CS11001/CS11002 Programming and Data Structures (PDS)

(Theory: 3-1-0)

Passings functions to other functions: pointers to functions

Pointers to functions

- A function like a variable has an address location in the memory.
- Thus we can have pointer to a function.
  - which can be passed as an argument to another function.
  - we call the function which is passed as the guest function.
  - we call the function to which the function is passed as the host function.

Example

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

void funct1(int i, float f)
{
  printf("%d %f\n",i,f);
}

int func2(char *s)
{
 printf("%s\n",s);
}
```

Example

```
main()
{
 char s[50];
 void (*p)(int, float);
 int (*q)(char *), i=5;
 float f=1.23;
 puts("Enter a string:");
 gets(s);
 puts(s);
 p= funct1;
 q=func2;
 p(i,f);
 q(s);
}
```

Declarations

- When a host function accepts a pointer to a guest function, the declaration of the host function is as follows:

host-funct-data-type host-function-name (guest-funct-data-type (* guest-function-name)(arg_type 1, arg_type 2,…);

Example

```
#include<stdio.h>

int large(int *a, int n)
{
 int big, i;
 big=a[0];
 for(i=1;i<n;i++)
   if(big<a[i])
      big=a[i];
 return big;
}
```

```
int small(int *a, int n)
{
 int small, i;
 small=a[0];
 for(i=0;i<n;i++)
   if(small>a[i])
        small=a[i];
 return small;
}
```

Function with pointer to function as argument

```
void select(int *b, int m1, int (*q)(int *, int))
{
 int ans;
 ans=q(b,m1);
 printf("%d\n",ans);
}
```

Passing pointer to function to a function

```
main()
{
 int n, i, a[20], (*ptr)(int *, int);
 printf("Enter the no of integers\n");
 scanf("%d",&n);
 for(i=0;i<n;i++)
   scanf("%d",&a[i]);

 ptr=large;
 printf("Largest value is:\n");
 select(a,n,ptr);
 printf("Smallest value is: \n");
 select(a,n,small);
}
```

More pointers

- What does this mean:

  `int *(*p)(int (*a)[ ]); ?`

More pointers

- What does this mean:

  int *(*p)(int (*a)[ ]); ?
- int *(*p)(…) means p is a pointer to a function which returns pointer to an integer.
- int (*a)[ ] indicates pointer to an array. This is the argument of the function.
- Thus, *(*p)(int (*a)[ ]) represents a pointer to a function whose argument is a pointer to an array.
- Thus, int *(*p)(int (*a)[ ]) represents a pointer to a function whose argument is a pointer to an array and which returns pointer to an integer.
  - What is **int *(*p[10])(int (*a)[ ]);** ?

Structures

What are structures?

- It is a heterogeneous user defined.
  - a structure may contain different data types.
- Example:

```
#define MAXLEN 100
struct student {
char name[MAXLEN];
char roll[MAXLEN];
int height;
float cgpa;
};
```

Defining instances of structure data

- This declaration gives a user-defined data of type struct student.
- That has four members: two character arrays of names name and roll, an integer named height and a floating point value named cgpa.
- The struct declaration only defines a data type but no instances of data of this type.
- In order to declare specific instances of structure data, one should employ the usual variable declaration procedure.
  - struct student thatStudent, FBStudents[60], *studPointer;

Another Example

- A second example is provided by complex numbers which can be represented as pairs of real numbers.
- One can use the following structure:

```
struct comp {
   double real;
   double imag;
 };
```

- One then uses the declaration

```
struct comp z, z1, z2;
```

  to obtain specific instances of complex numbers.

Type definitions

- The typedef declarations are used to *rename* data types in C.
- For example, if one plans to work with unsigned long long int variables, but plans not to write that big a name, one may define the following short-cut:

```
typedef unsigned long long int ull;
```

  After this definition, the data type unsigned long long int can be called also as ull, i.e., one can declare variables as:

```
ull n, array[100], *ptr;
```

Typedefs using user defined data types

- One can also typedef pointers and arrays:

```
typedef ull *ullPointer;
typedef ull ullArray[128];
```

- Here we assume that unsigned long long int is already typedef'd as ull.
- We use this definition to typedef two other data types.
- First, ullPointer is defined to be a pointer to an unsigned long long int, whereas ullArray is defined to be an array of 128 unsigned long long int data.
- One can instantiate data of these types in the usual way:
  - ullPointer p; defines a pointer variable p, whereas
  - ullArray A; defines an array A of 128 unsigned long long int data.

Typedefs for structures

- In an analogous way, one can use typedef's to give short single-word names to structure data types.
- For example, the declarations

```
typedef struct stud student;
typedef struct comp complex;
```

  give na~~mes student and complex to the user-~~ defined data types struct stud and struct comp.
- The following variable declarations are legal after these typedef's:
  - student thatStudent, FBStudents[60], *studPointer;
  - complex z, z1, z2;

Using typedef while defining structures

- One can combine a struct declaration and a subsequent typedef as follows:

-
  ```
  typedef struct stud
   {
     char name[MAXLEN];
     char roll[MAXLEN];
      int height;
      float cgpa;
    } student;
  ```

-
  ```
  typedef struct
  { double real;
    double imag;
  } complex;
  ```

Initializing structures

- Structures can be initialized much in the same way as arrays can be –
  - by a curly-braced comma-separated list of initializing constant values for the individual members.
- For example, the above student record can be initialized as:

  ```
  struct stud thatStudent = { "Foolan Barik", "03FB1331", 175, 9.81 };
  ```

  or with the typedef'd name as:

  ```
  student thatStudent = { "Foolan Barik", "03FB1331", 175, 9.81 };
  ```
- Initializing values populate (fill up) the members of the variable in the same order as they appear in the struct declaration.
  - For the above example, the string name receives the value "Foolan Barik", roll the value "03FB1331", height the value 175 and cgpa the value 9.81.

Accessing members of structures

- Accessing individual members of a structure is different from what is done with arrays.
- Now one should write the name of a structure variable followed by a dot (.) and then by the formal name given to the member.
- For example, if thatStudent is initialized as above, thatStudent.name refers to the string "Foolan Barik", thatStudent.roll refers to the string "03FB1331", thatStudent.height refers to the integer value 175 and thatStudent.cgpa to the floating point value 9.81.

Accessing members of array of structures

- If we have an array of structures, one first uses square brackets to refer to an element of the array and then uses dot and a member name to access the corresponding member of the structure,
- For example, FBStudents[5].height refers to the height field of the element at index 5 in the array FBStudents.

Accessing members of a pointer to a structure

- If we have a pointer to a structure, we first dereference the pointer in order to obtain the structure and then write dot and the member name.
- For example, if studPointer is a pointer to a struct stud, the notation (*studPointer).roll refers to the string holding the roll number of the student whose records are pointed to by the pointer studPointer.
- There is an alternative way of writing the same thing: studPointer->roll.
- The dereferencing * and the dot . are combined to the symbol -> which C designers deemed to be an intuitive and natural notation.

Example

- The following function computes the average CGPA of the students of the Department of Foobarnautic Engineering:
- float avCGPA (struct stud FBStudents[] , int n)

```
{ float sum = 0; int i;
  for (i=0; i<n; ++i)
      sum += FBStudents[i].cgpa;
 return (sum/(float)n); }
```

Example

- Here is how you can do the same with pointers:
- float avCGPA2 (struct stud FBStudents[] , int n)

```
{ float sum = 0;
  int i; struct stud *p; //define a pointer to the structure
  p = FBStudents; //p points to the structure stud
FBStudents
   for (i=0; i<n; ++i)
    { sum += p->cgpa;
      ++p; }
  return (sum/(float)n); }
```