Pointers: Parameter Passing and Return

Passing Pointers to a Function

- Pointers are often passed to a function as arguments
  - Allows data items within the calling function to be accessed by the called function, altered, and then returned to the calling function in altered form
  - Useful for returning more than one value from a function
  - Still call-by-value, but now the address is copied, not the content

Example: Swapping

```
int main()
{
   int  a, b;
   a = 5;  b = 20;
   swap (a, b);
   printf (“\n a=%d, b=%d”, a, b);
   return 0;
}

void  swap (int x, int y)
{
   int  t;
   t = x;
   x = y;
   y = t;
}
```

Output

```
a=5, b=20
```

Parameters passed by value, so changes done on copy, not returned to calling function

Example: Swapping using pointers

```
int main()
{
   int  a, b;
   a = 5;  b = 20;
   swap (&a, &b);
   printf (“\n a=%d, b=%d”, a, b);
   return 0;
}

void swap (int *x, int *y)
{
   int  t;
   t = *x;
   *x = *y;
   *y = t;
}
```

Output

```
a=20, b=5
```

Parameters passed by address, changes done on the value stored at that address, correctly swapped

- While passing a parameter to a function, when should you pass its address instead of the value?
  - Pass address if both these conditions are satisfied
    - The parameter value will be modified inside the function body
    - The modified value is needed in the calling function after the called function returns
- Consider the swap function to see this

Passing Arrays as Pointers

Both the forms below are fine in the function body, as arrays are passed by passing the address of the first element. Calling function calls it the same way

```
int main()
{
    int  n;
    float   list[100], avg;
    :
    avg  =  average (n, list);
    :
}

float  average  (int a, float x[])
{
    :
    sum = sum + x[i];
}
```

```
int main()
{
    int  n;
    float   list[100], avg;
    :
    avg  =  average (n, list);
    :
}

float  average  (int a, float *x)
{
    :
    sum = sum + x[i];
}
```

Returning multiple values from a function

- Return statement can return only one value
- What if we want to return more than one value?
- Use pointers
  - Return one value as usual with a return statement
  - For other return values, pass the address of a variable in which the value is to be returned

Example: Returning max and min of an array

Both returned through pointers (could have returned one of them through return value of the function also)

```
int main()
{
   int n, min, max, i, A[100];
   scanf("%d", &n);
   for (i=0; i<n; ++i)
     scanf("%d", &A[i]);
   MinMax(A, n, &min, &max);
   printf("Min and max are %d,
%d", min, max);
   return 0;
}
```

```
void MinMax(int A[], int n, int
*min, int *max)
{
   int i, x, y;
   x = y = A[0];
   for (i=1; i<n; ++i) {
     if (A[i] < x) x = A[i];
     if (A[i] > y) y = A[i];
   }
   *min = x; *max = y;
}
```

Example: Passing structure pointers

```
struct complex  {
     float  re;
     float  im;
};

int main()
{
   struct complex a, b, c;
   scanf("%f%f", &a.re, &a.im);
   scanf("%f%f", &b.re, &b.im);
   add(&a, &b, &c) ;
   printf("\n %f %f", c.re,
c.im);
   return 0;
}
```

```
void add (struct complex
*x, struct complex*y,
struct complex*t)
{
   t->re = x->re + y->re;
   t->im = x->im + y->im;
}
```

The program will print the sum of a and b correctly. Just try passing a, b, c directly (no pointers in call or in function declaration) and see what happens

Strings

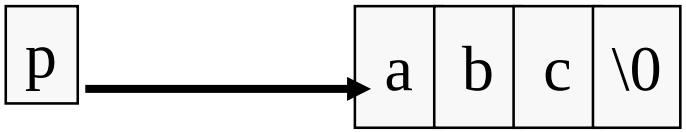
Strings

- 1-d arrays of type char
- By convention, a string in C is terminated by the end-of-string sentinel ‘\0’ (null character)
- char s[21] - can have variable length string delimited with \0
  - Max length of the string that can be stored is 20 as the size must include storage needed for the ‘\0’
- String constants : “hello”, “abc”
- “abc” is a character array of size 4

String Constant

- A string constant is treated as a pointer
- Its value is the base address of the string

```
char *p = “abc”;
```

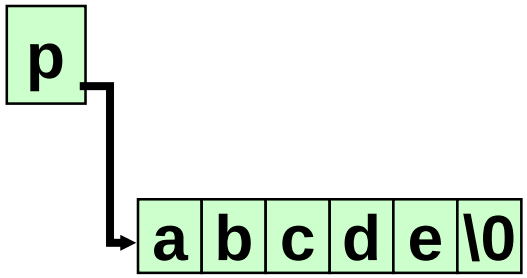



```
printf (“%s %s\n”,p,p+1); /* abc bc is printed */
```

Differences : array & pointers

char *p = “abcde”;

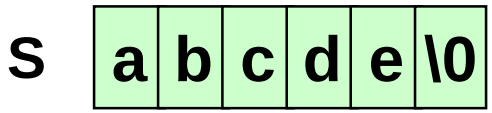
The compiler allocates space for p, puts the string constant “abcde” in memory somewhere else, initializes p with the base address of the string constant



char s[ ] = “abcde”;

≡ char s[ ] = **{‘a’,’b’,’c’,’d’,’e’.’\0’};**

The compiler allocates 6 bytes of memory for the array s which are initialized with the 6 characters



Library Functions for String Handling

- You can write your own C code to do different operations on strings like finding the length of a string, copying one string to another, appending one string to the end of another etc.
- C library provides standard functions for these that you can call, so no need to write your own code
- To use them, you must do

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

At the beginning of your program (after #include <stdio.h>)

String functions we will see

- strlen : finds the length of a string
- strcat : concatenates one string at the end of another
- strcmp : compares two strings lexicographically
- strcpy : copies one string to another

strlen()

int strlen(const char *s)

- Takes a null-terminated strings (we routinely refer to the char pointer that points to a null-terminated char array as a string)
- Returns the length of the string, not counting the null (\0) character

You cannot change contents of s in the function

```
int strlen (const char *s)  {
    int n;
    for (n=0; *s!='\0'; ++s)
        ++n;
    return n;
}
```

strcat()

- char *strcat (char *s1, const char *s2);
- Takes 2 strings as arguments, concatenates them, and puts the result in s1. Returns s1. Programmer must ensure that s1 points to enough space to hold the result.

You cannot change contents of s2 in the function

```
char *strcat(char *s1, const char *s2)
{
    char *p = s1;
    while (*p != '\0')   /* go to end */
        ++p;
    while(*s2 != '\0')
        *p++ = *s2++;  /* copy */
    *p = '\0';
    return s1;
}
```

Dissection of the strcat() function

char *p = s1;

p is being initialized, not *p. The pointer p is initialized to the pointer value s1. Thus p and s1 point to the same memory location

Dissection of the strcat() function

char *p = s1;

p is being initialized, not *p. The pointer p is initialized to the pointer value s1. Thus p and s1 point to the same memory location

while (*p != '\0') ++p;

As long as the value pointed to by p is not '\0', p is incremented, causing it to point at the next character in the string. When p points to \0, the control exits the while statement

Dissection of the strcat() function

char *p = s1;

p is being initialized, not *p. The pointer p is initialized to the pointer value s1. Thus p and s1 point to the same memory location

while (*p != '\0') ++p;

As long as the value pointed to by p is not '\0', p is incremented, causing it to point at the next character in the string. When p points to \0, the control exits the while statement

while(*s2 != '\0') *p++ = *s2++; /* copy */

At the beginning, p points to the null character at the end of string s1. The characters in s2 get copied one after another until end of s2

Dissection of the strcat() function

char *p = s1;

p is being initialized, not *p. The pointer p is initialized to the pointer value s1. Thus p and s1 point to the same memory location

while (*p != '\0') ++p;

As long as the value pointed to by p is not '\0', p is incremented, causing it to point at the next character in the string. When p points to \0, the control exits the while statement

while(*s2 != '\0') *p++ = *s2++; /* copy */

At the beginning, p points to the null character at the end of string s1. The characters in s2 get copied one after another until end of s2

***p = '\0';** put the '\0' at the end of the string

strcmp()

int strcmp (const char *s1, const char *s2);

Two strings are passed as arguments. An integer is returned that is less than, equal to, or greater than 0, depending on whether s1 is lexicographically less than, equal to, or greater than s2.

strcmp()

int strcmp (const char *s1, const char *s2);

Two strings are passed as arguments. An integer is returned that is less than, equal to, or greater than 0, depending on whether s1 is lexicographically less than, equal to, or greater than s2.

```
int strcmp(char *s1, const char *s2)
{
    for (;*s1!='\0'&&*s2!='\0'; s1++,s2++)
    {
        if (*s1>*s2) return 1;
        if (*s2>*s1) return -1;
    }
    if (*s1 != '\0') return 1;
    if (*s2 != '\0') return -1;
    return 0;
}
```

Important: When you use strcmp() from the string library, check the return value for >, < or = 0, not for +1, -1, and 0 (which are just one possible return value to satisfy the >, <, and = 0 condition

strcpy()

```
char *strcpy (char *s1, char *s2);
```

The characters is the string s2 are copied into s1 until \0 is moved. Whatever exists in s1 is overwritten. It is assumed that s1 has enough space to hold the result. The pointer s1 is returned.

strcpy()

char *strcpy (char *s1, const char *s2);

The characters is the string s2 are copied into s1 until '\0' is moved. Whatever exists in s1 is overwritten. It is assumed that s1 has enough space to hold the result. The pointer s1 is returned.

```
char * strcpy (char *s1, const char *s2)
{
    char *p = s1;
    while (*p++ = *s2++) ;
    return s1;
}
```

Example: Using string functions

```
int main()
{
char s1[ ] = "beautiful big sky country",
      s2[ ] = "how now brown cow";
printf("%d\n",strlen (s1));
printf("%d\n",strlen (s2+8));
printf("%d\n", strcmp(s1,s2));
printf("%s\n",s1+10);
strcpy(s1+10,s2+8);
strcat(s1,"s!");
printf("%s\n", s1);
return 0;
}
```

Output

```
25
9
-1
big sky country
beautiful brown cows!
```

Practice Problems

1. Write a function to search for an element in an array of integers that returns 1 if the element is found, 0 otherwise. If found, it also returns the index in the array where found
2. Write a function that returns the number of lowercase letters, uppercase letters, and digit characters in a string
3. Define a structure POINT to store the coordinates (integer) of a point in 2-d plane. Write a function that returns the two farthest (largest distance) points in an array of POINT structures
4. Write a function that takes two arrays of integers A and B and returns the size of the union set and the size of the intersection set of A and B
5. Write a function that returns the lengths of the largest palindromes formed by any substring (sequence of consecutive characters) of the string. It should also return the index in the string from which the palindrome starts.

For all of the above, add suitable main() functions to call the functions. Also, decide on what parameters you will need; for better practice, for all problems other than problems 1, assume that the return type of the function is void.