Pointers

Chapter 07

CMPE-112 *Programming Fundamentals*

Lecture Plan

- Sample program
- Basics of Pointers
  - Address and Dereferencing Operators
  - Pointer Type Declaration
  - Pointer Assignment
  - Pointer Initialization
  - Pointer Arithmetic
- Functions and Pointers
- Arrays and Pointers
- Strings. Library String Functions
- Sample Program

Sample Program (I)

```
/* The program works with pointers */
#include  <stdio.h>

int main()
{
  char *cp2;
  char c1, c2;

  puts("\nEnter a character:");
  c1 = getchar();
  cp2 = &c2;          // The pointer now has the address of the variable c2
  *cp2 = c1;          // Copying from c1 to the location pointed by cp2
  puts("The character is as follows:");
  putchar(c2);

  return 0;
}
```

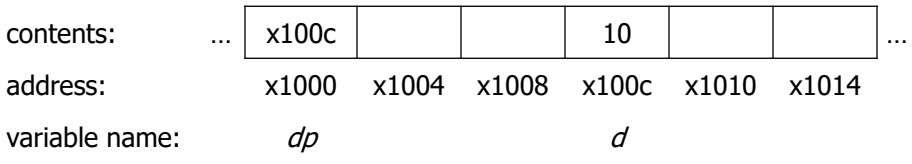

Basics of Pointers

Let's declare an integer variable

int d;

and denote its address in the RAM as *dp*

The diagram below depicts the relationship between *d* and *dp*



&d	*d*	*dp*	**dp*
x100c	10	x100c	10

Address and Dereferencing Operators

- 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
- Examples:

```
j = *ip + 10;                          j = i + 10;
k = ++(*ip);        Equivalent         k = ++i;
x = sqrt( (double) *ip );              x = sqrt( (double) i );
printf("%d", *ip);                     printf("%d", i);
```

5

Pointer Type Declaration

- The operator *&* can only be applied to a variable, so the following expressions are **incorrect**

  &10 *&'C'* *&(x+3)*

  If the type of an operand is *T*, the result is of type *"pointer to T"*
- The operator *** can only be applied to a pointer. If the type of an operand is *"pointer to T"*, the result is of type *T*
- To indicate that a variable contains a pointer to type, an asterisk is included before the variable name:

  type *identifier;

  *char *cp;* *double *mp;* *int *kp;*

6

Pointer Assignment

- A pointer value may be assigned to another pointer of the same type, for example

```
int  i = 1, j, *ip;
ip = &i;
j = *ip;
(*ip)++;
```

- An exception to this rule is the constant zero (the **NULL** pointer, declared in *stdio.h*) that can be assigned to a pointer of any type

```
ip = NULL;
```


Pointer Initialization

- An initial value may be assigned to a pointer at the declaration. The general form is

```
type  *identifier = initial_value;
```

- Examples

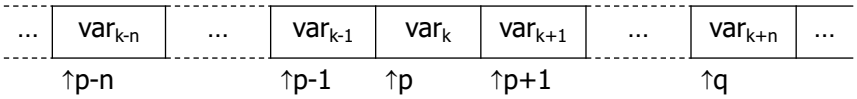
```
int  m;
int  *mp = &m;

double  d[10];
double  *d5p = &d[4];

char  s[] = "A string";
char  *s3p = &s[2];
```

Pointer Arithmetic

- Arithmetic operators "+", "-", "++" and "--" can be applied to pointers. The result depends on the data type of the pointer



- The result of *subtraction of two pointers* is **undefined** if the pointers do not point to the elements within the same array. Otherwise, the result is the number of elements between the two pointers:

  q - p is equal to *n*

Precedence of Operators *&* and *

- The unary operators **&** and ***** have the same precedence as any other unary operators, with the associativity is from right to left
- Special care is required when mixing ***** with **++** or **--** in a pointer expression, so

*c = *++cp;*		*c = *(++cp);*
*c = *cp++;*	Equivalent ⟹	*c = *(cp++);*
*c = ++*cp;*		*c = ++(*cp);*
*c = (*cp)++;*		*? ? ?*

Pointer Comparison

- The relational operators ==, !=, <, <=, > and >= are permitted between pointers (mainly, of the same type)
- Examples:

```
int  a[10], *ap;
ap = &a[7];
ap < &a[8]    is   true
ap < &a[4]    is   false
```

- The following comparisons may be abbreviated:

```
if (ip != NULL)   j += *ip;        Equivalent        if ( ip )  j += *ip;
if (ip == NULL)  puts("Warning");                    if ( !ip )  puts("Warning");
```


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;
```

- G*eneric pointers (void *)* are used to define functions whose formal parameters can accept pointers of any type
- Any pointer may be converted to type *void** and back without loss of information

```
prototype:   void  free(void *);
call:          free(cp);
```

Functions and Pointers

- A function can take a pointer to any data type as argument and can return a pointer to any data type
- Using pointers the programs in C can implement **call by reference**

```
/* The function finds a maximum */
double  *maxp(double *xp, double *yp)
{
  return *xp >= *yp ? xp : yp;
}
.....

{
  double u = 1, v = 2, s;
  double *mp = &s;

  mp = maxp(&u, &v);
  printf("Max = %lf", *mp);
}
```

```
/* The function exchanges two values */
void  swap(int *ap, int *bp)
{
  int  tmp;

  tmp = *ap;  *ap = *bp;  *bp = tmp;
}
.....

{
  int  m = 10, n = 20;

  swap(&m, &n);
  printf("m = %d\nn = %d", m, n);
}
```

13

Arrays and Pointers (I)

- C language treats a variable of type *"array of T"* as *"pointer to T"*, whose value is the address of the first element of the array

  *char m[MAX], *cp;*

  cp = m; is equivalent to *cp = &m[0];*

- Array subscripting is defined in terms of pointer arithmetic:

char *cp, c[MAX]; int i;	
Array Notation	**Pointer Notation**
&c[0]	c
c[i]	*(c+i)
&c[i]	c+i
cp[i]	*(cp + i)

14

Arrays and Pointers (II)

- Consider an example:

  char c[5] = {'a', 'b', 'c', 'd', 'e'};
  *char *cp = c;*

- ***These are incorrect statements***

  c = cp; c++;

Array Element	Pointer Arithmetic	Pointer with a Subscript	Value
c[0]	*cp	cp[0]	'a'
c[1]	*(cp+1)	cp[1]	'b'
c[2]	*(cp+2)	cp[2]	'c'
c[3]	*(cp+3)	cp[3]	'd'
c[4]	*(cp+4)	cp[4]	'e'

15

15

Array as Function Arguments

- In a function, if an array is necessary to be a formal parameter, it can be declared using pointers. Thus, the following functions are equivalent:

```
/* The function uses an array */
int max(int a[], int length)
{
  int  i, maxv;

  for (i=1, maxv = a[0]; i<length; i++)
      if (a[i] > maxv)   maxv = a[i];

  return  maxv;
}
```

```
/* The function uses a pointer */
int max(int *a, int length)
{
  int  i, maxv;

  for (i=1, maxv = *a; i<length; i++)
      if (*(a+i) > maxv)   maxv = *(a+i);

  return  maxv;
}
```

16

16

Strings

- A ***string*** is a null-terminated array of characters. The null character **'\0'** indicates the end of a string
- Examples of string declarations and initializations:

  char str1[5] = {'a', 'b', 'c', 'd', '\x0'}; OR

  char str1[] = "abcd";

  *char *str2;* OR

  str2 = "abcdef";

- Here, two versions of a function that copies one string to another string, are presented

```
/* The string copying function  #1 */
void strcpy(char *to, char *from)
{
  while (*to = *from)  to++, from++;
}
```

```
/* The string copying function  #1 */
void strcpy(char *to, char *from)
{
  while (*to++ = *from++)  ;
}
```

17

17

Library String Functions

- The standard header file <string.h> contains prototypes for a number of functions for processing strings in C programs:

char s1[MAX], s2[MAX];	
Statement	*Result*
strlen("abc")	3
strcpy(s1, "string")	string
strncpy(s2, "temp", 2)	te
strcat(s1, s2)	stringte
strcmp(s1, s2)	-1
strncmp(s1+6, s2, 4)	0
strchr(s1, 't')	tringte
strrchr(s1, 't')	te

18

18

Sample Program (II)

```
/* These functions determine if a given   */
/*   string is a palindrome.              */
/*   Example: Madam! I'm Adam             */
#include <string.h>

#define  MAXSIZE  80     // Max. # of characters

void transform(char *raw, char *std)
{
  for( ; *raw; raw++)
    if(*raw >= 'a' && *raw <= 'z')    // Convert
       *std++ = *raw - 'a' + 'A';         // to uppercase
    else
        if((*raw >= 'A' && *raw <= 'Z') ||
           (*raw >= '0' && *raw <= '9'))
             *std++ = *raw;         // Copy letters & digits
    *std = *raw;
}
```

```
int test(char *str)
{
  char *left  = str;          // Beginning pointer
  char *right = str + strlen(str) - 1;  // Ending

  for(; left < right; left++, right--)
     if(*left != *right)
        return 0;    // False - not a palindrome

  return 1;  // True - yes! a palindrome
}

int palindrome(char *rawstr)
{
  char stdstr[MAXSIZE];

  transform(rawstr, stdstr);

  return test(stdstr);
}
```


19