CSE102
Computer Programming with C

2016-2017 Fall Semester

Dynamic Data Structures
Examples and Teasers

Pointers and Function Arguments

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

int main()
{
        int a = 10, b = 20;
        swap(a, b);
        printf("a: %d b: %d\n", x, y);
}

OUTPUT:
```

Pointers and Function Arguments

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

int main()
{
        int a = 10, b = 20;
        swap(a, b);
        printf("a: %d b: %d\n", x, y);
}

OUTPUT: a: 10 b: 20
```

Pointers and Function Arguments

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

int main()
{
        int a = 10, b = 20;
        swap(&a, &b);
        printf("a: %d b: %d\n", x, y);
}

OUTPUT:
```

Pointers and Function Arguments

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

int main()
{
      int a = 10, b = 20;
      swap(&a, &b);
      printf("a: %d b: %d\n", x, y);
}
```

OUTPUT:

```
expected 'int' but argument is of type 'int *'
void swap(int x, int y)
```

Pointers and Function Arguments

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

int main()
{
        int a = 10, b = 20;
        swap(&a, &b);
        printf("a: %d b: %d\n", x, y);
}

OUTPUT:
```

Pointers and Function Arguments

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

int main()
{
        int a = 10, b = 20;
        swap(&a, &b);
        printf("a: %d b: %d\n", x, y);
}

OUTPUT: a: 20 b: 10
```

A strlen(…) Implementation

```
strlen("Hello World!");    /* string constant */
strlen(array);             /* char array[100] */
strlen(ptr);               /* char *ptr */


/* strlen --> return length of string str */
```

A strlen(…) Implementation

```
strlen("Hello World!");    /* string constant */
strlen(array);             /* char array[100] */
strlen(ptr);               /* char *ptr */


/* strlen --> return length of string str */
int strlen(char *str)
```

A strlen(…) Implementation

```
strlen("Hello World!");    /* string constant */
strlen(array);             /* char array[100] */
strlen(ptr);               /* char *ptr */


/* strlen --> return length of string str */
int strlen(char *str)
{
      int n;

      for (n = 0; *str != '\0'; str++)
            n++;

      return n;
}
```

A strlen(…) Implementation

```
strlen("Hello World!");    /* string constant */
strlen(array);             /* char array[100] */
strlen(ptr);               /* char *ptr */


strlen(&array[2]);
```

A strlen(…) Implementation

```
strlen("Hello World!");    /* string constant */
strlen(array);             /* char array[100] */
strlen(ptr);               /* char *ptr */


strlen(&array[2]);

ptr = &array[0];
strlen(ptr);
```

A strlen(…) Implementation

```
strlen("Hello World!");    /* string constant */
strlen(array);             /* char array[100] */
strlen(ptr);               /* char *ptr */


strlen(&array[2]);

ptr = &array[0];
strlen(ptr);

ptr = &array[10];
strlen(ptr);
```

A strlen(…) Implementation

```
strlen("Hello World!");    /* string constant */
strlen(array);             /* char array[100] */
strlen(ptr);               /* char *ptr */


strlen(&array[2]);

ptr = &array[0];
strlen(ptr);

ptr = &array[10];
strlen(ptr);

ptr = array;
strlen(ptr);
```

Character Pointers

```
/* strcpy --> copy source to dest */
```

Character Pointers

```
/* strcpy --> copy source to dest */
strcpy(char *source, char * dest)
{
      int i;

      i = 0;
      while ((dest[i] = source[i]) != '\0')
            i++;
}
```

Character Pointers

```
/* strcpy --> copy source to dest */
strcpy(char *source, char * dest)
{
      int i;

      i = 0;
      while ((dest[i] = source[i]) != '\0')
            i++;
}

strcpy(char *source, char * dest)
{
      while ((*dest = *source) != '\0') {
            source++;
            dest++;
      }
}
```

Character Pointers

```
/* strcpy --> copy source to dest */
strcpy(char *source, char * dest)
{
      int i;

      i = 0;
      while ((dest[i] = source[i]) != '\0')
            i++;
}

strcpy(char *source, char * dest)
{
      while ((*dest++ = *source++) != '\0')
            ;
}
```

List Rotation

Write a function that rotates a list clockwise by *n* elements. For example {1,2,3,4,5,6,7} rotated by *3* becomes {4,5,6,7,1,2,3}.

List Rotation

```
int *rotate_list(int n, int *list, int list_size)
{




}
```

List Rotation

```
int *rotate_list(int n, int *list, int list_size)
{
      if (n > list_size)
      {
            printf("Wrong parameter n\n");
            return 0;
      }
      int i, k;
      int *list_rotated = (int *)malloc(list_size);

      for (i = 0,k = n; k >= 0; i++,k--)
            list_rotated[list_size-k] = list[i];
      for (i = 0,k = n; i < list_size-n; i++,k++)
            list_rotated[i] = list[k];

      return list_rotated;
}
```

List Rotation

```
int i, n = 3;
int list[] = {1,2,3,4,5,6,7};
int list_size = sizeof(list)/sizeof(list[0]);

printf("Original List:\n");
for (i = 0; i < list_size; i++)
      printf("%d ", list[i]);
printf("\n");
int *list_rotated = rotate_list(n, list, list_size);

if (list_rotated > 0)
{
      printf("Rotated List by %d:\n", n);
      for (i = 0; i < list_size; i++)
            printf("%d ", list_rotated[i]);
      printf("\n");
}
```

List Rotation

Write a function that rotates a list clockwise by *n* elements. For example {1,2,3,4,5,6,7} rotated by *3* becomes {4,5,6,7,1,2,3}. Solve this without creating a copy of the list. How many swap and move operations do you need?

List Rotation Without Copy

```
void rotate_list_wc(int n, int *list, int list_size)
{
}
```

List Rotation Without Copy

```
void rotate_list_wc(int n, int *list, int list_size)
{
      if (n > list_size)
      {
            printf("Wrong parameter n\n");
            return;
      }
      int i, j, k, temp;

      for (i = n-1,k = list_size-1;
           i >= 0 && k < list_size; i--,k--)
      {
            temp = list[i];
            for (j = i; j < k; j++)
                  list[j] = list[j+1];
            list[k] = temp;
      }
}
```

List Rotation Without Copy

```
int i, n = 3;
int list[] = {1,2,3,4,5,6,7};
int list_size = sizeof(list)/sizeof(list[0]);

printf("Original List:\n");
for (i = 0; i < list_size; i++)
      printf("%d ", list[i]);
printf("\n");
rotate_list_wc(n, list, list_size);

printf("Rotated List by %d:\n", n);
for (i = 0; i < list_size; i++)
      printf("%d ", list[i]);
printf("\n");
```

List Rotation

WHAT ABOUT IF LIST IS A LINKED LIST ?