p == NULL){
     return
   }
   Node *tmp = p;
   Node *q = NULL;
   while(tmp->next != NULL) {
     if (tmp->next->info == old) {
          if (tmp->next->next != NULL
               && tmp->next->next->info == follower) {
                 q = tmp->next;
                 tmp->next = q->next;
                 free(q);
          }
     }
     tmp = tmp->next;
   }
}
```

Complete the code below that adds nodes to the end of a list so as to double the list. That is, if you print the values in the new list from first to last, it looks like the original list concatenated with itself. As a first step, you will want to find the last node in the list before you start adding nodes (use a `Node *` variable called, perhaps, `original_last`).

Example: If, before the function call, the list is:




Then, after the function call, the list would look like this




```
void list_doubler(Node *p)
/*  Adds nodes to a nonempty linked list with first node point p
    so that the list is effectively concatenated with itself */
{
    if (p == NULL) {
       return;
    Node *original_last;
    Node *new_last;
    Node *tmp;

    original_last = p;
    while(original_last->next != NULL)
       original_last = original_last->next;
    /* original_last now points to the last node of the list */

    new_last = original_last;
    tmp = p;
    while(tmp != original_last){
       new_last = MakeNode(tmp->info,NULL);
       new_last = new_last->next;
       tmp = tmp->next;
    }
    new_last->next = MakeNode(tmp->info,NULL);
}
```