Pointers and Arrays

Dr. Achraf EL Allali

Pointers

- C provides two unary operators, & and *, for manipulating data using pointers

- The operator &, when applied to a variable, results in the address of the variable. This is the address operator

- The operator *, when applied to a pointer, returns the value stored at the address specified by the pointer. This is the dereferencing or indirection operator

Variable

- Allocate 1 byte of memory

  char a;
- Memory allocated at some particular address

  char *ptr;

  ptr = &a;
- All pointers are of the same size
  - they hold the address
  - generally 4 bytes

More Pointers

```
int x = 1, y = 2, z[10];
int *ip;  /* ip is a pointer to int */
ip = &x;  /* ip now points to x */
y = *ip;  /* y is now 1 */
*ip = 0;  /* x is now 0 */
ip = &z[0]; /* ip now points to z[0] */
```

What is the output?

```
int x = 1, y = 2, z[10];
int *ip;
ip = &x;
*ip = *ip + 1;
printf(“%d %d %d”, x, y, *ip);
y = *ip + 1;
printf(“%d %d %d”, x, y, *ip);
*ip += 1;
printf(“%d %d %d”, x, y, *ip);
```

What is the output?

```
int x = 1, y = 2, z[10];
int *ip;
ip = &x;
*ip = *ip + 1;
printf(“%d %d %d”, x, y, *ip);    /* 2 2 2 */
y = *ip + 1;
printf(“%d %d %d”, x, y, *ip);    /* 2 3 2 */
*ip += 1;
printf(“%d %d %d”, x, y, *ip);    /* 3 3 3 */
```

Swap two numbers

```
void swap (int x, int y) /*wrong*/
{
        int temp;
        temp = x;
        x = y;
        y = temp;
}
main()
{
        int a = 5, b = 3;
        swap (a, b);
        printf (“After swap: %d %d”, a, b);
}
```

Swap two numbers

```
void swap (int *px, int *py)
{
        int temp;
        temp = *px;
        *px = *py;
        *py = temp;
}
main()
{
        int a = 5, b = 3;
        swap (&a, &b);
        printf (“After swap: %d %d”, a, b);
}
```

Arrays and Pointers

```
main()
{
        int a[4] = {11, 12, 13, 14};
        int x, y;
        int *pa;
        pa = &a[0];
        x = *pa;
        y = *(pa + 2);
        printf ("%d %d %d %d", *pa, x, y, a[2]);
}
```

Arrays and Pointers

```
main()
{
        int a[4] = {11, 12, 13, 14};
        int x, y;
        int *pa;
        pa = &a[0];
        x = *pa;
        y = *(pa + 2);
        printf ("%d %d %d %d", *pa, x, y, a[2]); /* 11, 11, 13, 13 */
}
```

Arrays and Pointers

- pa = a; is equivalent to pa = &a[0];
- x = a[i]; is equivalent to x= *(a + i);

Arrays and Pointers

```
int mystrlen (char s[])
{
        int i, len = 0;
        for (i = 0; s[i] != '\0'; i++)
                len++;
        return len;
}
main()
{
        char str[10];
        strcpy(str, “CSC215”);
        printf(“%d”, mystrlen(str));
}
```

Arrays and Pointers

```
int mystrlen (char *s)
{
        int i, len = 0;
        for (;  *s != '\0'; s++)
                len++;
        return len;
}
main()
{
        char str[10];
        strcpy(str, “CSC215”);
        printf(“%d”, mystrlen(str));
}
```

Arrays and Pointers

```
int mystrlen (char *s)
{
        int i, len = 0;
        for (;  *s != '\0'; s++)
                len++;
        return len;
}
main()
{
        char str[10];
        scanf(“%s”,str);
        printf(“%d”, mystrlen(str));
}
```

Array of pointers

```
int main()
{
        int x = 11, y = 12, z = 13, i;
        int *a[3];      /*array of pointers*/
        a[0] = &x;
        a[1] = &y;
        a[2] = &z;
        for  (i = 0; i < 3; i++)
                printf("%d\n", *a[i]);
}
```

Pointer arithmetic

- Arithmetic operators “+”, “-”, “++”and “--”can be applied to pointers.
- The result depends on the data type of the pointer.
- The result is undefined if the pointers do not point to the elements within the same array

Pointer arithmetic

```
#include <stdio.h>

const int MAX = 3;

int main ()
{
   int  var[] = {10, 100, 200};
   int  i, *ptr;

   /* let us have array address in pointer */
   ptr = var;
   for ( i = 0; i < MAX; i++)
   {

      printf("Address of var[%d] = %x\n", i, ptr );
      printf("Value of var[%d] = %d\n", i, *ptr );

      /* move to the next location */
      ptr++;
   }
   return 0;
}
```

Pointer arithmetic

The output is something as follow:

```
Address of var[0] = bf882b30
Value of var[0] = 10
Address of var[1] = bf882b34
Value of var[1] = 100
Address of var[2] = bf882b38
Value of var[2] = 200
```

Pointer arithmetic

The unary operators & and * have the same precedence as any other unary operator, with associativity from right to left.

c=*++cp	equivalent to:	c=*(++cp)
c=*cp++		c=*(cp++)
c=++*cp		c=++(*cp)
c=(*cp)++		???

Pointer comparison

The relational operators ==, !=, <, <=, > and >= are permitted between pointers of the same type

- Examples:

  int a[10], *ap;

  ap = &a[7];

  ap < &a[8] is true

  ap < &a[4] is false

Pointer comparison

```
#include <stdio.h>

const int MAX = 3;

int main ()
{
   int  var[] = {10, 100, 200};
   int  i, *ptr;
   /* let us have address of the first element in pointer */
   ptr = var;
   i = 0;
   while ( ptr <= &var[MAX - 1] )
   {

      printf("Address of var[%d] = %x\n", i, ptr );
      printf("Value of var[%d] = %d\n", i, *ptr );

      /* point to the previous location */
      ptr++;
      i++;
   }
   return 0;
}
```

Pointer comparison

The output is something as follow:

```
Address of var[0] = bfdbcb20
Value of var[0] = 10
Address of var[1] = bfdbcb24
Value of var[1] = 100
Address of var[2] = bfdbcb28
Value of var[2] = 200
```

Pointer conversion

A pointer of one type can be converted to a pointer of another type by using an explicit cast:

```
int *ip;
double *dp;
dp= (double *) ip; OR
 ip = (int*) dp;
```