Character Strings – Q1 (sweepSpace)

Write two versions of a C function that remove all the blank spaces in a sentence.

The first version **sweepSpace1()** will use **array** notation for processing the string, and the other version **sweepSpace2()** will use **pointer** notation.

Sample input and output session:

Enter the string:
i am a boy
sweepSpace1(): iamaboy
sweepSpace2(): iamaboy

The function prototypes are given below:

char *sweepSpace1(char *sentence);

// use array notation for accessing array elements

char *sweepSpace2(char *sentence);

// use pointer notation for accessing array elements

Write a C program to test the function.

Character Strings – Q1 (sweepSpace)

```
#include <stdio.h>
#include <string.h>
char *sweepSpace1(char *str);
char *sweepSpace2(char *str);
int main()
{
   char str[80],str2[80], *p;

   printf("Enter the string: \n");
   fgets(str, 80, stdin);
   if (p=strchr(str,'\n')) *p = '\0';
   strcpy(str2,str);
   printf("sweepSpace1(): %s\n", sweepSpace1(str));
   printf("sweepSpace2(): %s\n", sweepSpace2(str2));
   return 0;
}
```

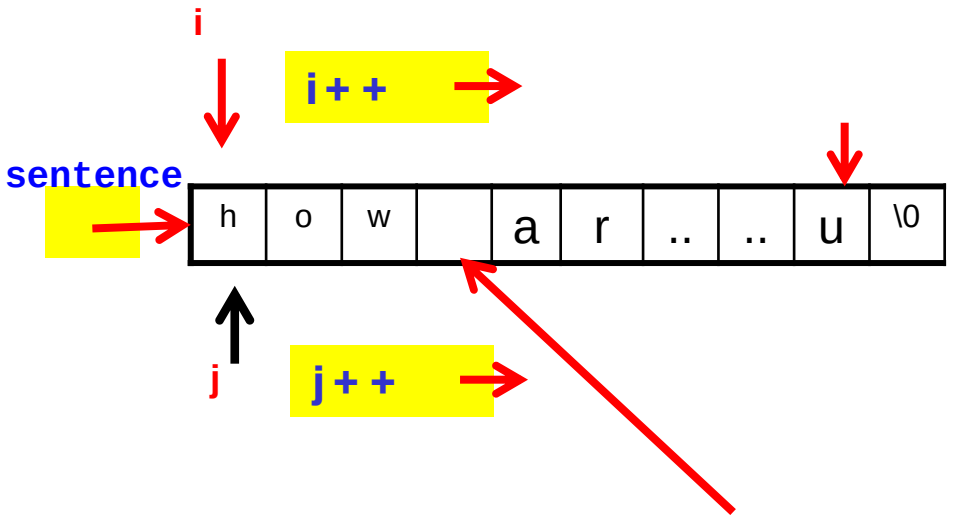
Sample input and output session:

Enter the string:
i am a boy
sweepSpace1(): iamaboy
sweepSpace2(): iamaboy

Character Strings – Q1 (sweepSpace)

Using array index for processing

```
char *sweepSpace1(char *sentence){
   int i, j, len;

   len = strlen(sentence);
   j = 0;
   for ( i=0; i < len; i++){
      if (sentence[i] != ' '){
         sentence[j]=sentence[i];
         j++;
      }
   }
   sentence[j] = '\0';
   return sentence;
}
```



When space is encountered, i is updated to move to the next index position, but j is not. So space is skipped.

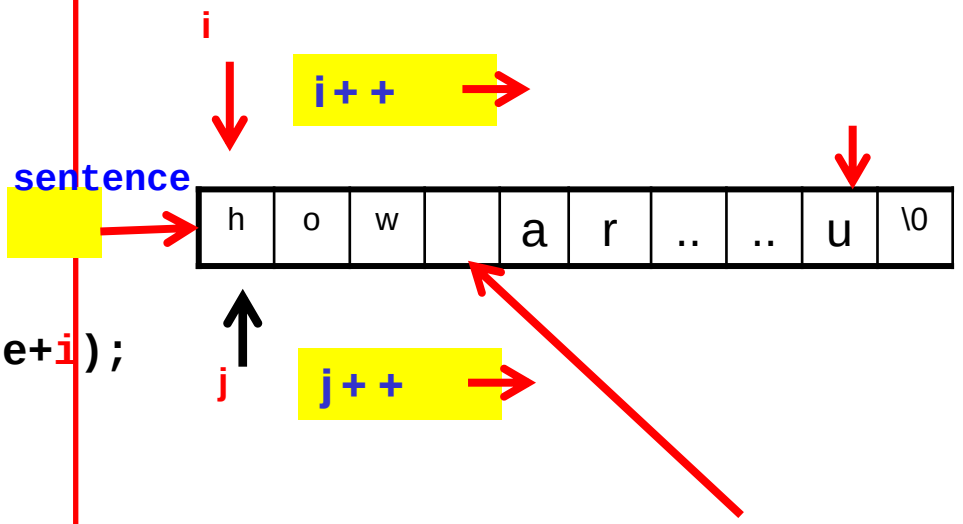
Note:

- Traverse each element of the character string using a for loop.
- Using a if statement to update the string if a space is encountered

Character Strings – Q1 (sweepSpace)

Using pointer for processing

```
char *sweepSpace2(char *sentence){
   int i, j, len;

   len = strlen(sentence);
   j = 0;
   for ( i=0; i < len; i++){
      if (*(sentence+i) != ' '){
         *(sentence+j)=*(sentence+i);
         j++;
      }
   }
   *(sentence+j) = '\0';
   return sentence;
}
```



When space is encountered, i is updated to move to the next index position, but j is not. So space is skipped.

Note:

- Traverse each element of the character string using a for loop.
- Using a if statement to update the string if a space is encountered

Character Strings – Q2 (findTarget)

Write a C program that reads and searches character strings. In the program, it contains a function **findTarget()** that searches whether a target name string has been stored in the array of strings. The function prototype is

```
int findTarget(char *target,
    char nameptr[SIZE][80],  int size);
```

where *nameptr* is the array of strings entered by the user, *size* is the number of names stored in the array and *target* is the target string.

If the target string is found, the function will return its **index** location, or -1 if otherwise.

In addition, it also contains the function **readNames()** that reads a number of names from the user. The function prototype is given as follows:

```
void readNames(char nameptr[][80], int *size);
```

where *nameptr* is the array of strings to store the input names, and *size* is a pointer parameter which passes the number of names to the caller. Similarly, the function prototype of **printNames()** is given as follows:

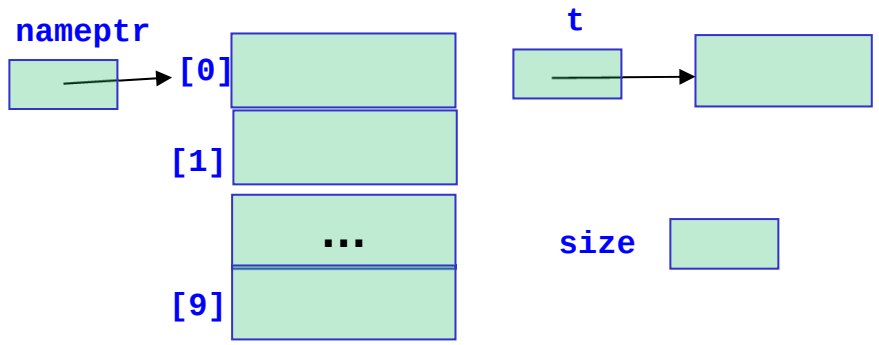
```
void printNames(char nameptr[][80], int size);
```

Sample input and output sessions:

```
Enter size:
4
Enter 4 names:
Peter Paul John Mary
Enter target name:
John
findTarget(): 2

Enter size:
5
Enter 5 names:
Peter Paul John Mary Vincent
Enter target name:
Jane
findTarget(): -1
```

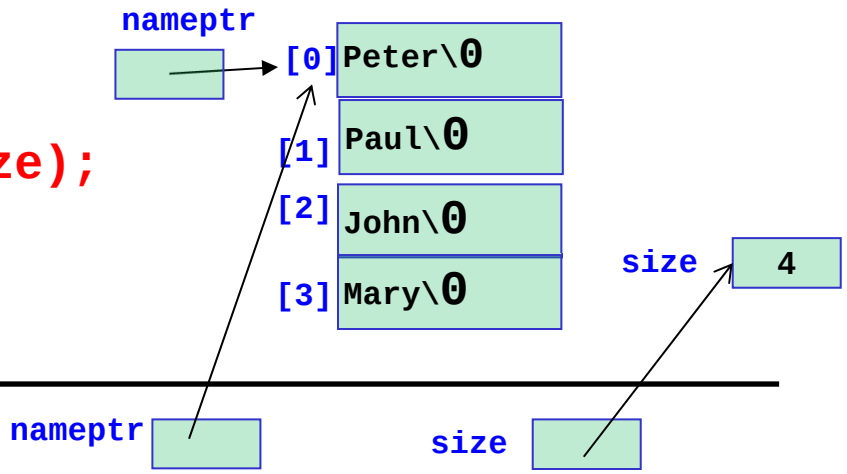
Character Strings – Q2 (findTarget)

```
#include <stdio.h>
#include <string.h>
#define SIZE 10
#define INIT_VALUE 999
void printNames(char nameptr[][80], int size);
void readNames(char nameptr[][80], int *size);
int findTarget(char *target, char nameptr[][80], int size);
int main(){
   char nameptr[SIZE][80], t[40],*p;
   int size, result = INIT_VALUE;  int choice;
   printf("Select one of the following options: \n");
   printf("1: readNames()\n");
   printf("2: findTarget()\n");
   printf("3: printNames()\n");
   printf("4: exit()\n");
   do {
      printf("Enter your choice: \n");
      scanf("%d", &choice);
      switch (choice) {
         case 1: readNames(nameptr, &size);
            break;
         case 2: printf("Enter target name: \n");
            scanf("\n");
            fgets(t, 80, stdin);
            if (p=strchr(t,'\n')) *p = '\0';
            result = findTarget(t, nameptr, size);
            printf("findTarget(): %d\n",  result);
            break;
         case 3: printNames(nameptr, size);
            break;}
   } while (choice < 4);
   return 0;}
```



Character Strings – Q2 (findTarget)

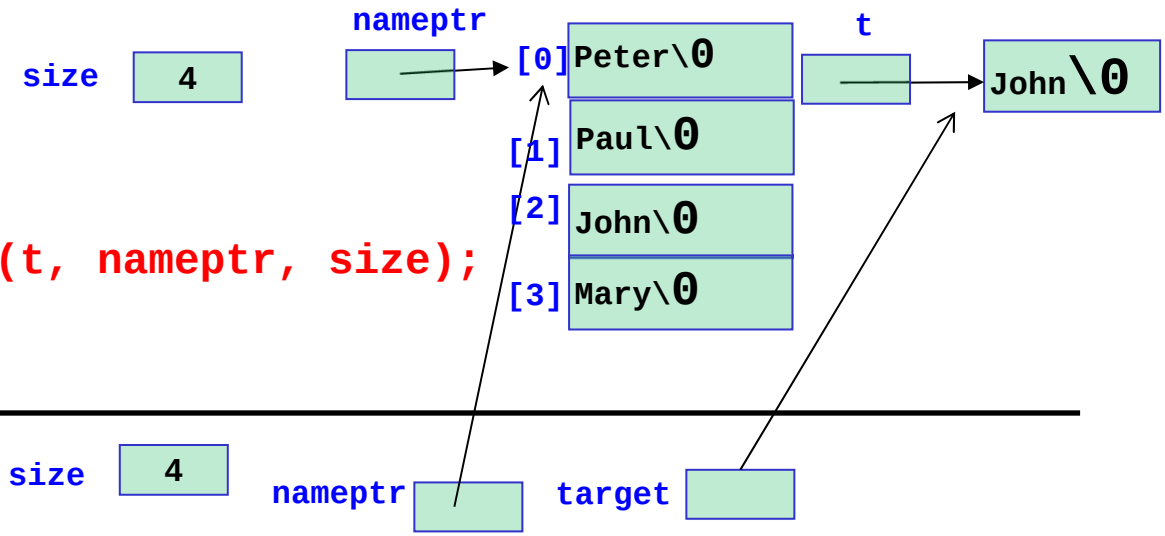
```
int main(){


  case 1: readNames(nameptr, &size);


}
```




```
void readNames(char nameptr[][80], int *size)
{
   int i;

   printf("Enter size: \n");
   scanf("%d", size);
   printf("Enter %d names: \n", *size);
   for (i=0; i < *size; i++)
      scanf("%s", nameptr[i]);
}
```

Character Strings – Q2 (findTarget)

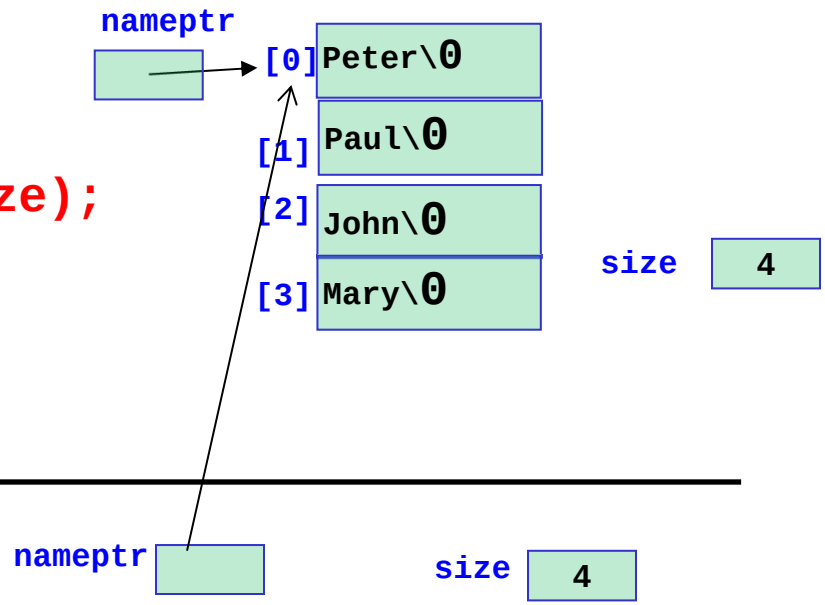
```
int main(){


  case 2:
    gets(t);
    result = findTarget(t, nameptr, size);

}
```




```
int findTarget(char *target, char nameptr[SIZE][80], int size)
{
   int i;

   for (i=0; i < size; i++) {
      if (strcmp(nameptr[i], target) == 0)
         return i;
   }
   return -1;
}
```

Character Strings – Q2 (findTarget)

```
int main(){


  case 3: printNames(nameptr, size);



}
```




```
void printNames(char nameptr[][80], int size)
{
   int i;

   for (i=0; i<size; i++)
      printf("%s ", nameptr[i]);
   printf("\n");
}
```

Character Strings – Q3 (palindrome)

Write a function palindrome() that reads a character string and determines whether or not it is a palindrome.

A palindrome is a sequence of characters that reads the same forwards and backwards.

For example, "abba" and "abcba" are palindromes, but "abcd" is not.

The function returns 1 if it is palindrome, or 0 if otherwise.

The function prototype is given as follows:

```
int palindrome(char *str);
```

Write a C program to test the function.

Sample input and output sessions:

Test Case 1:
Enter a string:
abcba
palindrome(): A palindrome

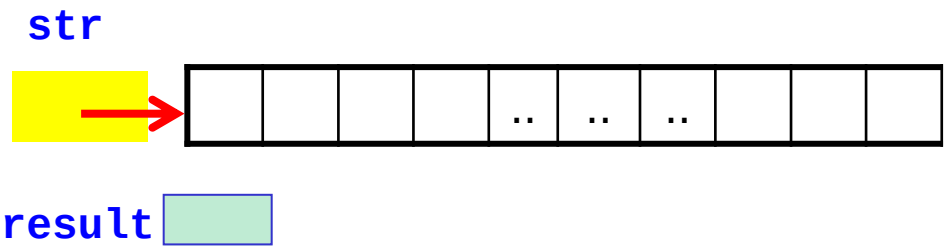
Test Case 2:
Enter a string:
abba
palindrome(): A palindrome

Test Case 3:
Enter a string:
abcde
palindrome(): Not a palindrome

Test Case 4:
Enter a string:
abb a
palindrome(): Not a palindrome

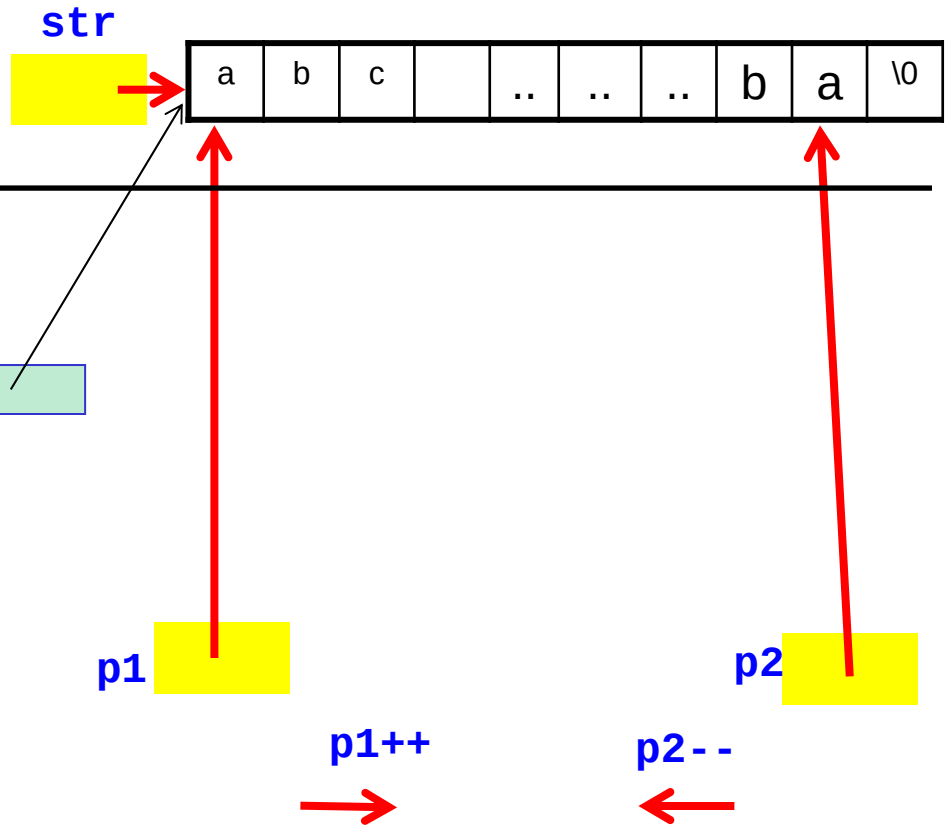
Character Strings – Q3 (palindrome)

```
#include <stdio.h>
#define INIT_VALUE -1000
int palindrome(char *str);
int main()
{
   char str[80];
   int result = INIT_VALUE;

   printf("Enter a string: \n");
   gets(str);
   result = palindrome(str);
   if (result == 1)
      printf("palindrome(): A palindrome\n");
   else if (result == 0)
      printf("palindrome(): Not a palindrome\n");
   else
      printf("An error\n");
   return 0;
}
```



Character Strings – Q3 (palindrome)

```
int main(){
    result
}
```




```
int palindrome(char *str)
{
   int len, i;
   char *p1, *p2;

   i=0; len=0;
   while (*(str+i)!='\0') {
      i++; len++;
   }
   p1 = str;
   p2 = str+len-1;
   while (p1 < p2){
      if (*p1 != *p2)
         break;
      else {
         p1++; p2--;
      }
   }
   if (p1 < p2) return 0;
   else return 1;
}
```