Computer Science 1510

Lecture 26

Lecture Outline

- Pointers
- Arguments to main

Pointers

- Pointers are variables that store memory addresses.
- Pointers are declared according to their type (ie. `int`, `float`, etc.), and are distinguished by an asterisk prepended to the variable name.
- Syntax:

```
type *identifier;
```

- Example:

```
int *ptr;
```

- Typically, a pointer of a given type is only used to point to a variable of that same type.
- To store the address of a variable in a pointer variable, you use the & ("address of") operator. For example,

```
ptr=&index;
```

- One then says that `ptr` "points to" `index` , that is, the memory address where `index` is stored.

Pointers

- A pointer can be considered a *reference* to a variable, that is, you can use it to *refer* to the variable that it points to.

- To obtain the value stored in the location to which a pointer is pointing, we must *dereference* the pointer. To do this we use the dereferencing operator *.

- For example,

```
int a=1,b=2,c;    /* declare three integer variables */
int *ptr1,*ptr2;  /* declare two integer pointers */
ptr1=&a;  /* ptr1 points to the address of a */
ptr2=&b;  /* ptr2 points to the address of b */
c=*ptr1;  /* c is set to the value referenced
           * by ptr1 (a) */
*ptr1=3;  /* the value pointed to by ptr1 (a) is
           * set to 3 */
ptr2=ptr1; /* set ptr2 to point to the same location
            * as ptr1 */
```

Example: Pointers

```
#include <stdio.h>
int main (int argc, char *argv[]) {
   int a=1,b=2,c;
   int *ptr1,*ptr2;
   ptr1=&a; ptr2=&b;
   printf("a=%d,b=%d,c=%d,ptr1=%p,ptr2=%p\n",
           a,b,c,ptr1,ptr2);
   printf("a=%d,b=%d,c=%d,*ptr1=%d,*ptr2=%d\n",
           a,b,c,*ptr1,*ptr2);
   c=*ptr1;
   printf("a=%d,b=%d,c=%d,*ptr1=%d,*ptr2=%d\n",
           a,b,c,*ptr1,*ptr2);
   *ptr1=3;
   ptr2=ptr1;
   printf("a=%d,b=%d,c=%d,*ptr1=%d,*ptr2=%d\n",
           a,b,c,*ptr1,*ptr2);
   return 0;
}
```

Output:

```
a=1,b=2,c=134518400,ptr1=0xbfac985c,ptr2=0xbfac9858
a=1,b=2,c=134518400,*ptr1=1,*ptr2=2
a=1,b=2,c=1,*ptr1=1,*ptr2=2
a=3,b=2,c=1,*ptr1=3,*ptr2=3
```

Example: Arrays

```
// compute average, maximum, and minimum assignment grades
// using a function for average, maximum, and minimum


#include <stdio.h>

double compute_avg(int *array,int n);
int compute_max(int *array,int n);
int compute_min(int *array,int n);

int main(int argc, char *argv[])
{
    int assign[99],i,n;
    double assign_avg;
    int assign_max,assign_min;

    printf("How many assignment grades would you like to enter?\n");
    scanf("%d",&n);

    /* Read in the assignment marks */
    for(i=0;i<n;i++){
       printf("Enter assignment mark %d\n",i);
       scanf("%d",&assign[i]);
    }

    assign_avg=compute_avg(assign,n);
    printf("The average assignment mark is %lf\n",assign_avg);

    assign_max=compute_max(assign,n);
    assign_min=compute_min(assign,n);
    printf("The maximum assignment mark is %d\n",assign_max);
    printf("The minimum assignment mark is %d\n",assign_min);

    return 0;
}
```

```
double compute_avg(int *array, int n){
    double sum;
    int i;
    sum=0.0;
    for(i=0;i<n;i++){
        sum+=array[i];
    }
    return sum/n;
}

int compute_max(int *array, int n){
    int max;
    int i;
    max=array[0];
    for(i=1;i<n;i++){
        if (array[i]>max) max=array[i];
    }
    return max;
}

int compute_min(int *array, int n){
    int min;
    int i;
    min=array[0];
    for(i=1;i<n;i++){
        if (array[i]<min) min=array[i];
    }
    return min;
}
```

Example: Swapping two variables

Using pointers, it is possible to write a function to swap two variables.

Consider the following program:

```
#include <stdio.h>
void swap (int *x, int *y);

int main (int argc, char *argv[])
{
   int i = 1, j = 2;
   printf("%d %d\n", i, j);
   swap(&i, &j);
   printf("%d %d\n", i, j);
   return 0;
}

void swap (int *x, int *y){ // x, y are pointers to i, j in the main program
   int temp;
   temp = *x; // save value of x
   *x = *y; // replace value of x with value of y
   *y = temp; // set value of y in x
   return;
}
```

Output:

```
1 2
2 1
```

Arguments to main

- So far we have been writing the initial line of a main function,

```
int main(int argc, char *argv[])
```

  without using the arguments `argc`, and `argv`.

- These variables allow arguments to be passed to the main function on the command line.

- For example, consider a program called myprog.c that was compiled with the line,

```
gcc -Wall -o myprog myprog.c
```

- This would create an executable called `myprog`.

- If we wanted to execute `myprog` and pass it two integers, we could do so on the command line as follows:

```
myprog 5 3
```

Arguments to main

- The first argument of main, `argc`, contains the number of arguments entered on the command line. In the above case `argc` would equal 3 since the executable name counts as an argument.

- The second argument of main, `argv`, is a pointer to an array of `'\0'`-terminated strings.

- `argv[0]` is the name of the executable.

- In the above example, `argv[1]` would point to the first string `"5"`, while `argv[2]` would point to the second string `"3"`.

- Since each element of argv is a string, to use the `5` and `3` as integers, we must first extract the numerical values. This can be done with the `sscanf` function.

C: sscanf

- `sscanf` takes a string and extracts the specified elements from it.

- Syntax:

  ```
  sscanf(string,description,&variable-list)
  ```

  where `string` is a character string (ie. an array of characters, `char *s`), `description` is the format of the elements to be read from the string, and `variable-list` is a list of the variables where the individual elements will be stored.

- For example, if we had a string called `buff` that looked like "1 2 3", then we could extract the individual numbers using the following `sscanf`:

  ```
  sscanf(buff,"%d %d %d",&a,&b,&c);
  ```

Example 1: sscanf

```
#include <stdio.h>

int main(int argc, char *argv[])
{
   int a,b,c;
   /* Read 3 ints from string "1 2 3" */
   sscanf("1 2 3","%d %d %d",&a,&b,&c);
   printf("%d, %d, %d\n",a,b,c);
   return 0;
}
```

Output:

```
1, 2, 3
```

Example 2: sscanf

```
#include <stdio.h>

int main(int argc, char *argv[])
{
   int a,b,c;
   /* Declare and initialize string of length 256 */
   char buff[256]="1 2 3";
   /* Read 3 ints from buff */
   sscanf(buff,"%d %d %d",&a,&b,&c);
   printf("%d, %d, %d\n",a,b,c);
   return 0;
}
```

Output:

```
1, 2, 3
```

Example 3: sscanf

```
#include <stdio.h>

int main(int argc, char *argv[])
{
   int a,b,c;
   char *buff; /* Declare character pointer */
   buff="1 2 3"; /* Point buff to the string "1 2 3" */
   /* Read 3 ints from buff */
   sscanf(buff,"%d %d %d",&a,&b,&c);
   printf("%d, %d, %d\n",a,b,c);
   return 0;
}
```

Output:

```
1, 2, 3
```

Example: argc and argv

```
#include <stdio.h>

int main(int argc, char *argv[])
{
   int a,b;
   printf("There were %d command line arguments\n",argc);
   if (argc!=3) {
     printf("USAGE: %s int int\n",argv[0]);
     return -1;
   }

   printf("argv[0]=%s, argv[1]=%s, argv[2]=%s\n",
           argv[0],argv[1],argv[2]);
   sscanf(argv[1],"%d",&a);
   sscanf(argv[2],"%d",&b);
   printf("a=%d, b=%d\n",a,b);
   return 0;
}
```