CS19003: Programming and Data Structures Laboratory

Aritra Hazra,
CSE Dept., IIT Kharagpur

http://cse.iitkgp.ac.in/~aritrah/course/lab/PDS/Spring2021/

08-Jun-2021

Aritra Hazra, CSE Dept., IIT Kharagpur

Tutorial

Pointers and Dynamic Memory Allocation

Pointers

VARIABLE that stores memory address

```
void main(){
   int i;
   int *ptr; //pointer to an int

   i = 4; /* store the value 4 into the
   memory location associated with i */
   ptr = &i; /* store the address of i
   into the memory location associated
   with ptr */
   *ptr = *ptr + 1;
   printf("%d\n", i);   //i=5
}
```

More examples

- argument for scanf()

```
scanf("%d %d", &data1, &data2);
```

Pass address of variable where you want result stored

- Declarations: (all have same meaning)

```
int* x, y;
int *x, y;
int *x; int y;
```

The $*$ operator binds to the variable name, not the type

Relationship between Arrays and Pointers

An array name is essentially a pointer to the first element in the array

```
char data[10];
/* data = addr where first element
is located = &data[0] */
char *cptr;
cptr = data; /* points to data[0] */
```

Pointers and Arrays

```
char data[10];
/* data = addr where first element
is located = &data[0] */
```

data	&data[0]
(data + n)	&data[n]
$*$data	data[0]
$*$(data + n)	data[n]

Pointers and Arrays

```
int main(void) {
int a[N] = {84, 67, 24, ...};
/*
&a[0] = a+0 = D000
&a[1] = a+1 = D004
&a[2] = a+2 = D008

a[0] = *a      = 84
a[1] = *(a+1) = 67
a[2] = *(a+2) = 24
*/

return 0;
}
```

Passing Pointers as Function Arguments

Alter variables outside a function's own scope

```
void swap(int *first, int *second);
int main(){
   int a = 4, b = 7;
   printf("pre-swap: a=%d, b=%d\n",a,b)
   swap(&a, &b);
   printf("post-swap: a=%d, b =%d\n",a,b)
   return 0;
}
void swap(int *first, int *second){
   int temp;
   temp = *first;
   *first = *second;
   *second = temp;
}
```

Passing Pointers as Function Arguments

```
void swap(int *first, int *second);
int main(){
   int a = 4, b = 7;
   printf("pre-swap: a=%d, b=%d\n",a,b)
   swap(&a, &b);
   printf("post-swap: a=%d, b =%d\n",a,b)
   return 0;
}
void swap(int *first, int *second){
   int temp;
   temp = *first;
   *first = *second;
   *second = temp;
}
```

The address-of operator (&) is used to pass the address of the two variables rather than their values

Passing Array as Function Argument

```
#define N 64
int average(int b[], int n) {
  int i, sum; // same as int *b
  //receives the value D000 from main
  for (i = 0; i < n; i++)
    sum += b[i];
  return sum / n;
}
int main(void) {
  int a[N] = {84, 67, 24, ..., 89, 90};
  printf("%d\n", average(a, N));
  return 0; //passes &a[0] = D000
}
```

Advantage? working with subarray

```
#define N 64
int average(int b[], int n) {
  int i, sum; // same as int *b
  //receives the value D000 from main
  for (i = 0; i < n; i++)
    sum += b[i];
  return sum / n;
}
int main(void) {
  int a[N] = {84, 67, 24, ..., 89, 90};
  printf("%d\n", average(a+5, 10));
  return 0; //passes &a[5] = D020
}
```

- compute average of a[5] through a[14]

Dynamic memory allocation

```
#include<stdio.h>
#include<stdlib.h>
int max(int a[], int c, int *b);
int main(){
 int i, j, m, *a;
 printf("enter number of elements\n");
 scanf("%d",&i);
 a=(int *)malloc(i * sizeof(int));
 for(j=0;j<i;j++){
   printf("enter element no. %d:",j);
   scanf("%d", &a[j]);
  }
 m=max(a, i, &j); // next slide
 printf("Max value is %d stored in a[%d]\n",m,j);
 return 0;
}
```

Returning multiple values

```
int max(int *a, int i, int *j)
{
    int k, max=-32767;
    for (k=0; k<i; k++)
    {
        if (a[k]>max)
        {
            max=a[k];
            *j=k;
        }
    }
    return(max);
}
```

Multi-dimensional Arrays

One-dimensional arrays are quite able to represent many natural collections. There are some other natural collections that may better be conceptualized as 2-dimensional data. Example: a matrix.

0	1	2	3
4	5	6	7
8	9	10	11

Two-Dimensional Arrays


```
int a[3][4] = {
 {0, 1, 2, 3} ,    /*for row index 0 */
 {4, 5, 6, 7} ,    /*for row index 1 */
 {8, 9, 10, 11}    /*for row index 2 */
};
```

or

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

Matrix addition

```
int m, n, c, d;
int first[9][9], second[9][9], sum[9][9];
printf("Enter no. of rows and columns\n");
scanf("%d%d", &m, &n);
printf("Enter the elements of matrix1\n");
for ( c = 0 ; c < m ; c++ )
  for ( d = 0 ; d < n ; d++ )
    scanf("%d", &first[c][d]);
/*now scan 2nd matrix*/

for ( c = 0 ; c < m ; c++ )
  for ( d = 0 ; d < n ; d++ )
    sum[c][d] = first[c][d] + second[c][d];
```

Thank You