Dynamic Memory

Dynamic vs Static

```
int staticInt;
int *pDynamicInt;

int staticArray[100];
int *pDynamicArray;
```

Allocation in C

```
#include <stdlib.h>

void *malloc(size_t size);
void free(void* ptr);
```

- //calloc realloc

Allocate one int

Allocate an Array of ints

```
int *pArray;
const int SIZE = 1024;

pArray  =       malloc(




free(
```

readelf -a

Memory Layout

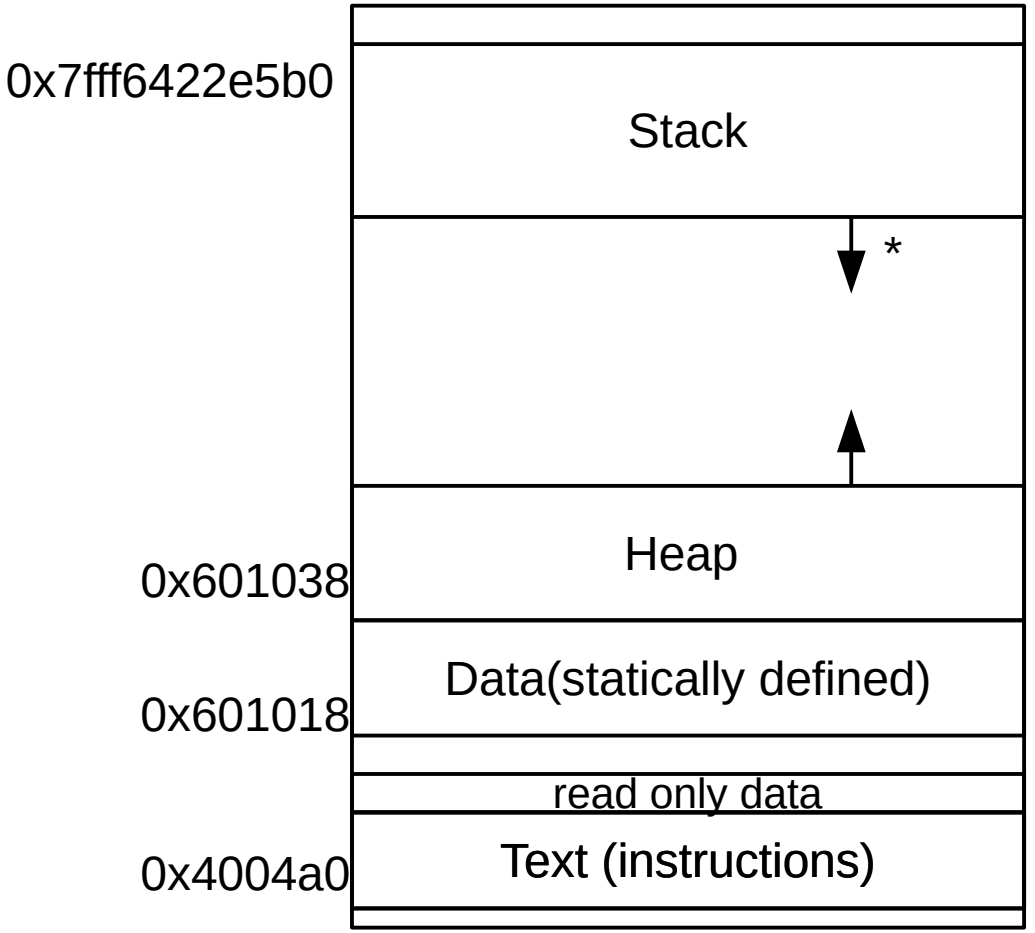
```
#include <stdio.h>
#include <stdlib.h>

int gValue = 9;
int gArray[1024];

int main()
{
  int *pArray;
  int value = 10;
  printf("%d", gValue);
  pArray =       malloc(

  free(pArray);
  return 0;
}
```

shared libraries?



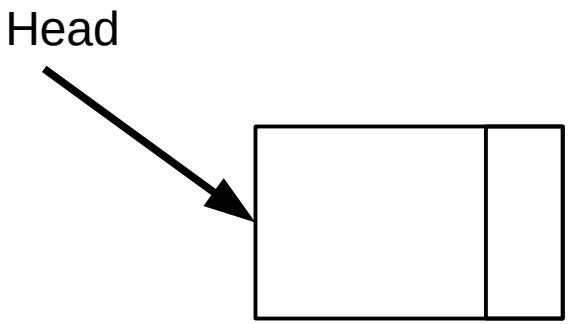
Linked List ADT

```
typedef struct ListElement* ListElementPtr;
typedef struct List* ListPtr;

typedef struct ListElement {
   int data;
   ListElementPtr psNext;
} ListElement;

typedef struct List {
   ListElementPtr psHead;
   ListElementPtr psTail;        // optional
   ListElementPtr psCurrent;
} List;
```

Linked Lists



Legal?

```
List sList;
ListPtr psList;

sList.psHead.data = 5;
sList->psHead = NULL;
sList = NULL;
psList->psHead->data = 5;
psList = NULL;
```

Practice

- Set psHead to NULL to denote an empty list.
- Add a node that contains 10
- Add a node that contains 20 after the node that contains 10
- Add a node that contains 15 between node 10 and 20
- A linked list exists pointed to by the list pointer **psList**. Write a function **length** that accepts the list pointer to a singly linked list and returns the number of nodes in the list.

Code

```
ListPtr psList;
```

File IO

previously used fgetc

```
#include <stdio.h>

int result, x, y;
FILE *pFile;
pFile = fopen("data/test.txt", "r");


result = fscanf(pFile, "%d %d", &x, &y);

fclose(pFile);

// what does fprintf() do?
```

Void*

- A pointer that can point at any type

```
void* pData;
int myInt;
char myChar;

pData = &myInt;
pData = &myChar;
```

In a List

```
typedef struct ListElement* ListElementPtr;
typedef struct List* ListPtr;

typedef struct ListElement {
   void* data;
   ListElementPtr psNext;
} ListElement;

typedef struct List {
   ListElementPtr psHead;
   ListElementPtr psTail;     // optional
   ListElementPtr psCurrent;
} List;
// add a node that points to 'C'
// add a node that points to 300
```

- ArrayOfVoidStars.c

```c
#include <stdio.h>
#include <stdlib.h>

#define MAX_DATA 10

int main ()
{
  void *pData[MAX_DATA];
  int i;

  puts ("PROGRAM START");

  for (i = 0; i < MAX_DATA; ++i)
  {
    if (i % 2 == 0)
    {
      pData[i] = malloc (sizeof (int));
      *(int *) pData[i] = i;
    }
    else
    {
      pData[i] = malloc (sizeof (double));
      *(double *) pData[i] = (double) i;
    }
  }

  // Write a loop for each of the following ... i.e. 4 different loops
  // 1. Free all dynamically allocated memory
  // 2. Pointers 0 through MAX_DATA - 1 are to point to 'A', 'B', 'C',...
  // 3. Print out each character
  // 4. Free all dynamically allocated memory


  puts ("PROGRAM END");

  return 0;
}
```

Pointer Arithmetic

```
const int SIZE =10;
int *paInts = (int*) malloc(sizeof(int) * SIZE);
int *pWalker;
int i = 0;
                                  // pWalker points to
pWalker = paInts;                    //
for( i = 0; i < SIZE; ++i )
{
   *pWalker = i*i;                   //
   pWalker++;                        //
   printf("%d ", *(paInts + i) );
}
```

Pointer Arithmetic

```
const int SIZE =10;
void *paChars = malloc(sizeof(char) * SIZE);
char *pWalker;
int i = 0;


pWalker = (char*) paChars;
for( i = 0; i < SIZE; ++i )
{
   *(pWalker + i) = ‘A’+i;
   printf(“%c ”, *(((char*) paChars) + i) );
}
```

C Strings

```
const int MAX = 5;

char* szData[MAX];
char* szClass = “CS300”;
char* szName = “Ada”;

// copy Ada into szData
strcpy(szData,

// copy CS300 into szData
strcpy(szData,

szClass[2] = ‘4’; // ??
```