**F.16 Chapter 16 Solutions**

# 16.1

int changeToPL(char \* word)

{

int i = 1;

char first = word[0];

if (first = '\0') return 1;

while (word[i] != '\0')

word[i  1] = word [i];

word[i] = first; word[i + 1] = 'a';

word[i + 2] = 'y';

word[i + 3] = '\0';

}

**16.3** x = 7

# 16.5

void insertionSort(char\* list[])

{

int unsorted;

int sorted;

char \*unsortedItem;

/\* This loop iterates from 1 thru MAX\_NUMS \*/ for(unsorted = 1; unsorted < MAX\_NUMS; unsorted++)

{

unsortedItem = list[unsorted];

/\* This loop iterates from unsorted thru 0, unless we hit an element smaller than current item \*/

for(sorted = unsorted  1;

(sorted >= 0) && (StringCompare(list[sorted], unsortedItem) == 2);

sorted)

list[sorted+1] = list[sorted];

list[sorted +1] = unsortedItem; /\* Insert Item \*/

}

}

* 1. A snapshot of the run-time stack is shown in the table below. Memory values are shown in the right-most column.

|  |  |  |
| --- | --- | --- |
| 0xEFF8 | ind | 0xEFFA |
| 0xEFF9 | ptr | 0xEFFA |
| 0xEFFA | apple | 125 |
| 0xEFFB | saved frame pointer | … |
| 0xEFFC | saved return address | … |
| 0xEFFD | return value | … |

/\* The list contains MAXNUMS integers \*/

/\* Also, all duplicate elements are converted to 0 \*/

void RemoveDuplicates(int list[])

{

int i;

int j;

int unique\_list = 0;

int found;

for (i = 0; i < MAXNUMS; i++) {

found = 0;

for (j = 0; j < unique\_list; j++) {

if (list[j] == list[i])

found = 1;

}

if (!found) {

list[unique\_list] = list[i];

unique\_list++;

}

}

/\* clean up the remainder of the list \*/

for (j = unique\_list; j < MAXNUMS; j++)

list[j] = 0;

return;

}

a. Findlen = 5 (return value, return address, saved frame pointer, 1 parameter, 1 local variable)

main = 13 (return value, return address, saved frame pointer, 0 parameters, 1 local variable of   
10 location)

b.

|  |  |  |
| --- | --- | --- |
| 0xEFEC | len | 5 |
| 0xEFED | saved frame pointer | 0xEFFA |
| 0xEFEE | saved return address |  |
| 0xEFEF | return value | 5 |
| 0xEFF0 | s | 0xEFF6 |
| 0xEFF1 | str[0] | ‘a’ |
| 0xEFF2 | str[1] | ‘p’ |
| 0xEFF3 | str[2] | ‘p’ |
| 0xEFF4 | str[3] | ‘l’ |
| 0xEFF5 | str[4] | ‘e’ |
| 0xEFF6 | str[5] | ‘\0’ |
| 0xEFF7 | str[6] | … |
| 0xEFF8 | str[7] | … |
| 0xEFF9 | str[8] | … |
| 0xEFFA | str[9] | … |
| 0xEFFB | saved frame pointer | … |
| 0xEFFC | saved return address | … |
| 0xEFFD | return value | … |

c. The activation record for main would contain the first ten characters of the string as shown in the table above. The extra characters would overwrite the saved frame pointer and return address in the activation record for main, causing unknown and unexpected behavior when main returns to its caller.

int Push(int item)

{

if (topOfStack == STACK\_SIZE)

return 1;

else {

stack[topOfStack] = item;

topOfStack++;

return 0;

}

}

int Pop(int \*item)

{

if (topOfStack == 0)

return 1;

else {

topOfStack;

\*item = stack[topOfStack];

return 0;

}

}