Cache Lab Helpful Hints

Code samples taken from www.tutorialspoint.com

Pointers

- A pointer in C is a variable whose value is the address of another variable
- Can also get address of variable using “&”

```
#include <stdio.h>

int main () {

   int  var1;
   char var2[10];

   printf("Address of var1 variable: %x\n", &var1  );
   printf("Address of var2 variable: %x\n", &var2  );

   return 0;
}
```

```
#include <stdio.h>

int main () {

   int  var = 20;   /* actual variable declaration */
   int  *ip;        /* pointer variable declaration */

   ip = &var;  /* store address of var in pointer variable*/

   printf("Address of var variable: %x\n", &var  );

   /* address stored in pointer variable */
   printf("Address stored in ip variable: %x\n", ip );

   /* access the value using the pointer */
   printf("Value of *ip variable: %d\n", *ip );

   return 0;
}
```

C memory management

- In C you must manage your own memory
  - Using malloc and free
- In Java, anytime you used “new” you were allocating memory
  - If you want to make a new object in C, must malloc
- malloc takes the size as a parameter, and returns an address to the beginning of the allocated region of memory
- When finished, use free to deallocate

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

int main() {
  char name[100];
  char *description;
  strcpy(name, ”Jeannie Albrecht”);

  /* allocate memory dynamically */
  description = malloc( 200 * sizeof(char) );

  if( description == NULL ) {
    fprintf(stderr, "Error - unable to allocate required memory\n");
  } else {
    strcpy( description, ”Jeannie Albrecht is a CS professor.");
  }

  printf("Name = %s\n", name );
  printf("Description: %s\n", description );

  /* release memory using free() function */
  free(description);
}
```

fgets

- fgets reads file from stream and stores in string
- fopen opens files for reading/writing

```
#include <stdio.h>

int main () {
  FILE *fp;
  char str[60];

  /* opening file for reading */
  fp = fopen("file.txt" , "r");
  if(fp == NULL) {
    perror("Error opening file");
    return(-1);
  }
  if( fgets (str, 60, fp)!=NULL ) {
    /* writing content to stdout */
    puts(str);
  }
  fclose(fp);

  return(0);
}
```

sscanf

- sscanf reads formatted input from string

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

int main () {
  int day, year;
  char weekday[20], month[20], dtm[100];

  strcpy( dtm, "Saturday March 25 1989" );
  sscanf( dtm, "%s %s %d  %d", weekday, month, &day, &year );

  printf("%s %d, %d = %s\n", month, day, year, weekday );

  return(0);
}
```

getopt

- Used to parse command line options

```
int main (int argc, char **argv) {
  int c;
  char *cvalue = NULL;

   while ((c = getopt (argc, argv, ”ac:")) != -1)
   switch (c) {
     case 'a’:
         //do something
        break;
      case 'c':
        cvalue = optarg;
        break;
      default:
       //do something
     }
}
```

Cache lab in a nutshell

- Define a struct(s) for representing your cache
- Write functions for:
  - main (get command line options, open trace file, read trace file, etc)
  - Initializing cache (i.e., malloc space for cache)
  - "Freeing" cache (i.e., any allocated memory must be freed)
  - Running simulation (update the flags of our cache accordingly)
  - Other helper functions as needed

Goals for first lab meeting

- Have the basic infrastructure in place
  - Parse command line opts
  - Read from trace file
  - Call (potentially empty) functions for cache simulation (i.e., initCache, freeCache, etc)

- Develop an implementation plan

  Define cache structures (C struct types)

  Plan functions and what order to implement