

WEEK 5 LAB - CHARACTER STRINGS

1. (**sweepSpace**) Write two versions of a C function that remove all the blank spaces in a string. The first version `sweepSpace1()` will use array notation for processing the string, while the other version `sweepSpace2()` will use pointer notation. The function prototypes are given below:

```
char *sweepSpace1(char *str);
char *sweepSpace2(char *str);
```

A sample program template is given below to test the functions:

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

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

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

    i=0; len=0;
    while (str[i]!='\0'){
        len++;
        i++;
    }
    j = 0;
    for (i=0; i < len; i++)
    {
        if (str[i] != ' ')
            // update the string by removing any space detected
    }
    str[j] = '\0'; // add a null character
    return str;
}

char *sweepSpace2(char *str)
{
    /* Write your program code here */
}
```

Some sample input and output sessions are given below:

- (1) Test Case 1:
Enter the string:
i am a boy
sweepSpace1(): iamaboy

```
sweepSpace2(): iamaboy
```

(2) Test Case 2:

Enter the string:

anybody

```
sweepSpace1(): anybody
```

```
sweepSpace2(): anybody
```

2. (**findTarget**) Write a C program that reads and searches character strings. In the program, it contains the 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[][80], int size);
```

where *nameptr* is the array of strings, *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, the program also contains the functions `readNames()` and `printNames()`. The function `readNames()` 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. The function prototype of `printNames()` which prints the names is given as follows:

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

A sample program template is given below for testing the functions:

```
#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;
}
void printNames(char nameptr[][80], int size)
{
    /* Write your program code here */
}
void readNames(char nameptr[][80], int *size)
{
    /* Write your program code here */
}
int findTarget(char *target, char nameptr[][80], int size)
{
    /* Write your program code here */
}

```

Some sample input and output sessions are given below:

- (1) Test Case 1:
Select one of the following options:

1: readNames()
2: findTarget()
3: printNames()
4: exit()

Enter your choice:

1

Enter size:

4

Enter 4 names:

Peter Paul John Mary

Enter your choice:

2

Enter target name:

John

findTarget(): 2

Enter your choice:

4

- (2) Test Case 2:
Select one of the following options:

1: readNames()
2: findTarget()
3: printNames()
4: exit()

Enter your choice:

1

Enter size:

5

Enter 5 names:

Peter Paul John Mary Vincent

Enter your choice:

2

Enter target name:

Jane

findTarget(): -1

Enter your choice:

4

(3) Test Case 3:

Select one of the following options:

1: readNames()

2: findTarget()

3: printNames()

4: exit()

Enter your choice:

1

Enter size:

5

Enter 5 names:

Peter Paul John Mary Vincent

Enter your choice:

3

Peter Paul John Mary Vincent

(4) Test Case 4:

Select one of the following options:

1: readNames()

2: findTarget()

3: printNames()

4: exit()

Enter your choice:

1

Enter size:

6

Enter 6 names:

Peter Paul John Mary Vincent Joe

Enter your choice:

2

Enter target name:

Joe

findTarget(): 5

Enter your choice:

4

3. (**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);
```

A sample program template is given below for testing the function:

```
#include <stdio.h>
#include <string.h>
#define INIT_VALUE -1000
int palindrome(char *str);
```

```

int main()
{
    char str[80], *p;
    int result = INIT_VALUE;

    printf("Enter a string: \n");
    fgets(str, 80, stdin);
    if (p=strchr(str, '\n')) *p = '\0';
    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;
}
int palindrome(char *str)
{
    /* Write your code here */
}

```

Some test input and output sessions are given below:

- (1) Test Case 1:
 Enter a string:
abcba
 palindrome(): A palindrome
- (2) Test Case 2:
 Enter a string:
abba
 palindrome(): A palindrome
- (3) Test Case 3:
 Enter a string:
abcde
 palindrome(): Not a palindrome
- (4) Test Case 4:
 Enter a string:
abb a
 palindrome(): Not a palindrome